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

Projeto Cooperativo no Ambiente Cave Baseado em Espaço Compartilhado de Objetos

Por

Sandro Sawicki

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

Prof. Dr. Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, dezembro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Sawicki, Sandro

Projeto Cooperativo no Ambiente Cave Baseado em Espaço Compartilhado de Objetos/ por Sandro Sawicki – Porto Alegre: PPGC da UFRGS, 2002.

156f.:il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós Graduação em Computação, Porto Alegre, BR – RS. Orientador: Reis, Ricardo Augusto da Luz.

1. Microeletrônica. 2. Ambientes Distribuídos. 3. Tecnologia Jini. 4. Tecnologia Java 5. Apoio ao Projeto de Circuitos Integrados. 6. Trabalho Cooperativo. 7. Microeletrônica. I. Reis, Ricardo Augusto da Luz. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Prof. Jaime Evaldo Fernsterseifer
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos A. Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Tenho muito a agradecer a várias pessoas que contribuíram direta ou indiretamente neste trabalho. Mas antes, gostaria de dizer que esse período de mestrado foi um dos mais legais que já passei. Pois através dele, conheci e convivi com pessoas maravilhosas, aprendi muitas coisas, em vários aspectos. Isso se tornou um aprendizado de vida que jamais esquecerei. Gostaria de agradecer em primeiro lugar aos meus pais, Mário e Nelda, aos meus avós Olinda e Reiholdo, que, além de rezarem por mim, sempre incentivaram e confiaram nos meus objetivos. Agradeço também ao meu irmão Jackson que de uma forma toda particular também estava torcendo pelo meu trabalho.

Agradeço ao meu orientador Prof. Ricardo Reis, uma pessoa muito especial, preocupada com a justiça social de nosso país, justo, além de ser um grande amigo e companheiro. Obrigado por me proporcionar anos de muita alegria junto ao convívio do grupo de microeletrônica da UFRGS.

Aos amigos da sala 211, dentre eles Lucas Brusamarelo, Emerson B. Hernandez, Marcio Bystronski, Luciano Agostini, Rafael Krapf, Diogo Zandonai, Alex Dias Gonzales, Alex Panato, Fábio Martinazzo entre os outros. Mas, um agradecimento especial a Lisane Brisolara, minha companheira de projeto, a qual, assim como eu, perdeu muitas noites de sono para resolver problemas na criação das estruturas de classe na compilação dos programas. Um agradecimento aos professores, santarosenses como eu, do Instituto de Informática, Prof. Renato Ribas e Prof. André Reis, que sempre incentivaram o meu trabalho.

Um agradecimento ao meu grande amigo Leandro Soares Indrusiak, que, assim com o Prof. Reis, foi o mentor do projeto Cave. Este trabalho deve muito a ele, pois sempre contribuiu com críticas, sugestões e idéias. Espero que minha contribuição sirva de alavanca para projetos colaborativos no âmbito do Ambiente Cave. Um agradecimento à Cassius Frosi Lenzi, um grande amigo e companheiro que sempre me deu forças. Várias conversas sobre inúmeros assuntos fizeram dele um convívio muito divertido. Acho que ele vai entrar até para o meu partido (PDSD - Partido Do Stress Desproporcional ✍). Um agradecimento especial à Marcelo Antunes, vulgo Pir (?r).

Aos caboclos paraenses que conheci na UFRGS, Abraham Lincoln Rabelo e Edson Eustáchio, amigos inesquecíveis ao qual tenho um grande carinho. Lembro das tardes de Domingo, onde após um grande carreteiro, íamos ao Gigante da Beira-Rio assistir o colorado gaúcho (período de alegrias e também de tristezas). Um agradecimento a minha afilhada Alessandra Dahmer (Duda), agora noiva do caboclo Lincoln (afilhado), a qual passamos momentos divertidos, regados novamente a muito carreteiro! felicidades... sou até padrinho do casamento! Um agradecimento especial também à Rejane Frozza, a qual trocávamos muitas idéias nos corredores dos laboratórios.

Agradeço também a Deus, que sempre esteve ao meu lado, e ao Odaylson Eder, amigo e colega inesquecível que sempre acreditou e apostou no meu trabalho em todos os momentos. Sem eles, provavelmente não chegaria até aqui.

Sumário

Lista de Abreviaturas 8

Lista de Figuras..... 10

Lista de Tabelas 12

Resumo 13

Abstract 14

1 Introdução 15
1.1 Objetivos 16
1.2 Organização do trabalho 17
2 Ambiente Cave 21
2.1 Introdução 21
2.2 Ambiente Cave: proposta original 21
2.2.1 World Wide Web com ambiente de projeto 21
2.2.2 Arquitetura do ambiente de Projeto Cave original 22
2.3 Ambiente Cave2: proposta atual 25
2.3.1 Acesso às ferramentas 29
2.4 Alternativas para implementação do repositório de dados 30
2.4.1 RDBMS (Relational Database Management System) 30
2.4.2 OODBMS (Object-oriented Database Management System) 30
2.4.3 Modelo baseado em Tupla-Space 30
2.5 Protótipos de ferramentas CAD baseadas na arquitetura Cave2 31
2.5.1 CIF2VRML 31
2.5.2 JALE (Java Layout Editor) 32
2.5.3 BLADE (Block and Diagram Editor) 33
2.5.4 Homero (VHDL Editor) 34
2.6 Cooperação entre as ferramentas de CAD do Ambiente Cave2 35
2.7 Resumo do Capítulo 36
3 Trabalho Cooperativo 37
3.1 Introdução 37
3.2 CSCW (Computer Supported Cooperative Work) 38
3.3 Tecnologias de Suporte 38
3.3.1 Redes de Computadores 38
3.3.2 Banco de Dados 39
3.3.3 Sistemas Operacionais 39
3.3.4 Aspectos de Interface Homem-Máquina 40
3.3.5 Agentes Inteligentes 41
3.3.6 Hipertextos e Multimídia 41

3.4 Ambientes colaborativos 41
3.4.1 Comunicação 41
3.4.2 Negociação 43
3.4.3 Coordenação 44
3.4.4 Percepção 44
3.4.5 Compartilhamento 44
3.5 Sistemas colaborativos 45
3.5.1 Comunicação como necessidade 45
3.5.2 Identificadores de usuários 46
3.5.3 Estados Compartilhados 46
3.5.4 Desempenho 46
3.6 Modelos de cooperação 46
3.6.1 Modelo de cooperação segundo Käfer 46
3.6.2 Modelo de cooperação segundo Nodine e Skarra 47
3.6.3 Modelo de cooperação segundo Iochpe 48
3.6.4 Modelo de cooperação proposto para o Collaborative Service 49
3.7 Ferramentas cooperativas 50
3.7.1 SEPIA – Structured Elicitation and Processing of Ideas for Authoring 50
3.7.2 PREP – Work in Preparation 51
3.7.3 QUILT 51
3.7.4 CoWeb 51
3.7.5 BSCW – Basic Support for Cooperative Work 52
3.7.6 DISKEDIT 52
3.7.7 MUT – (Multiuser Talk) & OCE – (Graphic Object Cooperative Editor) 53
3.7.8 WebDAV – Distributed Authoring and Versioning on the Web 53
3.7.9 ALLIANCE 53
3.7.10 GROVE 53
3.7.11 PROSOFT 54
3.8 Ferramentas cooperativas na área de EDA (Electronic Design Automation) 54
3.8.1 STAR 54
3.8.2 CollabTop 55
3.8.3 Quants 56
3.8.4 OmniFlow 56
3.9 Metodologias de cooperação 57
3.9.1 Metodologia WYSIWIS (What You See Is What I See) 57
3.9.2 Metodologia Pair-Programming 57
3.10 Resumo do Capítulo 59
4 Tecnologias Jini/Javaspaces 61
4.1 Introdução 61
4.1.1 JINI – (Java Intelligent Network Internet) 61
4.2 O Funcionamento da tecnologia Jini 65
4.3 Conexão unicast 66
4.4 Conexão multicast 66
4.5 Processo de discovery 67
4.6 Processo join 67
4.7 Processo de lookup 68

4.8 RMI (Remote Method Invocation) 68
4.9.1 Introdução 69
4.9.2 Arquitetura RMI 69
4.9.3 Camadas RMI 69
4.9 Javaspaces 71
4.10.1 Introdução 71
4.10.2 Modelo de aplicação Javaspaces 71
4.10.3 Persistência distribuída usando Javaspaces 72
4.10.4 Benefícios 73
4.10.5 Metas e exigências 73
4.10.6 Algumas operações 74
4.11 Transações Distribuídas usando Javaspaces 74
4.11.1 Introdução 74
4.11.2 Como se efetua uma transação – protocolo duas fases 75
4.11.3 ransações e Javaspaces 76
4.11.4 Participantes de uma transação 77
4.11.5 Usando a Transaction Manager 77
4.11.6 Um Exemplo de Transação 78
4.11.7 Tecnologia Javaspaces e Banco de Dados 79
4.11 Resumo do capítulo 80
5 Inserção de uma Abordagem Cooperativa para o Ambiente Cave 81
5.1 Motivação 81
5.2 Serviço Cooperativo Baseado em Conceitos de Orientação a Objetos 82
5.3 Armazenamento de objetos 82
5.4 Metodologia de Cooperação (Pair-Programming- PP) 83
5.5 Arquitetura do serviço de cooperação 84
5.5.1 TLS (Tabela de Localização de Serviços) 85
5.5.2 Modelagem da estrutura de classes usando UML 86
5.5.3 Modelo de Classes da Estrutura do Chat 87
5.5.4 Armazenamento dos projetos 88
5.5.5 Armazenamento dos participantes 89
5.5.6 Armazenando mensagens do chat 90
5.5.7 Comunicação entre objetos (projeto, participantes e chat) 90
5.5.8 Atualizando projetos 92
5.5.9 Transmitindo token de escrita 95
5.6 Casos de uso envolvendo Collaborative Service 96
5.7 Diagramas de estado envolvendo o Collaborative Service 99
5.8 Criação de pacotes 102
5.8.1 cave.collaborative.connect 103
5.8.2 cave.collaborative.chat 103
5.8.3. cave.collaborative.service 103
5.9 Contribuição deste trabalho para o Ambiente Cave 104
5.10 Resumo de capítulo 105
6 Estudo de Caso 1: Blade, Um Editor de Diagramas Esquemáticos Hierárquico 107
6.1 Introdução 107

6.1.1 Característica da Ferramenta Blade 107
6.2 Integração do Collaborative Service com a ferramenta Blade 108
6.3 Demonstrando a aplicação com o Collaborative Service 113
6.4 Modelos de Visualização 121
6.5 Resumo do Capítulo 122
7 Estudo de Caso 2: Homero, Um Editor de Descrições Textuais voltado a programação VHDL, Linguagem C e Verilog 123
7.1 Introdução 123
7.1.1 Características da Ferramenta 123
7.2 Integração da ferramenta Homero com o Collaborative Service 123
7.3 Resumo do capítulo 129
8 Conclusão 131
8.1 Contribuições do Trabalho 132
8.2 Trabalhos Futuros 133
Anexo 1 Tutorial de Inicialização das Tecnologias Jini/Javaspaces 137
Bibliografia 137

Lista de Abreviaturas

ACID	Atomicidade, Consistência, Isolamento, Durabilidade.
ACM	Association for Computing Machinery
API	Aplication Programming Interface
AWT	Abstract Windowing Toolkit
CAD	Computer Aided Design
CAVE	Computer Aided Virtual Environment
CI	Circuito Integrado
CIF	Caltech Intermediate Format
CSCW	Computer Supported Computer Work
EAD	Eletronic Aided Design
EDA	Eletronic Design Automation
FTP	File Transfer Protocol
GME	Grupo de Microeletrônica
GUI	Graphical User Interface
HDL	Hardware Description Language
HTML	HyperText Markup Language
HTTP	HiperText Transfer Protocol
ID	Identifier
JDK	Java Development Kit
JFC	Java Foundation Class
JMF	Java Media Framework
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LAN	Local Area Network
OO	Object-Oriented
OODBMS	Object-Oriented Database Management System
ORB	Object Request Broker
PP	Pair-Programming
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTL	Register- Transfer Level
SDK	Software Development Kit
SDL	System Description Language
SMTP	Simple Mail Transfer Protocol
TCP	Transfer Control Protocol
TLS	Tabela de Localização de Serviços
TTL	Time-To-Live
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Uniform Resourse Locators

VHDL	Very High Integrated Circuits HSIC Hardware Description Language
VLSI	Very Level System Integration
VRML	Virtual Reality Modeling Language
WWW	World Wide Web
XML	Extensible Markup Language
XP	Extreme Programming

Lista de Figuras

FIGURA 1. 1 - Organização da dissertação....19
FIGURA 2. 1 - Distribuição de recursos entre cliente e servidor no primeiro grupo....22
FIGURA 2. 2 - Distribuição de recursos entre cliente e servidor no segundo grupo....23
FIGURA 2. 3 - Arquitetura original - modelo cliente/servidor [IND 98]....23
FIGURA 2. 4 - Cave2 - projeto da interface com o usuário....25
FIGURA 2. 5 - Arquitetura Cave2....26
FIGURA 2. 6 - Interação de usuários com o *Framework Server* [IND 99a]....27
FIGURA 2. 7 - Cooperação no Ambiente Cave2....28
FIGURA 2. 8 - CIF2VRML interface gráfica com o usuário....30
FIGURA 2. 9 - Célula de um circuito integrado gerado pelo CIF2VRML....31
FIGURA 2. 10 - *Jale*: Interface com o Usuário....32
FIGURA 2. 11 - *Blade - Block and Diagram Editor*....33
FIGURA 2. 12 - Homero - Editor VHDL....34
FIGURA 3. 1 - Composição de um sistema colaborativo, segundo [FAR 98]....44
FIGURA 3. 5 - Colaboração entre várias sessões [KON 99]....54
FIGURA 3. 6 - Eventos gerados pelo *broadcast*....55
FIGURA 4. 1 - Camadas da tecnologia *Jini*....61
FIGURA 4. 2 - Registrando serviço do *Lookup Service*....62
FIGURA 4. 3 - Requisição de serviços em uma rede *Jini*....63
FIGURA 4. 4 - Camadas RMI....68
FIGURA 4. 5 - Fase *precommit*....73
FIGURA 4. 6 - Fase *commit*....74
FIGURA 4. 7 - Transações entre diferentes *Javaspaces*....74
FIGURA 5. 1 - Interação entre os serviços e o projetista....83
FIGURA 5. 2 - Tabela de localização de serviços....83
FIGURA 5. 3 - Diagrama de classes: armazenamento de projetos e participantes....84
FIGURA 5. 4 - Diagrama de classes: *Chat*....86
FIGURA 5. 5 - Classes do projeto a serem armazenadas....87
FIGURA 5. 6 - Classes de participantes a serem armazenadas....87
FIGURA 5. 7 - Estrutura de classes do *chat*....88
FIGURA 5. 8 - Objetos referentes a um bloco de projeto....89
FIGURA 5. 9 - Blocos de projeto....90
FIGURA 5. 10 - Atualizando o projeto....91
FIGURA 5. 11 - Atualizando os projetistas participantes....91
FIGURA 5. 12 - Atualizando as mensagens do *chat*....92
FIGURA 5. 15 - Caso de uso envolvendo projetistas....94
FIGURA 5. 16 - Caso de uso da requisição do *Collaborative Service*....95
FIGURA 5. 17 - Caso de Uso: cenário após a conexão....95
FIGURA 5. 18 - Caso de uso: cenário após abertura ou criação de um projeto....96
FIGURA 5. 19 - Estados da criação de um novo projeto....97
FIGURA 5. 20 - Estados da recuperação de um projeto....98
FIGURA 5. 21 - Estados da atualização do projeto....99
FIGURA 5. 22 - Transmissão de *tokens*....100
FIGURA 5. 23 - Pacotes que compõem o *Service Collaborative*....101

FIGURA 6. 1 - Interfaces intermediárias.....107
FIGURA 6. 2 - Menu de cooperação inserido na ferramenta *Blade* 108
FIGURA 6. 3 - Criação de um novo projeto cooperativo.....109
FIGURA 6. 4 - Abrindo um projeto compartilhado 109
FIGURA 6. 5 - Interação entre dois projetistas através do *chat*.....110
FIGURA 6. 6 - Cave GUI e Tool Launcher.....111
FIGURA 6. 7 - Pontos após a inserção do *Collaborative Service*.....112
FIGURA 6. 8 - Etapas da comunicação com o repositório de dados 113
FIGURA 6. 9 - Representação dos níveis hierárquicos em um *full-adder* 114
FIGURA 6. 10 - *Full-Adder* Nivel 1: interação entre dois projetistas.....115
FIGURA 6. 11 - Nível 2: *Half-Hader* instanciada 116
FIGURA 6. 12 - Nível 3: EX-OR inacabada 117
FIGURA 6. 13 - Requisição de escrita.....118
FIGURA 6. 14 - Transferência da permissão de escrita 119
FIGURA 7. 1 - Ferramenta Homero após a inserção dos componentes de cooperação 123
FIGURA 7. 2 - Objetos envolvidos na cooperação.....124
FIGURA 7. 3 - Interação de quatro instâncias da ferramenta Homero sob um bloco VHDL 125
FIGURA 7. 4 - Interação sob um bloco VHDL 126
FIGURA 7. 5 - Projeto de uma ALU em VHDL 127

Lista de Tabelas

TABELA 3. 1 - Taxonomia espaço temporal segundo [UNA 91] 42

Resumo

Este trabalho apresenta o módulo *Collaborative Service*, uma extensão do ambiente Cave, desenvolvido para suportar conceitos de trabalho cooperativo no projeto de circuitos integrados. Esta extensão por sua vez, é baseada na metodologia *Pair-Programming* e nas tecnologias *Jini* e *Javaspaces*.

O módulo *Collaborative Service* foi desenvolvido para auxiliar a continuidade do processo de desenvolvimento de circuitos integrados complexos, inserindo uma dinâmica de grupo através da extensão de *Pair-Programming* para máquinas remotas. Esse modelo permite que dois ou mais projetistas interajam em um mesmo projeto ou blocos de projeto, independente de suas localizações geográficas e tipos de plataformas de *hardware/software*. Ele foi projetado para ser genérico e essa característica o torna capaz de suportar as ferramentas de CAD, atuais e futuras, do ambiente Cave (um *framework* de apoio ao projeto de circuitos integrados).

Como estudo de caso, foram utilizadas duas ferramentas do Ambiente Cave. O primeiro caso mostra uma cooperação em nível de descrições gráficas, representada pela ferramenta *Blade*, um editor de esquemáticos hierárquico. O segundo caso foi representado pelo editor de descrições textuais (VHDL, Verilog e Linguagem C), chamado Homero.

No estudo de caso com a ferramenta *Blade* foi demonstrado que a cooperação proposta por esse modelo pode atuar sob diferentes níveis de hierarquia de projeto, além de suportar a interação de inúmeros projetistas em um mesmo bloco. Na ferramenta Homero, demonstrou-se a cooperação em nível de descrições textuais, representados por (códigos) projetos VHDL acrescidos da participação de vários projetistas. Com esses exemplos, foi possível demonstrar as estratégias de percepção e comunicação com os projetistas, além de descrever a criação de blocos de projeto de uma forma cooperativa. Como contribuição desse trabalho, acrescenta-se ao Ambiente Cave mais um recurso para o projeto de circuitos integrados. Nesse sentido, grupos de projetistas podem projetar um sistema ou circuito integrado de forma cooperativa utilizando-se das funcionalidades desse modelo.

Palavras-Chave: Ferramentas CAD, *Framework*, Tecnologia Java, Tecnologia Jini, Trabalho Cooperativo, Ambientes Distribuídos, Microeletrônica.

TITLE: “COLLABORATIVE DESIGN IN THE CAVE ENVIROMENT BASED ON SHARED OBJECT SPACES”

Abstract

This work presents the Collaborative Service module developed to support collaborative work. This module is an extension of the Cave environment which is a framework that supports design of integrated circuits. This module is based on Pair-Programming methodology and Jini/Javaspaces technologies.

The collaborative service module was developed to aid the complex development integrated circuit design, inserting the Pair-Programming methodology extension in remote machines. This unit allows two or more designers located in different places to work in the same design blocks. It was designed to be generic and this characteristic makes the Cave Environment able to support current and future CAD tools.

As case studies, two tools from the Project Cave were used. The first one, a hierarchical diagram editor named Blade, shows collaboration in graphical description level. The second one, a high level textual editor (VHDL, Verilog and C) entitled Homero, shows collaboration in textual level.

In the Blade case study, it was demonstrated that collaboration proposal for this model can act under different levels of design hierarchy, and support interaction between several designers in a same circuit block. In the Homero case, collaboration at textual level descriptions has demonstrated, represented by VHDL design, the increase of participation among several designers. With these examples, it was possible to demonstrate an addiction of perception and communication with the designers, and to describe the creation of design blocks in a cooperative form. The main contribution of this work is the new Collaborative Service module attached to the Cave Environment as a new resource for the design of ICs. In this approach, groups of designers can design a system in a cooperative way using the features of this model.

Keywords: CAD Tools, Framework, Java Technology, Jini Technology, Cooperative Work, Distributed Environment, Microeletronics

1 Introdução

Avanços tecnológicos têm permitido a construção de circuitos integrados cada vez mais complexos. Com isso, a implementação de novas ferramentas de apoio aos seus projetos e novas metodologias para o seu desenvolvimento são fatores determinantes para a continuidade do aumento da complexidade na concepção desses circuitos.

Proposto por Indrusiak e Reis [IND 97], o ambiente Cave visa suprir a necessidade de migração por parte do projetista para diferentes plataformas de *hardware/software*, além de integrar diversas ferramentas de CAD, cada uma delas voltada a uma ou mais etapas na concepção de circuitos. A estrutura desse ambiente evoluiu para um modelo de dados baseado fortemente em objetos (evolução seção 2.2), com o intuito de direcionar o seu foco de trabalho na pesquisa envolvendo *projetos cooperativos*. A incorporação de um recurso baseado em conceitos de trabalho cooperativo é facilmente justificado, principalmente para um ambiente de apoio ao projeto, como é o caso do Ambiente Cave.

Dentre algumas vantagens oferecidas pelo trabalho cooperativo é possível destacar que:

- uma equipe pode melhorar as decisões se comparada a um único projetista, já que um grupo possui mais informações e conhecimentos, gerando menor probabilidade de se cometerem erros;

- a existência de uma harmonia maior entre o projeto e seus projetistas torna o projeto mais consistente, evitando informações errôneas entre a equipe de desenvolvimento;

- há maior eficiência nas ações: uma equipe pode tornar o projeto mais eficaz na implementação dos resultados quando seus membros participam das decisões, aumentando a aceitação e a compreensão do trabalho proposto;

- os projetistas podem trabalhar em um mesmo projeto, simultaneamente, mesmo estando distantes geograficamente;

- há diminuição dos custos com deslocamento no caso de uma equipe de projetistas distribuída;

- a familiaridade dos projetistas como o WWW diminui a carga cognitiva do ambiente de projeto. Isto é, as tarefas são realizadas mais eficiente e rapidamente pelo usuário, visto que não precisa aprender a lidar com a interface do ambiente.

Neste sentido, é possível perceber na literatura que o trabalho em equipe auxiliado por computador voltado à concepção de circuitos integrados é algo recente. Nesse contexto, surgiram algumas propostas de utilização da *Web* como mecanismo de integração de ferramentas cooperativas para diversos fins, inclusive na área de EDA *(Electronic Design Automation)*, como em [LAV 97, BRG 2001, KIR 2001 e LAV 2000]. Os detalhes desses e de outros trabalhos estão descritos na seção 3.7.

Outro fator importante para o interesse do projeto Cave em constituir-se como um ambiente de projeto cooperativo, é a expansão no mercado de projeto de circuitos integrados, o que levou à criação de novos pólos de trabalho, muitas vezes distantes geograficamente. Desse modo, projetistas e organizações se depararam com a barreira da falta de interatividade entre núcleos de projeto. Em outras palavras, a questão é: como fazer para que projetistas geograficamente distantes interajam em um mesmo projeto, sem a necessidade de deslocamentos?

A resposta pode seguir da idéia de que o trabalho cooperativo consiste em fazer com que a produção de um grupo seja maior do que a produção de cada um dos seus membros individualmente. Isso mostra que a produtividade em um projeto cooperativo aumenta significativamente e aponta o trabalho em grupo como um importante fator para a sobrevivência das empresas no mercado. No escopo dessa idéia se insere o projeto Cave que, através do módulo *Collaborative Service*, busca diminuir a distância entre grupos de projeto, criando uma proposta de trabalho cooperativo em projetos de CIs [SAW 2002, SAW 2002a, SAW 2002b].

A motivação deste trabalho surgiu da necessidade de criar novas ferramentas e metodologias de projeto para acompanhar o constante aumento da complexidade dos circuitos integrados. A abordagem apresentada neste trabalho cria um modelo de cooperação genérico o suficiente para suportar a cooperação sobre diferentes modelos de dados, visando sua adaptação em diferentes ferramentas de CAD do ambiente Cave. Este trabalho baseia-se também em conversas informais e no *feedback* dado por empresas do ramo de projeto de CIs quando investigadas sobre seus interesses quanto à cooperação entre projetistas. Observou-se nestas pesquisas que a cooperação é desejada em níveis de abstração mais altos, os quais utilizam representações gráficas no sentido de facilitar a troca de informações entre os projetistas. Entretanto, o Ambiente Cave é composto tanto por ferramentas de descrições gráficas quanto descrições textuais. Isso motiva o desenvolvimento de um módulo que pode auxiliar a cooperação em ambos os níveis.

1.1 Objetivos

O objetivo deste trabalho é propor, modelar e implementar um módulo genérico e independente de plataforma, de modo que possibilite a interação entre projetistas, mesmo distantes geograficamente em nível de projetos e blocos de projeto, em um ambiente de apoio ao projeto de circuitos integrados [SAW 2001, SAW 2001a, SAW 2002, SAW 2002a e SAW 2002b]. A tecnologia Jini, descrita no capítulo 3, é

utilizada como infra-estrutura, pois disponibiliza diversos serviços que auxiliam na validação dessa arquitetura [EDW 99, FRE 99 e MOR 2000]. Também é incorporado ao trabalho o estilo de programação chamado *Pair-Programming* [WILL 99, WIL 2000, WIL 2000a e WILL 2000b]. Este estilo é utilizado como metodologia de cooperação e está inserido no escopo da proposta de cooperação.

Esta dissertação envolve também a pesquisa e implementação de um meio de armazenamento persistente de dados. Esse ponto é de extrema importância, pois se entende que um espaço persistente de informações compartilhadas é um dos grandes problemas de um projeto cooperativo. Na verdade, esta questão é fundamental, pois o uso de um meio de armazenamento persistente fornece suporte ao trabalho cooperativo pelo fato de servir como repositório de informações e permitir o acesso, com facilidade, a essas informações.

A idéia de cooperação proposta para esse serviço é estender a metodologia *Pair-Programming* para que seja executada em máquinas remotas. Na metodologia *Pair-Programming* clássica, a cooperação é realizada através de pares de pessoas trabalhando em uma só máquina, tal que uma pessoa detinha o direito de escrita (teclado), enquanto a outra poderia contribuir com sugestões e idéias. A extensão descrita anteriormente envolve tornar a metodologia *Pair-Programming* distribuída. Com isso, projetistas de circuitos integrados poderiam, por exemplo, projetar um microprocessador de forma cooperativa, dividindo-o em blocos de projetos, inserindo outros projetistas participantes que poderiam interagir, projetar, dar opiniões, observar tudo o que está acontecendo no projeto (percepção), mesmo que estejam geograficamente distantes.

Seguindo a metodologia adotada, na proposta desse trabalho, somente um projetista detém o direito de escrita, e é chamado de *escritor*, os demais participantes são chamados de *ouvintes*. A permissão de escrita poderá ser requisitada pelo ouvinte ao escritor. Com isso, cada um dos projetistas pode contribuir, tanto com opiniões quanto com alterações diretas no projeto. A cada atualização de projeto executada pelo projetista escritor, serão disparadas notificações aos demais projetistas envolvidos, que por sua vez saberão da situação do projeto.

O modo de percepção dos projetistas e a sua modalidade de permissão serão atualizadas e repassadas a todos os participantes da sessão de projeto, sempre que uma modificação acontecer. Quando a escrita for autorizada para algum projetista, essa modificação na percepção será propagada aos demais projetistas envolvidos. A troca de idéias entre projetistas é realizada através de um *chat* de texto, implementado utilizando-se das tecnologias Jini/Javaspaces. Pesquisas estão sendo realizadas dentro do GME (Grupo de Microeletrônica da UFRGS) a fim de estender o *chat* de texto para uma comunicação através de áudio assíncrono.

1.2 Organização do trabalho

Com o intuito de validar essa proposta, o presente trabalho está organizado da seguinte forma:

O **capítulo 2** descreve o *Framework* Cave, mostrando sua estrutura. São ilustradas as ferramentas que o compõem, sua forma de acesso, e como novas ferramentas podem ser inseridas no ambiente. Esse capítulo aborda também sua evolução, da etapa inicial baseada em hiperdocumentos à atual, focando uma estrutura baseada em objetos.

O **capítulo 3** traz um estudo bibliográfico sobre o trabalho cooperativo, descrevendo conceitos e aplicações. Abordada também o estado da arte de sistemas cooperativos em geral, tanto em trabalhos realizados no Instituto de Informática da UFRGS, quanto em trabalhos envolvendo a área de EDA no contexto nacional e internacional.

O **capítulo 4** enfoca a tecnologia adotada para o armazenamento dos projetos. Descreve, também, seus mecanismos de acesso, armazenamentos, notificações, transações, acesso a serviços, dentre outros. A arquitetura desse modelo também é detalhada nesse capítulo.

O **capítulo 5** descreve uma proposta de utilização de trabalho cooperativo na área de EDA usando como estudo de caso o *framework* Cave e suas ferramentas. Além disso, relata a modelagem utilizada para descrever o funcionamento do serviço de cooperação, abordando as estratégias de acesso a dados, notificação de projetistas, dentre outros. Todo o modelo baseia-se na notação UML *(Unified Modeling Language)*.

O **capítulo 6** utiliza outra ferramenta como objeto de estudo de caso para a validação do serviço de cooperação. Essa ferramenta é conhecida como *Blade*, e é um editor de esquemáticos hierárquico. O intuito desse estudo de caso é demonstrar que o serviço de cooperação proposto foi implementado de forma genérica. Com isso, torna-se possível suportar tanto ferramentas de descrições textuais, quanto ferramentas gráficas. A demonstração de suas funcionalidades também é descrita nesse capítulo.

O **capítulo 7** relata um estudo de caso utilizando a ferramenta Homero. Trata-se de um editor de descrições textuais utilizado para descrever programas em VHDL, Linguagem C e Verilog.

É importante ressaltar que nos capítulos 6 e 7, são demonstradas todas as funcionalidades do serviço de cooperação. Dentre elas, a percepção dos participantes do projeto, assim como as permissões de acesso para os projetos e a comunicação através do *chat* de texto.

O **capítulo 8** encerra o trabalho com as conclusões sobre a proposta de cooperação em ambientes de apoio a projeto de circuitos integrados. Inclui também a descrição de novas idéias e trabalhos futuros, assim como a contribuição deixada para o projeto Cave. A figura 1.1 ilustra a organização do texto da dissertação, associando etapas da pesquisa aos capítulos.

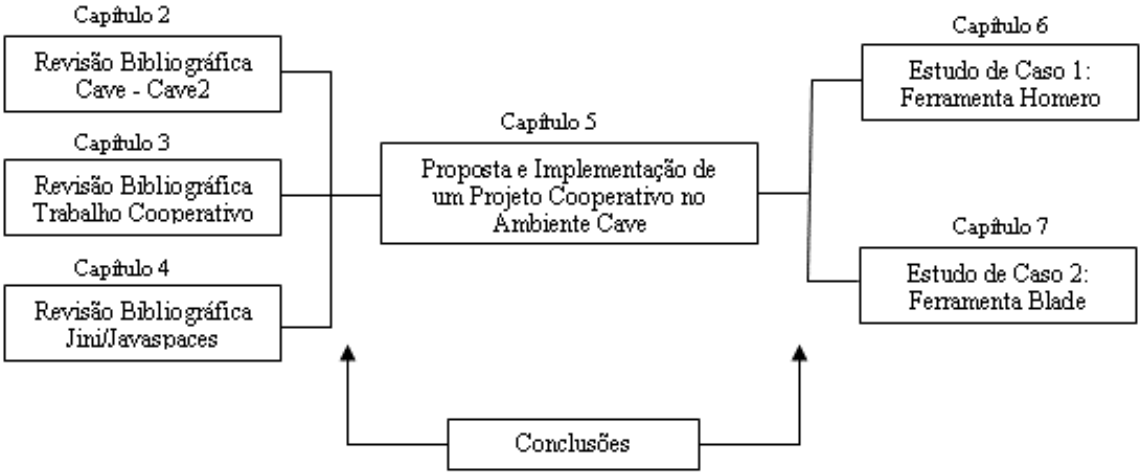


FIGURA 1. 1 - Organização da dissertação

2 Ambiente Cave

2.1 Introdução

Proposto por Indrusiak e Reis [IND 97], o Projeto Cave discute um modelo para integração de ferramentas de CAD em um único ambiente – formando um *framework* [BAR 92, WAG 94], também conhecido pela expressão da língua inglesa *CAD Framework* cujo objetivo é acelerar o processo de concepção dos circuitos através da automatização de tarefas. Isso livra o projetista de tarefas como a administração de recursos distribuídos, o armazenamento de arquivos, entre outros.

Um ambiente de projeto de circuitos integrados é formado por diversas ferramentas, cada uma delas associada a uma ou mais etapas da concepção do circuito. Essas ferramentas podem ter grande interação com o usuário ou podem executar tarefas sem que o usuário perceba.

Este capítulo mostra a evolução do ambiente Cave para Cave2 [IND 98, IND 98b, IND 98c, IND 99, IND 99a]. Baseado inicialmente em hiperdocumentos, atualmente, utiliza uma nova abordagem, enfatizando paradigmas de orientação a objetos.

2.2 Ambiente Cave: proposta original

O projeto Cave, em sua primeira versão discutida em [IND 98], propunha um *framework* baseado no ambiente *World Wide Web*, doravante WWW. Seu autor descreve que, ao utilizar o *World Wide Web* como base para o ambiente de integração de ferramentas, muito trabalho é poupado, uma vez que grande parte da interface gráfica e do controle de rede do *framework* já está implementada. No caso de uma equipe de projeto distribuída, a facilidade de acesso ao WWW também é uma grande vantagem.

O objetivo principal desse trabalho era lançar a idéia de uma nova arquitetura para a integração de ferramentas de apoio ao projeto de circuitos integrados, visando com isso, reduzir o tempo perdido pelo projetista na administração dos recursos distribuídos, bem como o aprendizado do uso das interfaces de integração. Essa abordagem explorava o modelo cliente-servidor, estendendo ao cliente a tarefa do processamento da informação, fortemente baseada no ambiente WWW, no qual o armazenamento de dados era centrado em sistema de arquivos.

2.2.1 *World Wide Web* com ambiente de projeto

Por definição, WWW é um ambiente independente de plataforma, o que já o qualifica como suporte para um ambiente de projeto que pretende englobar recursos distribuídos em diferentes tipos de *hardware/software*. Além disso, a WWW, conseguiu

uma enorme abrangência, alcançando todos os grandes centros de desenvolvimento tecnológico. Isso se deve, em grande parte, à sua simplicidade de uso, que permite fácil interação usuário-máquina-rede. Por já contar com a familiaridade dos usuários, a WWW diminui a carga cognitiva do ambiente de projeto, ou seja, as tarefas são mais eficientemente e rapidamente realizadas pelo usuário, visto que não precisa aprender a lidar com a interface do ambiente antes de realizar o seu trabalho [IND 98].

Além da abrangência e da grande familiaridade obtida por parte dos usuários, a WWW traz outras vantagens para abrigar um ambiente de apoio ao projeto de circuitos integrados. A simplicidade de construção da estrutura do ambiente é uma delas, pois é toda baseada em hiperdocumentos – um método flexível de acesso à informação armazenada em uma rede de nós interligados [BAL 93] – criados com a linguagem HTML. Outra vantagem da estruturação em hiperdocumentos está na facilidade de integração de material educacional e informativo com ferramentas de CAD, pois pode ser implementada através de simples ligações hipertextuais. Isso facilita o trabalho do projetista, que pode alterar entre ambiente de trabalho, *help on-line* e tutoriais utilizando uma única interface.

A arquitetura WWW permite que seja transparente para o usuário a distribuição de recursos em rede. Isso quer dizer que o usuário pode desconhecer em qual máquina conectada à rede – e em que tipo de plataforma de *hardware* - estão os recursos por ele acessados. Em se tratando de um ambiente de apoio ao projeto de circuitos integrados isso é bastante útil, pois no fluxo de desenvolvimento de um projeto, etapas computacionalmente diferentes – como, por exemplo, simulação elétrica (requer processamento numérico intenso) e edição de leiaute (requer processamento gráfico) - podem ser processadas em diferentes máquinas da rede. Além destas características, o WWW agrega ainda ao ambiente de projeto as possibilidades de trabalho cooperativo e aceso remoto.

2.2.2 Arquitetura do ambiente de Projeto Cave original

A arquitetura original do Projeto Cave é baseada na distribuição de recursos entre os lados *cliente* e *servidor* da rede. Essa integração de ferramentas segue dois grupos de modelos.

O primeiro grupo engloba ferramentas que requerem do projetista o uso intenso de interface gráfica, como editores de esquemáticos [HER 2000, BRI 2002], editores de *layout* [GRA 99, IND 97] e de ferramentas gráficas de síntese de alto nível. Todas essas ferramentas são implementadas utilizando a linguagem Java [JAV 2000, SUN 99, SUN 99a, DEI 2001] e anexadas a um hiperdocumento. Quando o usuário do ambiente requisita uma ferramenta desse grupo ao servidor, tanto a ferramenta quanto o hiperdocumento são transmitidos pela rede à máquina cliente e materializados na tela através do *web browser*. O *web browser* fará a interface da ferramenta com o sistema de *hardware/software* da máquina cliente e fará todo o processamento requerido pela ferramenta, havendo completa desvinculação com o servidor. Novas conexões de rede poderão ser abertas, caso a ferramenta necessite de arquivos de configuração, bibliotecas

ou de outros arquivos. O modelo permite o armazenamento dos dados de projeto tanto no lado cliente quanto no lado servidor na forma de sistema de arquivos. Na figura 2.1. é possível observar a interação com o primeiro grupo.

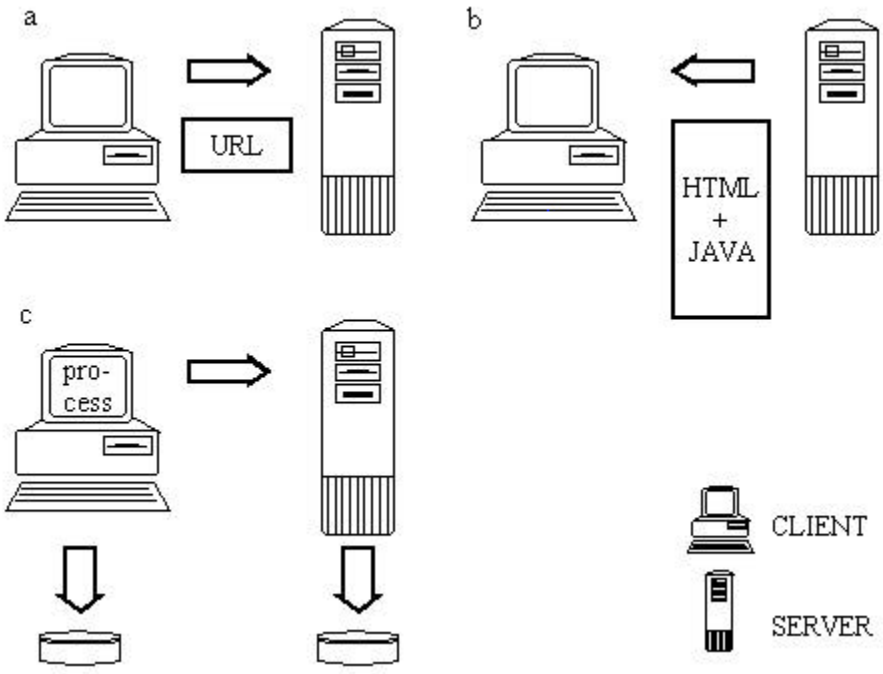


FIGURA 2. 1 - Distribuição de recursos entre cliente e servidor no primeiro grupo

[IND 98]

O segundo grupo, por sua vez, engloba ferramentas que não exigem uma interação muito grande com o usuário. Nesse caso, o usuário age como provedor de dados e analisador de resultados, enquanto o processamento desses dados é feito pela máquina servidora.

Quando um usuário requisita uma ferramenta do segundo grupo, o servidor envia um formulário no qual o usuário insere os dados a serem processados e parâmetros a serem passados para a ferramenta. Esses dados e parâmetros são enviados pela rede ao servidor, que os processa de acordo com os parâmetros, retornando os resultados para o usuário, que os analisa no *web browser*.

Nessa modalidade, o ciclo de interatividade é bem maior, já que, para qualquer modificação na entrada de dados, é necessária uma série de conexões de rede. Assim como no grupo anterior, o armazenamento dos dados pode se dar tanto no servidor, quanto no cliente também na forma de sistema de arquivos. A figura 2.2 a seguir, ilustra a interação com o segundo grupo.

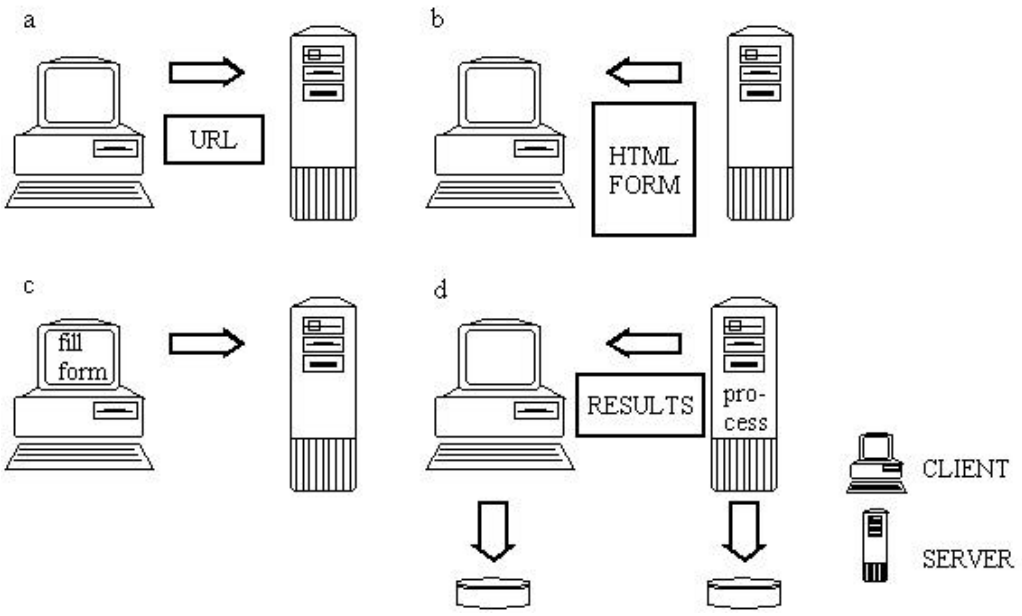


FIGURA 2. 2 - Distribuição de recursos entre cliente e servidor no segundo grupo

[IND 98]

Na seqüência, a figura 2.3 ilustra a abordagem original do Projeto Cave. As setas representam os protocolos de rede, através dos quais os dados e o código executável das ferramentas serão transferidos entre o servidor e o cliente. Nessa ilustração vê-se um único cliente conectado a um único servidor. Todavia, o modelo permite que um cliente troque dados com mais de um servidor, bem como possibilita um servidor atender a quantos clientes sua capacidade de processamento permitir.

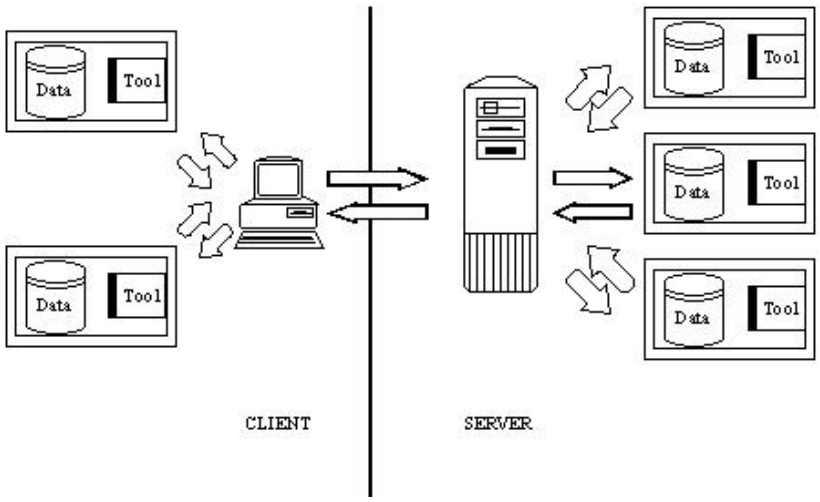


FIGURA 2. 3 - Arquitetura original - modelo cliente/servidor [IND 98]

2.3 Ambiente Cave2: proposta atual

Apesar de todos avanços gerados na arquitetura original do Projeto Cave, algumas modificações foram necessárias para acrescentar usabilidade e reusabilidade em seu *framework*. Com isso, o modelo orientado a objetos foi avaliado a fim de substituir-se o modelo de transmissão e armazenamento de dados baseado em hiperdocumentos e sistema de arquivos por um modelo baseado em objetos. Essa substituição foi planejada quando percebeu-se a necessidade de ter projetistas cooperando em um mesmo projeto através de uma rede.

O armazenamento de dados baseado em sistema de arquivos não se mostra muito eficiente no tratamento de acessos multi-usuário e armazenamento de dados complexos. Então, algumas alternativas de armazenamento de dados foram pesquisadas para que o tratamento dos dados não fosse a visão de apenas um projetista, mas sim uma possibilidade para que vários projetistas possam interagir com seus projetos de uma forma organizada. Em comparação com um sistema de arquivos, o modelo orientado a objetos permite uma maior integração com as ferramentas de projeto e um suporte eficiente para o armazenamento de dados. Atualmente, vale destacar, um sistema baseado em arquivos requer um intenso uso de conversores de formatos devido à falta de um modelo unificado.

A nova proposta tem duas características básicas em que podem resolver alguns problemas encontrados na versão anterior. A primeira característica é o uso intenso de GUIs ativas, anexadas em hiperdocumentos. Com essa característica, cada ferramenta teria uma interface dinâmica com o usuário. O usuário pode então, interagir com ambos os lados da rede. O fluxo de projeto pode ser modelado com uma alta interação entre as ferramentas, uma vez que as GUIs ativas podem carregar muitas ferramentas, permitindo o uso concorrente e o fácil compartilhamento dos dados. Isso sem a necessidade de sucessivas gerações de arquivos e *parsers*, geralmente demorados e propensos a erros.

Uma vez que a interface com o usuário e os problemas com a integração das ferramentas podem ser resolvidos com o uso de GUIs ativas, a segunda característica – a qual a estrutura de dados é baseada em paradigmas de orientação a objetos – é necessária para manter a consistência dos dados de projeto. Isso também controla e evita indesejáveis replicações e redundância de dados, problemas que surgem quando acessamos um projeto através de diferentes visões de todas as interfaces ativas das ferramentas.

A arquitetura do Cave2 tem o enfoque numa biblioteca de códigos orientado a objetos reutilizáveis e na extensão desta biblioteca através do conceito de *framework* utilizado na área de engenharia de software [PRE 92]. Segundo [JOH 92], um *framework* é um projeto de software reutilizável implementado como um conjunto de classes abstratas e o modo como suas instâncias colaboram. Baseado nesta abordagem, o Cave2

provê um conjunto extensível de primitivas GUI que são utilizadas para montar dinamicamente a interface entre o usuário e o ambiente de projeto.

Além disso, a abordagem de um *framework* ao nível de engenharia de software permite a modelagem dos dados usando objetos, diferentemente do modelo baseado em arquivos da primeira versão. As relações entre os objetos que irão representar os dados de projeto podem ser definidas usando padrões de software [GAM 2000].

Desta forma, a evolução pode ser definida como uma mudança de um modelo de arquivos centrado em documentos para uma arquitetura baseada em modelos orientados a objeto.

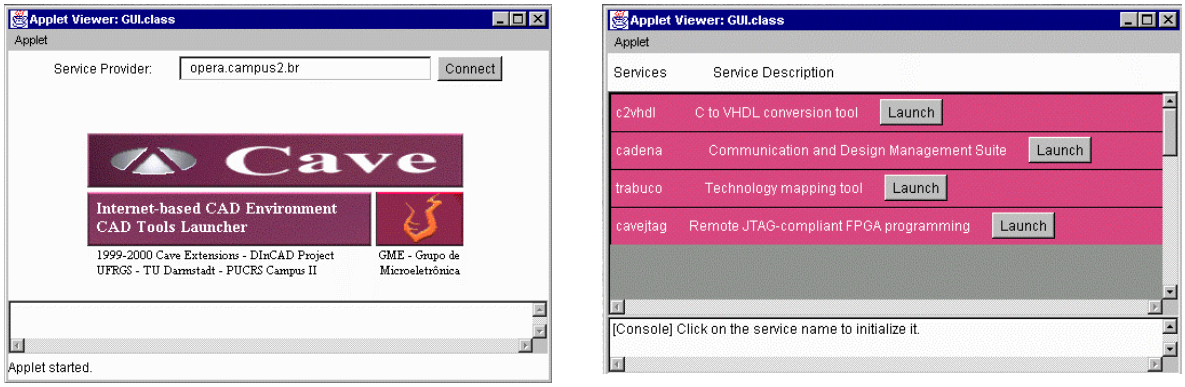


FIGURA 2. 4 - Cave2 - projeto da interface com o usuário

Um dos esforços do Cave2 é atualizar a interface dos usuários das ferramentas, bem como as interfaces de seu ambiente de projeto. Essas interfaces permitiriam ao usuário especificar as funcionalidades que ele necessite a partir do ambiente de projeto. Com isso, os objetos que ele necessita para realizar suas tarefas seriam instanciados.

A redefinição da estrutura do *framework* para suportar basicamente modelos de objetos é uma tarefa que demanda a utilização de mecanismos de comunicação com objetos distribuídos, representação de projetos orientados a objetos, simulação e acesso multi-usuário a dados de projeto. Para integrar uma nova ferramenta para o ambiente de projeto, o desenvolvedor da ferramenta deve implementar uma interface ativa com o usuário ativa para a ferramenta, que deve herdar das classes já implementadas e organizadas no *Framework* Cave.

Logo após, o desenvolvedor pode anexar a ferramenta no ambiente de projeto e usar as facilidades de comunicação do ambiente, sem precisar escrever um código adicional. A nova ferramenta também pode ser capaz de usar o modelo orientado a objetos para o armazenamento de projetos.

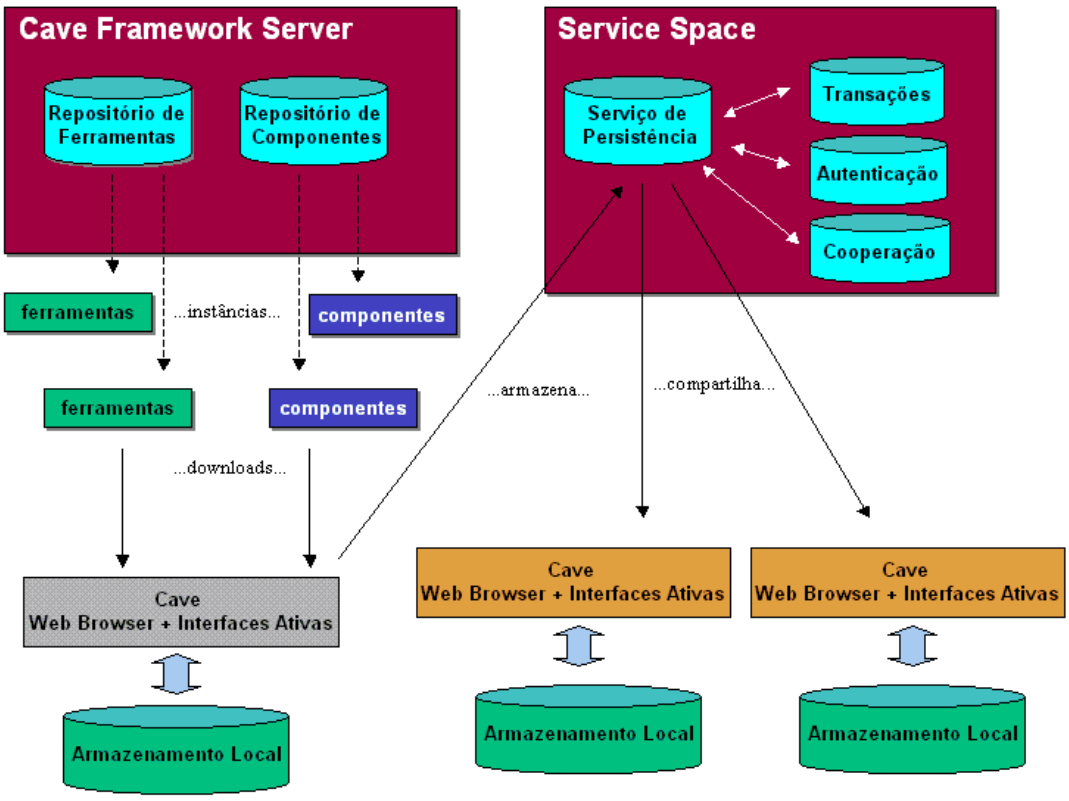


FIGURA 2. 5 - Arquitetura Cave2

A figura 2.5 ilustra a presença de dois servidores: o *Framework Server* e o *Service Space*, ambos enquadrados na nova arquitetura do ambiente Cave.

O *Cave Framework Server* é um *framework* de softwares reutilizáveis que possui um repositório de classes para criação de ferramentas de projeto e de classes para modelagem das representações dos dados de projeto, que também serão chamadas de primitivas de projeto. O *Service Space* atualmente disponibiliza serviços de persistência de dados, autenticação, integração com ferramentas externas e cooperação. O *Service Space* é um dos elementos mais importantes no *framework*, pois através dele são compartilhados os dados de projeto, permitindo a cooperação entre os projetistas. O *Service Space* é o foco deste trabalho e será apresentado com mais detalhes nos próximos capítulos.

O Cave2 pode ser executado sobre a Internet, usando-se o protocolo HTTP juntamente com *browsers* compatíveis com a linguagem Java, ou *standalone* através de uma conexão *socket*. A distribuição de recursos de projeto pode ser facilmente realizada, visto que a arquitetura suporta a cooperação entre vários *Framework Servers* e *Service Spaces*. Atualmente, porém, utiliza-se uma abordagem não redundante: tem-se um único *Framework Server* e um único *Service Space*. Assim, toda a representação de projetos e módulos de ferramentas de projeto estão armazenadas num único *Framework Server*, e

um protocolo interno é usado para o compartilhamento de tais recursos. Com isso, todos os serviços são disponibilizados num único *Service Space* e podem ser localizados através de um mecanismo de *lookup*, descrito em [SAW 2002], é utilizado como um intermediário entre o cliente e as aplicações.

O usuário pode selecionar o servidor ao qual deseja se conectar, entrando com seu *username* e senha. Após a autenticação do usuário, é apresentada uma janela chamada de *Tool Launcher*. O *Tool Launcher* possui uma lista das ferramentas disponíveis no servidor selecionado, podendo esta lista ser personalizada para cada usuário. Ao selecionar uma ferramenta da lista, uma conexão *socket* é aberta e uma instância da ferramenta é criada na máquina cliente, possibilitando o seu uso. A representação da interação do usuário com o *Framework Server* está ilustrada na figura 2.6. Pode-se observar, através dessa representação, como funciona o processo de invocação das ferramentas e o uso das primitivas de projeto.

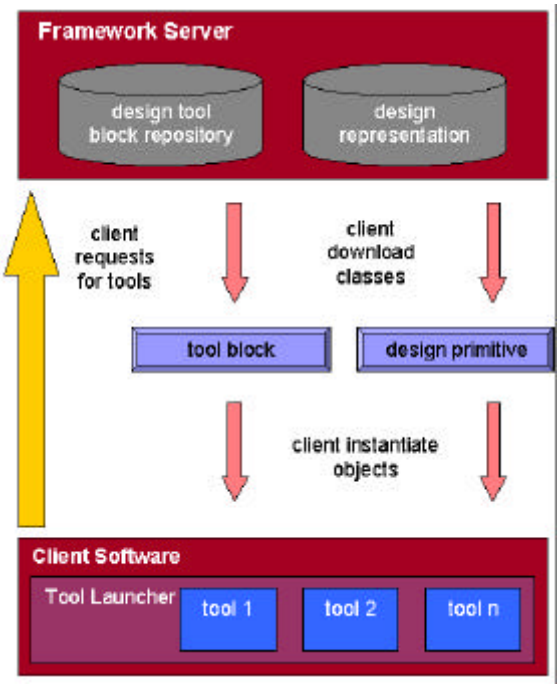


FIGURA 2. 6 - Interação de usuários com o *Framework Server* [IND 99a]

Durante a interação do usuário com as ferramentas, os objetos de projeto são instanciados pelo usuário. Ao término de uma sessão, estes objetos são armazenados em um servidor de persistência, chamado de *Cave Service Space*.

Estudos relacionaram três possibilidades para o armazenamento dos projetos: Banco de Dados Relacional [COD 70], Banco de Dados Orientado a Objetos [MUE 2000] ou um meio de armazenamento compartilhado baseado em *Tupla-Spaces* [GEL 85, MOR 2000]. Essas alternativas de implementação estão melhor detalhadas na seção 2.4.

Atualmente, o armazenamento de objetos do ambiente Cave baseia-se nas tecnologias *Jini/Javaspaces* - modelo *Tupla-Spaces* proposto inicialmente por Gelernter [GEL 85]. Essa tecnologia serviu de base para a construção de um módulo cooperativo *(Collaborative Service)*. Esse módulo já está incorporado ao Cave2 e aparece descrito com maiores detalhes no capítulo 5.

2.3.1 Acesso às ferramentas

O processo de criação de um projeto pelo usuário e o compartilhamento dos seus dados com os demais projetistas envolvidos no projeto é ilustrado na figura 2.7 a seguir. Como visto anteriormente, o usuário cria um projeto instanciando primitivas de projeto representadas por classes no *Cave Framework Server*. Após a criação do projeto, é feito o *upload* dos dados para o *Cave Service Space* (serviço de persistência), onde são armazenados com a possibilidade de serem acessados por outros projetistas.

Além do compartilhamento dos dados, um canal de comunicação é necessário para permitir a cooperação entre projetistas. Atualmente, o Cave utiliza o Cadena [IND 98c], uma ferramenta de *chat* via texto. Pesquisas estão sendo realizadas pelo GME, com o intuito de criar um serviço de *chat* de voz assíncrono. Tal serviço irá permitir que projetistas troquem mensagens de voz de forma assíncrona. Um *chat* de texto implementado com a tecnologia *Jini/Javaspaces* também está sendo testado na tentativa de armazenar voz.

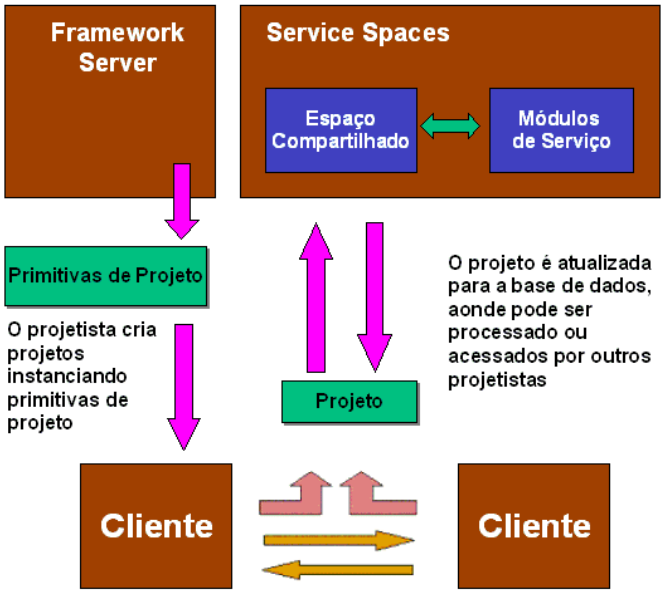


FIGURA 2. 7 - Cooperação no Ambiente Cave2

É importante ressaltar que, durante a implementação do Cave2, houve uma preocupação constante em tornar o sistema facilmente extensível para que haja uma grande reutilização de código, aproveitando-se as vantagens do uso da modelagem

orientada a objetos. Quando analisamos a questão da cooperação, temos como um dos principais objetivos o compartilhamento de objetos de projeto entre projetistas. Da forma como a cooperação foi implementada, os objetos podem ser de qualquer tipo. Não se limita o domínio de uso dessa implementação e permite-se que outras ferramentas que manipulam objetos diferentes possam ser facilmente incorporadas ao *framework*. A cooperação no projeto Cave está especificada com mais detalhes no capítulo 5.

2.4 Alternativas para implementação do repositório de dados

As estratégias de representação dos dados e a arquitetura do repositório de dados são um dos pontos chaves para o desenvolvimento de um ambiente de projeto. No contexto do suporte, na cooperação, ou na interação entre os projetistas, o problema não é diferente. O repositório de dados deve dar suporte ao seu compartilhamento, além de permitir acesso a dados concorrentes, suporte para múltiplas visões, entre outras coisas.

Nesta seção são analisadas diferentes tecnologias para a implementação do repositório de dados: banco de dados relacionais, chamados RDBMS (*Relational Database Management System)*; banco de dados orientado a objetos, conhecidos OODBMS (*Object-oriented Database Management System)*; e espaço de compartilhamento de objetos - que é o foco desse trabalho.

2.4.1 RDBMS (Relational Database Management System)

RDBMS atualmente são modelos muito utilizados em aplicações comerciais e podem também ser apropriados para inúmeras outras aplicações. Entretanto, não são considerados muito adequados para a modelagem de dados manipulados por ferramentas de CAD. Podem ser encontrados também alguns trabalhos que consideram esse modelo não apropriado para suportar colaboração, principalmente devido à sua modelagem e estratégias de acesso aos dados, principalmente seu armazenamento [IND 2002].

2.4.2 OODBMS (Object-oriented Database Management System)

Analisando-as quanto à modelagem dos dados, ambas estratégias, OODBMS e RDBMS, seguem os conceitos de orientação a objetos. Esse paradigma oferece uma semântica rica para modelagem, como por exemplo tipos de dados complexos comuns tanto em ferramentas de síntese como em ferramentas de edição de esquemáticos. Alguns OODBMS, porém, requerem especiais características para os objetos que são armazenados no repositório, tais como uso específico de superclasses e declaração explícita de métodos que alteram o estado dos objetos [MUE 2000]. Em muitos desses casos, a implementação dessas características é simples, mas pode haver alguma restrição na modelagem, principalmente quando se utilizam linguagens que não suportam herança múltipla.

2.4.3 Modelo baseado em *Tupla-Space*

Esse modelo de persistência utiliza conceitos de espaço compartilhado de objetos. Esse meio de persistência distribuída foi introduzido inicialmente por *Gelernter* [GEL 85] em 1980 e, recentemente, estendido pelo grupo *Jini* da *Sun Microsystems* [EDW 99, FRE 99 e MOR 2000]. Essa tecnologia provê um serviço de persistência que trabalha sem a complexidade de RDBMS ou OODBMS, utilizando vários protocolos e serviços que viabilizam a construção de uma arquitetura cooperativa. A arquitetura cooperativa proposta neste trabalho utiliza como base a tecnologia *Jini/Javaspaces*. Seus detalhes estão descritos no capítulo 5.

2.5 Protótipos de ferramentas CAD baseadas na arquitetura Cave2

2.5.1 CIF2VRML

A primeira ferramenta implementada – chamada CIF2VRML, descrita em detalhes em [IND99] – é uma ferramenta de conversão de descrições de *layout* regular para descrições 3D usando VRML. A figura a seguir mostra a interface gráfica dessa ferramenta.

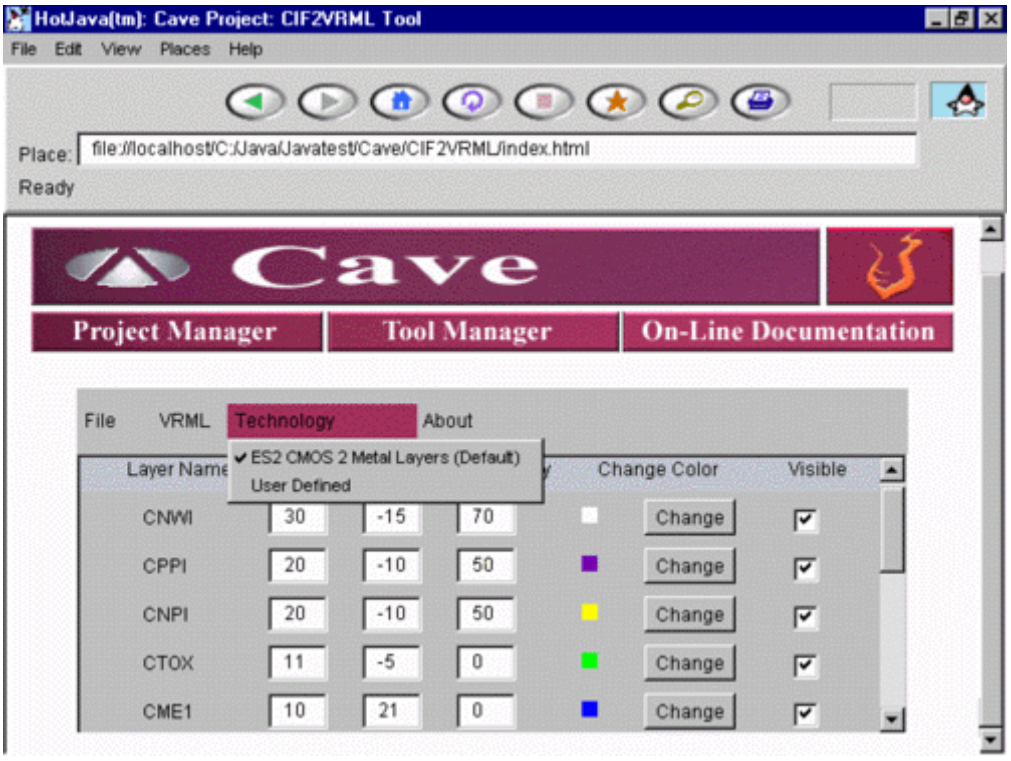


FIGURA 2. 8 - CIF2VRML interface gráfica com o usuário

Como se percebe, o usuário pode carregar o arquivo CIF do disco local ou internet através de uma URL. A ferramenta tem um arquivo de configuração para identificar os nomes do *layout*, usualmente encontrados em arquivos CIF, para seus

parâmetros de conversão necessários, tais como cor e profundidade. A ferramenta carrega um arquivo de configuração padrão, mas o usuário pode utilizar seu próprio arquivo. A conversão é feita de acordo com essas configurações pré-definidas. Depois da conversão, o arquivo VRML pode ser armazenado no disco cliente ou no servidor do *framework* se o usuário estiver autorizado.

Esta ferramenta é mais educacional do que uma ferramenta de automação de projetos. Sua principal característica é permitir uma melhor visualização de *layout* de circuitos através de ilustrações do modelo em 3 dimensões. A figura 2.9 mostra a visualização de um *layout* através da ferramenta CIF2VRML.

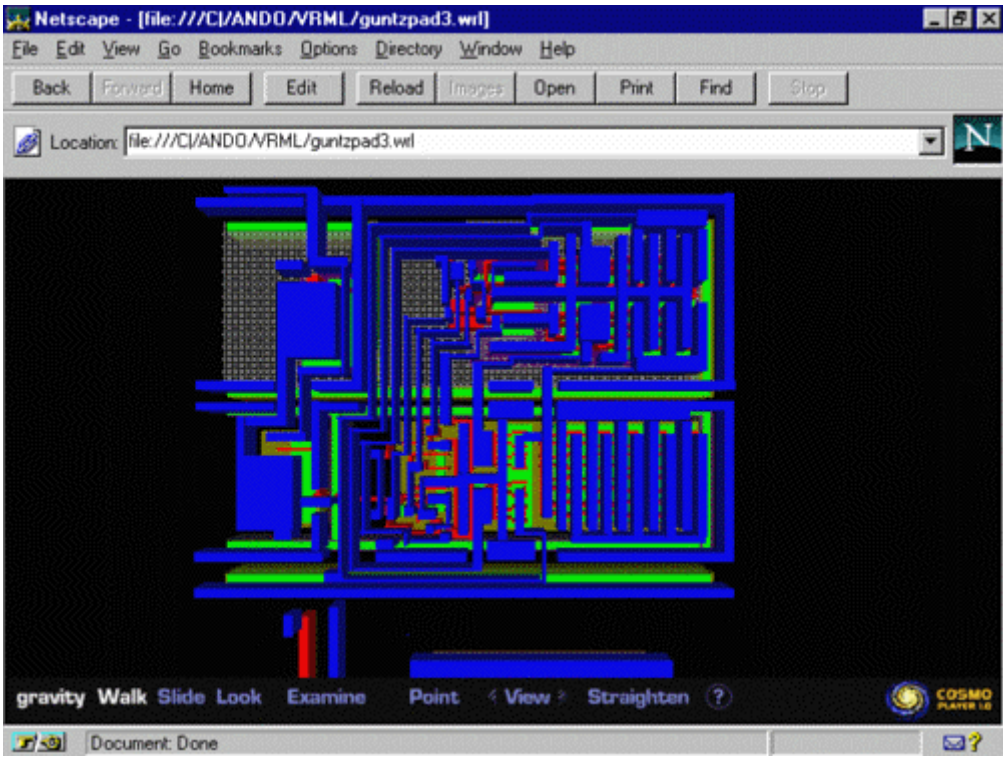


FIGURA 2. 9 - Célula de um circuito integrado gerado pelo CIF2VRML

2.5.2 JALE *(Java Layout Editor)*

O *Jale* é uma ferramenta de edição de *layout*, desenvolvida com o uso da linguagem de programação Java. O desenvolvimento dessa ferramenta foi planejado para ser incremental, baseada em um conjunto de classes Java e de relacionamentos entre elas. Cada uma dessas classes poderia ser reutilizada por outras ferramentas do *framework*, utilizando-se a fundo os conceitos de orientação a objetos.

O protótipo tem funcionalidades básicas de um editor de *layout*: é capaz de ler arquivos CIF, mostrar seu conteúdo graficamente e permitir que o usuário faça algumas edições, além de salvar os dados em formato CIF. Esse protótipo reutiliza alguns

módulos desenvolvidos na ferramenta CIF2VRML tais como: criação, edição e visualização do modelo.

As atualizações na ferramenta *Jale* continuam, assim como o ambiente Cave. Essa ferramenta está incluída na representação de modelos orientados a objeto, e podendo permitir o acesso multi-usuário.

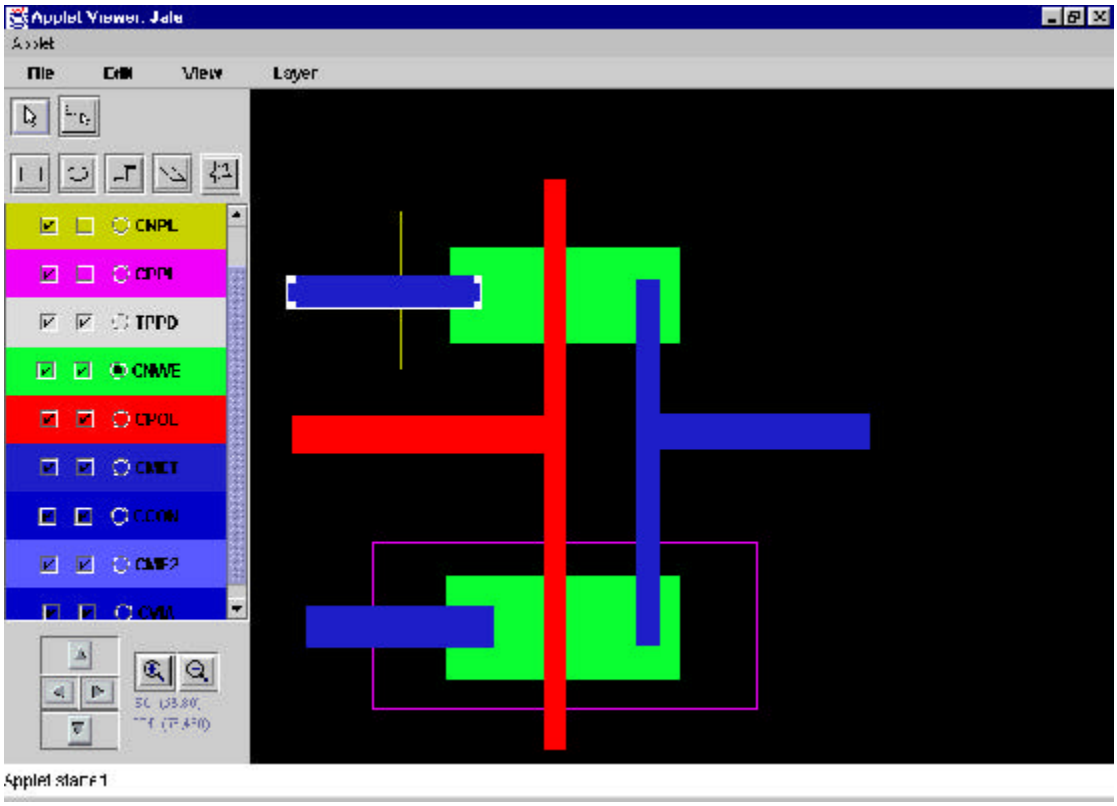


FIGURA 2. 10 - *Jale*: Interface com o Usuário

2.5.3 BLADE *(Block and Diagram Editor)*

Blade é uma ferramenta de edição de esquemáticos hierárquica. E está sendo desenvolvida utilizando modelagem orientada a objetos como abordagem para trabalho cooperativo.

Um editor de esquemáticos geralmente permite a edição de circuitos em nível lógico, usando primitivas de projeto tais como: portas lógicas (AND, OR, NOT,...), blocos funcionais, conexões, portas, etc. Durante a edição do circuito, devem ser permitidas a inserção, remoção e movimentação das primitivas de projeto na área de trabalho do editor. Para tanto, as conexões devem estar fortemente ligadas aos componentes de forma que, quando um componente for movimentado, a conexão a que este estiver ligado também seja alterada. Essa característica é necessária para manter a consistência do circuito.

Uma outra característica importante é a hierarquia, que permite a especificação de um projeto complexo usando-se um editor de esquemáticos. Nesta abordagem, proposta por Brisolara [BRI 2001], o sistema é primeiramente especificado com o uso de blocos funcionais interconectados, os quais na etapa seguinte, são descritos em níveis mais baixos de abstração, como lógicos ou RTL. Tem-se, assim, uma visão gráfica hierárquica do circuito, o que permite a visualização dos sistemas complexos em diferentes níveis de abstração por meio de diferentes representações gráficas / textuais.

Após o término da especificação gráfica do projeto, uma descrição *netlist* é gerada num formato padrão de forma que possa ser utilizada por outras ferramentas que são utilizadas no fluxo de projeto do sistema, tais como: ferramentas de síntese, simulação, etc. Como o editor de diagramas está sendo desenvolvido em Java, uma de suas vantagens é a independência de plataforma.

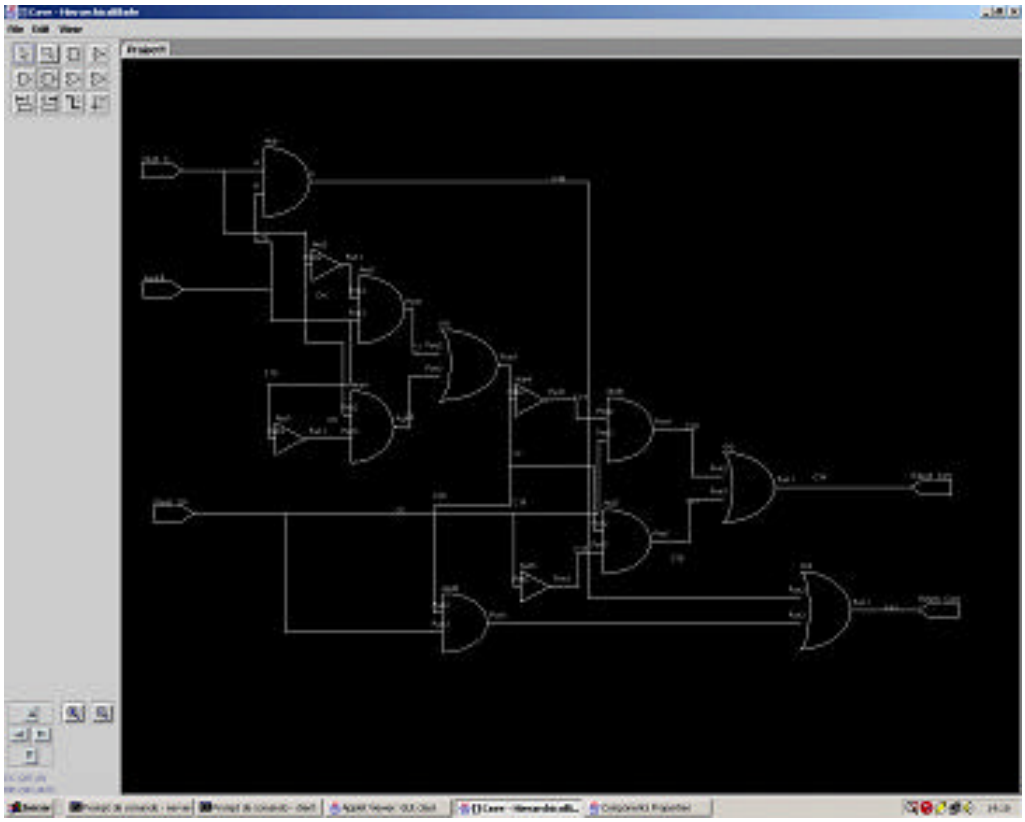


FIGURA 2. 11 - *Blade - Block and Diagram Editor*

2.5.4 Homero *(VHDL Editor)*

A ferramenta Homero consiste em um editor de texto implementado em Java, ao qual se incorpora um analisador sintático da linguagem VHDL. Permite que o projetista escreva seu código VHDL e localize os erros antes de repassar o código para outras ferramentas. Também permite a visualização de cores diferentes para as palavras

reservadas e estruturas sintáticas. A inserção de trabalho cooperativo no Ambiente Cave utiliza essa ferramenta como estudo de caso para validação. Esta pesquisa realiza implementações utilizando as tecnologias *Jini/Javaspaces* como mecanismo de comunicação e armazenamento de projetos. Uma das metodologias empregadas para a cooperação é o *Pair-Programming* (PP) descrita com mais detalhes em [WIL 2000, WIL 2000a, WIL 2000b]. Esse editor implementa os conceitos de *Pair-Programming* em máquinas remotas: dois ou mais projetistas são capazes de compartilhar e editar simultaneamente o mesmo arquivo VHDL. A figura a seguir ilustra a interface com o usuário da ferramenta Homero.

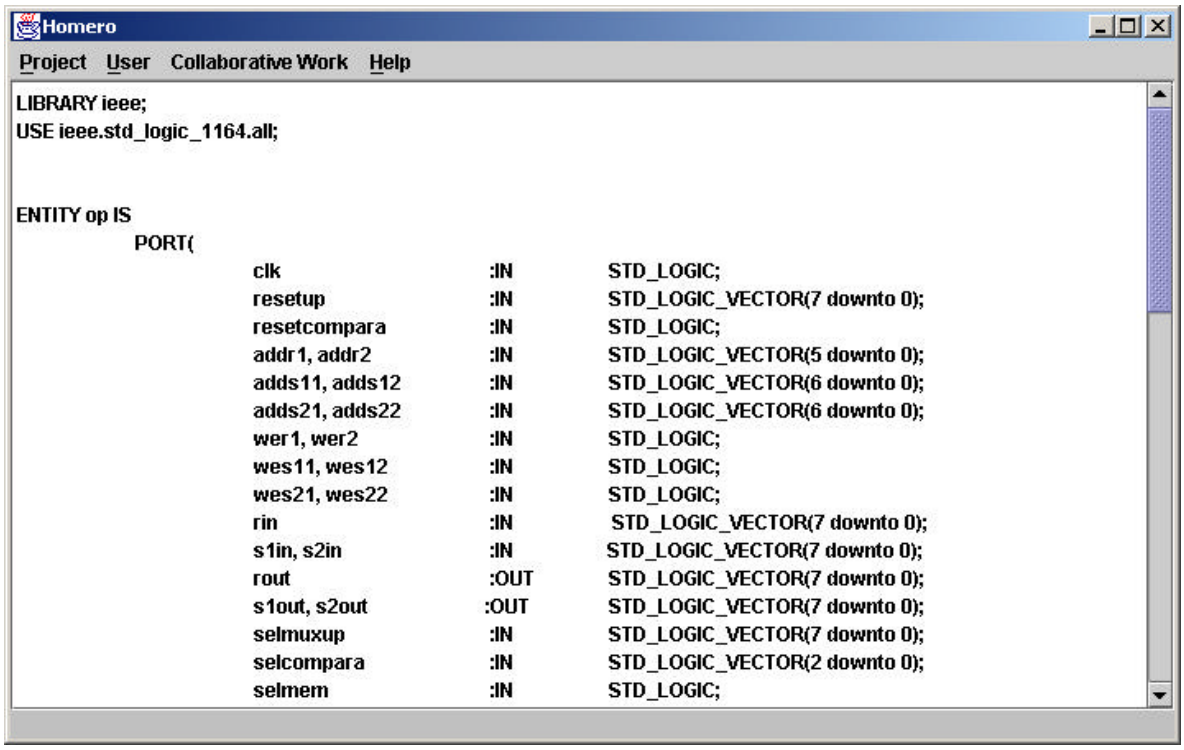


FIGURA 2. 12 - Homero - Editor VHDL

2.6 Cooperação entre as ferramentas de CAD do Ambiente Cave2

Como visto anteriormente, o *framework* Cave incorpora várias ferramentas de CAD em um único ambiente. Com isso, o projetista não necessita mais alternar sua base de trabalho entre plataformas de *hardware/software*. Muito trabalho é poupado, uma vez que grande parte da interface gráfica e do controle de rede do *framework* já está implementada. Além disso, diminui-se o tempo perdido pelo projetista em função das migrações entre diferentes plataformas.

Com a evolução do Ambiente Cave para Cave2, um novo foco de estudo foi introduzido, qual seja, o trabalho cooperativo no projeto de circuitos integrado. Este trabalho descreve e implementa um modelo baseado em espaço compartilhado de objetos

que possibilita a interação entre vários projetistas, mesmo separados geograficamente. Com isso, qualquer ferramenta do *framework* pode requisitar esse recurso. O capítulo 3, como já mencionado anteriormente, descreve o estado da arte das ferramentas que utilizam conceitos de trabalho cooperativo em nível geral e na área de EDA.

Os detalhes sobre a integração desse modelo com o ambiente Cave2, bem como sua especificação, estão descritas nos capítulos 5 e 6. Neste capítulo foram abordados alguns aspectos básicos para o entendimento da aplicabilidade do serviço de cooperação *(Collaborative Service)*.

2.7 Resumo do Capítulo

Este capítulo mostrou a evolução do ambiente Cave para Cave2, destacando alguns pontos importantes dessas arquiteturas. A proposta original do Ambiente Cave foi um modelo de transmissão baseado em hiperdocumentos e com o armazenamento de dados voltado a sistemas de arquivos. A proposta sofreu mudanças a fim de proporcionar uma base mais sólida para aplicações envolvendo projetos cooperativos, o que fez com que o ambiente Cave2 tivesse seu foco em modelos baseados em objetos ao invés de documentos.

Nesse sentido, algumas propostas para infra-estrutura de objetos foram pesquisadas, dentre essas destacam-se: RDBMS (*Relational Database Management System),* OODBMS (*Object-oriented Database Management System)* e espaço de compartilhamento de objetos *(Tupla-Spaces)*. A decisão, por se utilizar um modelo baseado em espaço compartilhado de objetos, deveu-se ao fato de apresentar um modelo de persistência de objetos simples, mecanismos de comunicação compatíveis com a arquitetura Cave2, serviço de transações que utiliza as quatro propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), além de utilizar a linguagem Java como base para as implementações. O capítulo 4 descreve com mais detalhes essa arquitetura, que foi introduzida recentemente pelo grupo da *Sun Microsystems*, chamada *Jini/Javaspaces*.

Este capítulo, assim, teve o intuito de mostrar onde o *Collaborative Service* está inserido. Neste caso, o Ambiente Cave2, com a inserção desse módulo de cooperação, incorpora mais um recurso, aumentando as opções de trabalho no projeto de circuitos integrados.

3 Trabalho Cooperativo

3.1 Introdução

Este capítulo tem como objetivo discutir alguns princípios do trabalho cooperativo, enfocando o estado da arte de ferramentas cooperativas em diversas áreas. Visa também expor alguns modelos de cooperação descritos na literatura, além de verificar quem está trabalhando atualmente com aplicações cooperativas na área da EDA *(Electronic Design Automation)* e descrever a arquitetura de alguns desses modelos.

O termo "trabalho cooperativo", segundo Souza [SOU 96], teve sua origem no século XIX, quando utilizado por economistas para designar o trabalho envolvendo vários autores. De acordo com Borges [BOR 95], o termo CSCW *(Computer Supported Cooperative Work)* surgiu na década de 80, quando foi título de uma conferência da ACM *(Association for Computing Machinery)*.

Segundo [FAR 98], um sistema cooperativo é formado por vários usuários que estão envolvidos em uma atividade compartilhada, normalmente em lugares remotos. Os sistemas cooperativos são diferenciados das aplicações distribuídas pelo fato de que os usuários do sistema estão sempre trabalhando para um objetivo em comum, com a necessidade de interagirem: trocam solicitações, dividem informações e avaliam sua situação junto aos outros usuários. Em um trabalho em grupo interativo, membros cooperam entre si para chegar em um objetivo comum, cada um contribuindo com sua parte para o produto final.

Um sistema que possui um certo grau de concorrência entre usuários que buscam a interação também pode ser considerado um sistema cooperativo. Por exemplo, uma sessão de *chat* é um processo de cooperação, pois todos os usuários envolvidos precisam coordenar suas tarefas para garantir que os participantes da mesma sessão não percam o comentário de ninguém. Um outro exemplo pode envolver um simples cliente de *e-mail* que não tem cuidado sobre o estado de nenhum outro cliente, pois não precisa se coordenar com ninguém para alcançar seu objetivo. Em [CAY 2000] foram ressaltados alguns dos elementos principais de um sistema cooperativo, tais como: interfaces com usuários, servidores, repositório de dados e transações entre usuários, servidores e repositório de dados.

Com essas condições, o processo de desenvolvimento de circuitos integrados, assim como o processo de desenvolvimento de *software*, apresenta muitas atividades inerentemente cooperativas. Quer dizer, possuem inúmeras atividades que requerem o trabalho em grupo e conseqüentemente de ferramentas de apoio [BOR 95, SOU 98, PIN 99, PIN 99a]. Como relata [WIL 94]: "a distribuição das organizações tem obrigado seus profissionais a trabalharem com colegas distantes". Essa é uma motivação para que pesquisas sobre trabalho cooperativo tenham aplicabilidade também em ambientes de apoio ao projeto de circuitos integrados.

3.2 CSCW (Computer Supported Cooperative Work)

Segundo Souza [SOU 96], CSCW é uma área de pesquisa que auxilia os projetos de sistemas de aplicação. O CSCW deve ser entendido como um esforço para o entendimento da natureza e característica do trabalho cooperativo cujo objetivo é melhor adequar os projetos a tecnologias baseadas em computador. Então, esta área aborda a criação e desenvolvimento de um suporte informatizado ao trabalho em grupo [GRU 94].

De acordo com Dietrich [DIE 96], alguns autores diferenciam CSCW de *groupware*. Segundo eles, CSCW indica a pesquisa na área de trabalho cooperativo suportado por computador, enquanto *groupware* referencia os sistemas que são frutos dessas pesquisas. A citação de Souza [SOU 96] resume a idéia por traz dos *groupwares:* "As soluções baseadas em *groupwares* estão emergindo para fornecer maior competitividades às organizações". As empresas (ou grupos) mais produtivas mantêm-se competitivas por mais tempo no mercado, tendo mais chance de sobrevivência e de alcançar uma posição de destaque.

Vários fatores têm sido determinantes para o sucesso da área de CSCW. A necessidade de resultados rápidos tem sido a maior incentivadora do processo de formação de grupos. Outros fatores que impulsionaram a pesquisa em CSCW, destacados em [FER 98, IND 97 e SOU 96], são a disseminação de redes de computadores nos mais variados ambientes de trabalho, a adoção de sistemas distribuídos e a necessidade de compartilhamento de recursos.

3.3 Tecnologias de Suporte

O reconhecimento e a aplicação de conceitos de áreas das ciências do comportamento (Psicologia, Sociologia, dentre outras) não são suficientes para garantir com que o desenvolvimento de ferramentas de suporte ao trabalho cooperativo sejam eficazes. A complexidade dessas ferramentas envolve conceitos e tecnologias de diversas áreas da Ciência da Computação. Dentre elas: Redes de Computadores, Banco de Dados, Sistemas Operacionais, Interação Homem-Máquina, Multimídia e Hipermídia e Agentes Inteligentes [SOU 96, DIE 96]. A seguir, é discutida a contribuição de cada uma dessas tecnologias para o suporte ao trabalho cooperativo.

3.3.1 Redes de Computadores

Da mesma forma que a diversidade e a sofisticação dos computadores sofreram uma considerável expansão em um período bastante curto, as tecnologias de redes de computadores e comunicação também propiciaram um espantoso aumento no volume e tipo de interações entre computadores. Segundo [DIE 96], as redes de computadores são o meio com o qual são implantados os mecanismos de comunicação, compartilhamento e troca de informações. O próprio modelo cliente/servidor ganhou funcionalidade e crescimento com a popularidade computadores pessoais e redes locais

(LANs – *Local Area Network*) como solução para organizar grupos de trabalho locais – *Workgroups* [TAN 96]. As aplicações baseadas na premissa de altas velocidades e redes de comunicação públicas de baixo custo devem facilitar ainda mais a expansão do trabalho cooperativo.

É bastante comum em ferramentas cooperativas a arquitetura baseada no modelo cliente/servidor. O que, segundo [DIE 96], é uma forma eficiente e simples de estruturar um sistema como um grupo de processos cooperativos (servidores), os quais oferecem serviços para os usuários (clientes).

3.3.2 Banco de Dados

O suporte para um espaço de informações compartilhadas é um dos grandes problemas de um projeto cooperativo. Na verdade, esse elemento é fundamental. O uso de um meio de armazenamento persistente fornece suporte ao trabalho cooperativo pelo fato de servir como repositório de informações e permitir o acesso com facilidade a estas informações.

Uma estrutura de armazenamento de dados, em essência, deve prover serviços que garantam o acesso eficiente e concorrente a informações, permitindo notificações e conhecimento das atividades de outros membros do grupo, acompanhamento da evolução das atividades, informações do grupo, etc.

O conjunto básico de características para o suporte de armazenamento de dados compreende: controle de concorrência aos objetos; controle de versões dos diversos objetos compartilhados; manutenção da história de uso dos diversos objetos do sistema; acesso estruturado e não-estruturado aos objetos armazenados; gerenciamento de informações sobre o grupo e seus membros, incluindo seus privilégios e papéis; gerenciamento das interações entre os membros.

Os objetos compartilhados pelo grupo são dos mais variados tipos: som, texto, imagens, vídeos, programas e dados estruturados dos mais diversos formatos. Além disso, um ambiente de cooperação impõe novos requisitos para o tratamento e armazenamento de objetos, como, por exemplo, o aspecto temporal, visto que as ações se dão síncrona ou assíncronamente [SOU 97].

Segundo [DIE 96], outro mecanismo passível de utilização é o mecanismo de notificação. Pode ser aplicado como parte do controle de concorrência da seguinte forma: sempre que um participante do grupo tentar acessar um objeto que está sendo utilizado por outro participante, ambos são notificados. Há, assim, uma interação para resolver o confronto de uso simultâneo.

3.3.3 Sistemas Operacionais

São inúmeras as funcionalidades básicas oferecidas pelos Sistemas Operacionais: controle a recursos compartilhados entre os participantes locais e remotos,

mecanismos de comunicação, dentre outros [TAN 92]. Podem-se aproveitar algumas funções já existentes relacionadas, por exemplo, o controle de concorrência, paralelismo e gerência de recursos distribuídos. Segundo [DIE 96], essas funções vão determinar o acesso consistente e eficaz aos recursos, evitando-se conflitos maiores entre os membros do grupo, levando em conta fatores como segurança, transparência de localização e tolerância a falhas.

3.3.4 Aspectos de Interface Homem-Máquina

As diretrizes e práticas associadas ao projeto de boas interfaces foram em geral formuladas considerando sistemas com um único usuário, nos quais o indivíduo interage com o computador. No caso de ferramentas cooperativas, é necessário que se leve em conta características da comunicação mediada por computador, em que indivíduos interagem com outros indivíduos através do computador. A transição de sistemas de um único usuário para sistemas voltados para o trabalho em grupos requer, portanto, uma mudança de perspectiva. No primeiro caso, o indivíduo está interagindo com a máquina, podendo estar no controle das ações. No entanto, em um sistema cooperativo, existem outros elementos inteligentes (humanos, ou gerados pela máquina), e isso traz conseqüências para a interface, pois o usuário não está mais totalmente no controle da situação. Do ponto de vista do usuário, a interface não é somente seu caminho para o sistema, mas também o caminho para os outros membros do grupo. Para [DIE 96], a utilização de tecnologias de Interface Homem-Máquina se dá basicamente através da interface do sistema. Esta deve perfeitamente encorajar a cooperação.

Abaixo são mostrados alguns requisitos levantados para o desenvolvimento de interfaces cooperativas:

- *Aplicações efetivas devem ser consistentes internamente entre si*: é comum o usuário ter acesso a aplicações compartilhadas pelo grupo juntamente com aplicações de uso individual. Neste caso, é importante manter a consistência entre diferentes tipos de aplicações.

- *Fornecer um "feedback" imediato:* significa mandar uma notificação ao usuário sobre uma ação que foi executada e sobre qual resultado foi atingido.

- *Concorrência:* é necessário um suporte a múltiplas entradas simultâneas em uma aplicação compartilhada. Sem este suporte, o período de tempo que cada indivíduo terá para acesso ao recurso compartilhado será dependente do número de membros do grupo.

- *Comunicação entre os membros do grupo:* esta função deve ser provida em paralelo às demais funções do sistema para permitir a interação entre membros durante a execução de suas atividades.

3.3.5 Agentes Inteligentes

O uso de Agentes Inteligentes concentra-se principalmente realização de tarefas que seriam tediosas para a realização humana. Todavia, são tarefas que, no contexto cooperativo, são muito importantes, como a criação de boletins informativos e a notificação da presença de participantes [DIE 96].

3.3.6 Hipertextos e Multimídia

O enfoque de hipertextos permite a criação e a interligação de fragmentos de informação e combina bem com as necessidades do suporte ao trabalho cooperativo. Por outro lado, os problemas de interação existentes no trabalho em grupo representam uma importante vertente da utilização e adaptação dos conceitos de hipertexto a ferramentas cooperativas – *groupware.*

Nesse sentido, o conceito de multimídia torna-se essencial quando se pensa em usar hipertextos como forma de comunicação entre membros de um grupo. Afinal, diferentes formas de mídia aumentam as possibilidades de expressão nas diversas atividades humanas. Em geral, para atender às necessidades do trabalho cooperativo, a ferramenta de hipertexto deve possuir recursos adicionais tais como: compartilhamento e proteção (controle de acesso às informações), acesso concorrente, distinção entre dados privados e públicos, suporte a aplicações específicas por cada usuário (editores, planilhas, etc.), mecanismos de coordenação das atividades do grupo [DIL 94].

3.4 Ambientes colaborativos

De acordo com Otsuka [OTS 97] um ambiente para ser colaborativo deve reunir algumas funcionalidades que servem de apoio para as principais atividades: comunicação, negociação, percepção, coordenação, compartilhamento, construção colaborativa de conhecimentos, representação de conhecimentos e avaliação colaborativa. A seguir, são apresentadas algumas considerações sobre cada uma destas atividades.

3.4.1 Comunicação

Sem dúvida, a comunicação é a mais importante característica das atividades em grupo; sem ela não existe colaboração. Segundo Cayres [CAY 2001], é durante a comunicação que ocorrem as trocas de idéias, discussões e os conflitos entre os pares. É a base das interações sociais, e a responsável, segundo os sócio-construtivistas, pela catalisação do processo de desenvolvimento cognitivo individual. Segundo a corrente sócio-cultural, citada em Cayres:

> *É através da comunicação que ocorre o desenvolvimento a nível inter-psicológico, a partir do qual conceitos são internalizados e ativamente transformados pelo indivíduo, através da reflexão, constituindo assim o desenvolvimento a nível intra-psicológico. Já, de acordo com a corrente da cognição compartilhada, a comunicação e todo o contexto social que*

> *envolve os seus participantes, permite a construção e manutenção de conceitos compartilhados, os quais são considerados produtos do grupo todo.*

Através das redes de computadores, a comunicação entre os participantes de um grupo pode ser apoiada por ferramentas de comunicação **síncronas** e **assíncronas**. A comunicação síncrona ocorre com os usuários que estão ativos no sistema num mesmo momento, podendo ser realizada através de trocas de mensagens textuais (como no *talk e chat*) ou através de áudio e vídeo (videoconferências). Na comunicação assíncrona, não se exige esta coordenação temporal, os participantes podem realizar suas contribuições no momento em que julgarem mais apropriado, o que soluciona um dos grandes problemas do trabalho em grupo, que é a incompatibilidade do horário de trabalho entre os participantes de um grupo. A comunicação assíncrona pode ocorrer através de mensagens textuais (como no correio eletrônico, *news e* documentos/hiperdocumentos), ou através de áudio e vídeo gravados [CAY 2001].

Os dois conceitos descritos acima atuam diretamente no modo como se dá uma consciência de grupo. Neste sentido, surge a **colaboração transparente** e a **colaboração consciente**. Com a colaboração transparente, as atualizações realizadas por um determinado usuário só são percebidas pelos demais quando acessadas por alguém. Na colaboração consciente, há um mecanismo para notificar todos os envolvidos no projeto de que alguma alteração ocorreu [CAY 2001]. No *Collaborative Service* aplica-se a colaboração consciente, pois seu modelo implementa mecanismos de notificação entre os usuários das ferramentas.

Outro ponto importante é o local da comunicação. Pode-se ser no **mesmo** ou em **diferentes** lugares. As classificações espaço-temporais são as principais em qualquer análise de atividade colaborativa auxiliada por computador [CAY 2001]. Para Barros [BAR 94], combinando-se as dimensões tempo e espaço, têm-se diferentes modalidades de interação entre os participantes de um grupo, conforme mostra a tabela 1.

TABELA 3. 1 - Taxonomia espaço temporal segundo [UNA 91]

Taxinomia espaço-temporal	**Mesmo tempo**	**Tempo diferente**
Mesmo lugar	Interação face a face	Interação assíncrona
Lugar diferente	Interação distribuída síncrona	Interação distribuída assíncrona

Em termos de modalidade, a comunicação pode ser **comunicação direta** (ou explícita) ou **interação indireta** (ou implícita). Na comunicação direta, a interação acontece através de gestos, áudio e/ou vídeo, transmissão de texto entre o grupo (que é chamado normalmente de comunicação). Na interação indireta, a comunicação acontece através do objeto de trabalho, por exemplo, um texto ou imagem compartilhados [CAY 2001].

Quanto à forma de comunicação, ela pode ser **livre** ou **estruturada.** Na forma livre, a comunicação acontece através de representações simples, tais como frases, textos, desenhos, etc., que são lançadas livremente e sem estruturação. Porém, essa forma de comunicação depende de objetivo e de duração, pois pode gerar uma grande quantidade de informações de forma desorganizada [CAY 2001]. Nesse caso, é preferível dispor de mecanismos que auxiliem a organização da informação [BAR 94].

É importante destacar que a forma de comunicação livre permite a realização de uma **comunicação informal**. Atualmente, esse tipo de comunicação é reconhecida por pesquisadores como sendo de grande importância para um ambiente de trabalho. Essa comunicação, dita como "acidental", pode ter um melhor entendimento através dos exemplos citados em Cayres [CAY 2001]: dois colegas se encontram num corredor e discutem um tópico de interesse comum, ou um grupo se reúne informalmente para o cafezinho e conversam sobre algum assunto relacionado ao trabalho. A **comunicação formal,** ao contrário da comunicação informal, segue um modelo de conversação pré-estabelecido, que pode ser representado por alguma ferramenta formal. Porém, nem os padrões de comunicação, nem a estrutura dos grupos conseguem manter a estabilidade no andamento de um trabalho colaborativo, o que torna estes modelos de conversação bastante limitados. Por esse motivo, é cada vez mais comum a construção de sistemas que oferecem recursos para a realização de encontros informais à distância tanto para a comunicação entre indivíduos, quanto para a comunicação entre grupos [CAY 2001].

3.4.2 Negociação

A negociação deve ser apoiada efetivamente por um ambiente de apoio às atividades colaborativas. As ferramentas de comunicação não são suficientes quando existe a necessidade do grupo tomar alguma decisão em conjunto. Portanto, para auxiliar o grupo na tomada de decisões, para que se satisfaça a maioria, os sistemas de CSCW devem prover ferramentas que auxiliem a solução de conflitos [BAR 94, CAY 2001 e DIE 96]. Com tais ferramentas, cada integrante do grupo pode defender a sua opinião e expor os seus argumentos. Essas opiniões são lidas pelos colegas e contra-argumentadas até que se chegue em um consenso entre todos [CAY 2001].

Segundo Cayres [CAY 2001], em um processo de negociação, estão envolvidos vários mecanismos cognitivos e afetivos - lógica, inferência, dedução, crença, dúvida, sutileza e envolvimento emocional com o assunto e também com os participantes. Vários fatores devem ser cuidadosamente observados para se fazer uma opção tecnológica de instrumento de suporte ao processo. Dentre eles, o tema, o objetivo, as questões, o “custo” das decisões, o tempo disponível, as pessoas envolvidas, o conhecimento comum, as hierarquias, os locais de reunião, as características culturais, entre outros.

Classificando-se a negociação quanto à forma, ela pode ser **livre** ou **estruturada.** Na negociação livre, o computador é utilizado como um mecanismo de registro e distribuição dos argumentos usados durante a negociação e são expostos de

forma livre e não formatada. Por outro lado, em negociações ditas complexas, este modelo torna difícil a percepção dos relacionamentos entre os argumentos e posicionamentos. Quando ocorrem esses casos, a negociação estruturada é mais indicada, pois permite mais clareza entre as falas da negociação [CAY 2001].

Quanto à coordenação, uma negociação pode ser **livre** ou **orientada.** Na forma livre todos têm os mesmos direitos, prioridades e responsabilidades. Diferente de uma coordenação orientada, na qual são definidos "papéis" com direitos e prioridades diferentes para cada integrante da negociação. Esse tipo de negociação se faz necessário em casos mais complexos, em que haja um grande número de participantes com um objetivo de conduzir melhor a negociação.

3.4.3 Coordenação

A coordenação das atividades em grupo é de extrema importância para que os objetivos sejam alcançados de forma organizada, produtiva e harmoniosa [CAY 2001]. Segundo Dietrich [DIE 96], a necessidade de coordenação existe quando a tarefa de um participante depende do que o outro participante está fazendo ou vai fazer. Estes precisam coordenar e sincronizar suas atividades.

Um grupo necessita de coordenação. Essa atividade envolve planejamento, distribuição de tarefas, bem como acompanhamento. Na fase de planejamento são definidas metas e prazos a serem cumpridos, também é realizada a delegação das tarefas que necessitam ser realizadas para que o grupo alcance o seu objetivo comum. O sistema pode realizar muitas das atividades de coordenação, por exemplo, o registro de tarefas concluídas, envio de avisos sobre atividades atrasadas, dentre outros [CAY 2001].

3.4.4 Percepção

É de fundamental importância para as atividades em grupo, que cada participante tenha a percepção das ações dos demais participantes. Segundo Dietrich [DIE 96], existem duas formas de prover a percepção em *groupwares*: a explicitamente gerada e a passivamente colecionada e distribuída. Em uma forma explicitamente gerada, as ações são armazenadas pelo sistema e freqüentemente precisam ser informadas pelos participantes. Com isso, as ações dos participantes são distribuídas a todos em formato de boletins informativos ou relatórios em horários pré-determinados ou quando solicitadas. Já, na forma passivamente colecionada e distribuída, as ações são distribuídas em tempo real, à medida que vão acontecendo [CAY 2001].

3.4.5 Compartilhamento

Para o desenvolvimento de uma colaboração, é necessária a criação de uma "memória organizacional do grupo" [BAR 94] que seja acessível a todos os participantes, pois as atividades de aprendizagem e trabalho colaborativo envolvem o compartilhamento de objetivos, idéias, descobertas, objetos e produtos destas atividades. Portanto, nesta memória devem ser armazenados padrões, orientações, projetos,

descobertas e resultados do trabalho que está sendo desenvolvido pelo grupo. Segundo Cayres [CAY 2001], também devem ser registrados resultados de reuniões, decisões, planos de ações e outras informações que orientem o desenvolvimento do trabalho do grupo, além de auxiliar a aprendizagem colaborativa. Neste caso, o compartilhamento se pode dar ao mesmo tempo ou em tempos diferentes.

3.5 Sistemas colaborativos

De acordo com [FAR 98], um sistema colaborativo é formado por vários usuários envolvidos em uma atividade compartilhada, normalmente em locais distribuídos. É importante destacar que, dentre as aplicações distribuídas, os sistemas colaborativos são diferenciados pelo fato dos seus usuários estarem sempre trabalhando em relação a um objetivo comum e, conseqüentemente, têm a necessidade de interagir. Segundo Cayres [CAY 2001], alguns dos principais elementos de um sistema colaborativo são: 1) interfaces com usuários; 2) servidores; 3) repositório de dados; e 4) transações entre usuários, servidores e repositório de dados. Isso é o que está ilustrado na figura 3.1 a seguir:

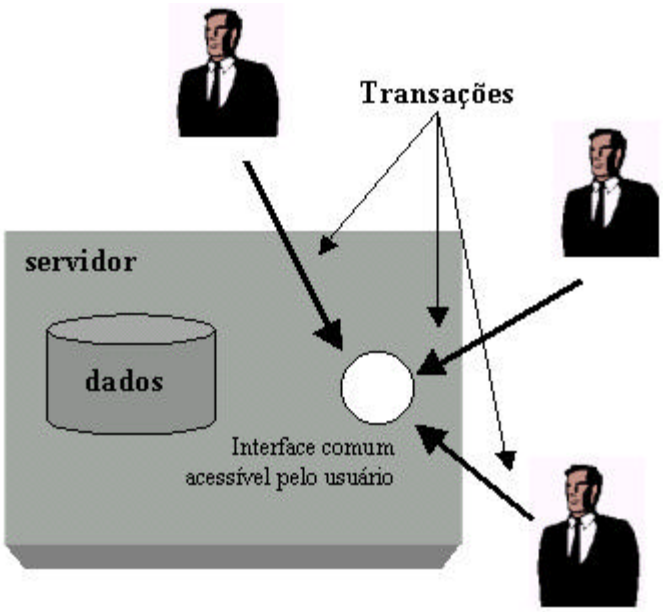


FIGURA 3. 1 - Composição de um sistema colaborativo, segundo [FAR 98]

3.5.1 Comunicação como necessidade

Em um sistema colaborativo, podem existir vários usuários distribuídos cooperando entre si. A comunicação, nesse caso, depende muito da aplicação. Segundo [CAY 2001], uma comunicação precisa suportar: 1) mensagens ponto-a-ponto entre

usuários; 2) envio de mensagens *broadcast* para uma comunidade de usuários; ou 3) envio de mensagens *narrowcast* para usuários participantes de grupos específicos.

3.5.2 Identificadores de usuários

Segundo [CAY 2001], se o acesso ao sistema ou para certos recursos associados com o sistema necessitarem ser restritos, é necessário o uso de uma identificação. Como exemplo, pode-se citar a aplicação de quadro compartilhado *whiteboard* [FAR 98]. Um quadro compartilhado é um espaço virtual de desenho no qual vários usuários distribuídos podem ver e escrever, compartilhando informações, sugestões, etc. Em um exemplo real, um grupo de pessoas pode estar reunido em uma sala de aula, trabalhando ao redor de um quadro negro. Com certeza, para que um indivíduo que esteja usando o quadro saiba quem escreveu e o que escreveu, ele terá que instituir algum tipo de identificador para cada um dos participantes. Com isso, cada contribuição é mostrada com sua respectiva identificação, o que se torna útil para cada indivíduo e possibilita a correção e modificação de suas contribuições [CAY 2001].

3.5.3 Estados Compartilhados

Nos sistemas colaborativos em geral, vários dados e recursos são compartilhados entre muitos participantes. Manter a integridade da informação quando múltiplos usuários estiverem acessando e modificando o estado compartilhado se torna uma questão importante. Afinal, se dois ou mais participantes tentarem alterar o mesmo "trecho" de informação, deve existir uma maneira proteger e informar aos participantes afetados [CAY 2001].

3.5.4 Desempenho

Para muitos sistemas colaborativos, é necessário alterar entre manter a consistência dos estados compartilhados entre todos os usuários e maximizar desempenho [CAY 2001]. Uma forma mais simples de se fazer isso é com o apoio de um mediador central que atua como um distribuidor de eventos [FAR 98]. Segundo [CAY 2001], um problema é que um mediador central pode se tornar lento à medida que o sistema aumenta. Caso existam muitos usuários para serem notificados, o mediador pode ter dificuldades para manter o tráfego. Com isso, os usuários do sistema perderão tempo esperando pelas atualizações.

3.6 Modelos de cooperação

Essa seção apresenta, de forma sucinta, as principais características de três modelos de cooperação propostos na literatura. Além disso, introduz um quarto, criado por Sawicki [SAW 2002] no âmbito do projeto Cave.

3.6.1 Modelo de cooperação segundo Käfer

Em Käfer [KÄF 90], a aplicação do projeto é representada por um grafo. Os nodos do grafo representam as tarefas a serem executadas no contexto do projeto; os arcos especificam as relações de dependência da tarefa/sub-tarefa. A figura 3.2 ilustra o conceito de grafo de tarefas através de um exemplo. Neste caso, o microprocessador é dividido em um conjunto de sub-projetos (memória, barramento, ucp e unidade de ponto flutuante). Essa divisão de projetos em conjuntos de sub-projetos é repetida de forma recursiva até que os sub-projetos alcancem o nível de complexidade desejado [IOC 91].

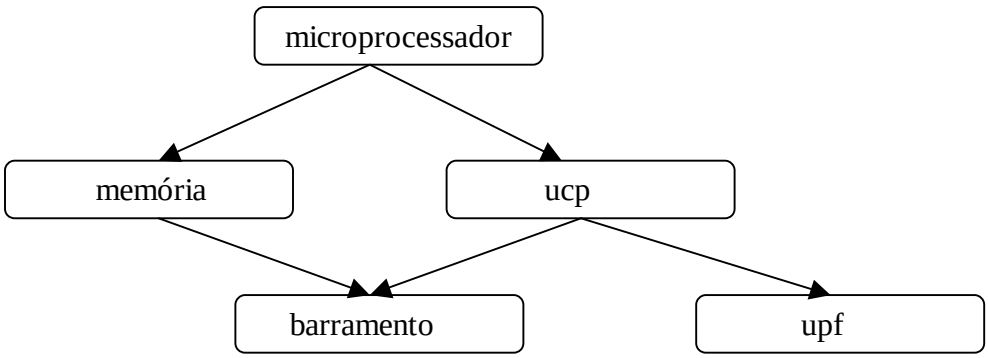


Figura 3. 2 - – Exemplo do grafo de tarefas [KÄF 90]

Como se vê, Käfer modela as características dinâmicas da aplicação de projeto, associando a cada tarefa do grafo um único projetista. Com isso, se um objeto é complexo demais para ser projetado por um projetista, é dividido em sub-objetos, tal que cada um dos últimos possa ser desenvolvido por um único projetista. Conseqüentemente, tarefas relacionadas com o projeto de sub-objetos de um mesmo objeto assumem uma relação íntima de dependência [IOC 91].

3.6.2 Modelo de cooperação segundo Nodine e Skarra

Nesse modelo, Nodine e Skarra [NOD 90a, NOD 90b e SKA 90], mostram a aplicação de projeto como uma hierarquia de projetos (sub-projetos) em forma de árvore. Esses projetos são recursivamente subdivididos até que os objetos a serem desenvolvidos atinjam um nível de complexidade aceitável. Ao contrário do modelo de Käfer, o modelo de cooperação de Nodine e Skarra admite a possibilidade de mais de um projetista estar envolvido no desenvolvimento do mesmo sub-projeto.

O modelo proposto parte da premissa de que, em um projeto muito complexo, os projetistas reúnem-se naturalmente em grupos e que cada grupo é responsável pela execução de uma ou mais tarefas. Acompanhando a hierarquia de tarefas, os grupos de projetistas formam também uma estrutura hierárquica [IOC 91].

O modelo de Käfer, que centra a problemática da cooperação entre projetistas na troca de objetos de projeto, difere do modelo de Nodine e Skarra ao propor o suporte à cooperação através da sincronização de tarefas realizadas pelos grupos de projetistas e

também daquelas realizadas por diferentes projetistas de um mesmo grupo, como ilustra a figura 3.3 para a organização de um projeto de um avião [IOC 91].

Ao contrário de Käfer, Nodine e Skarra permitem que um objeto de projeto seja tanto criado, quanto alterado por mais de um projetista. Todavia, para que isso seja possível é necessário especificar regras de acesso que garantam a integridade dos dados.

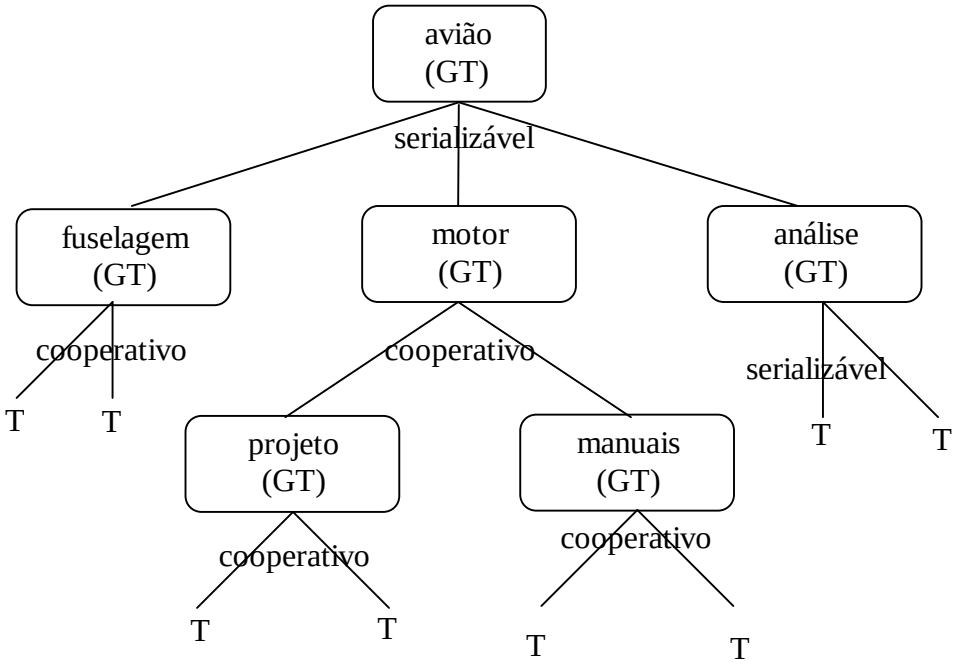


Figura 3. 3 - – Organização do projeto em sub-projetos

3.6.3 Modelo de cooperação segundo Iochpe

O modelo de cooperação proposto citado em [IOC 91] pode ser visto como uma extensão do modelo de Käfer [KÄF 90]. Igual ao modelo de Käfer, o processo evolucionário da construção de objetos é materializado por grafos de versões que podem ser percebidos e utilizados pelos projetistas. De outro lado, esse novo modelo reconhece e tenta contornar algumas deficiências do modelo de cooperação de Käfer. Em Käfer, os projetistas não têm visão global do projeto, nem de sua evolução. Não existe nenhuma estrutura de dados ou transação que represente o projeto, sua divisão em tarefas ou seu estado [IOC 91].

O modelo de Iochpe tenta dar subsídios aos projetistas para que possam pré-definir as condições para a cooperação e qual o momento em que essa vai acontecer. Esse modelo oferece, ainda, elementos que permitem a especificação e o controle de ambientes cooperativos em aplicações que fazem uso de ferramentas automatizadas [IOC 91].

Os sub-projetos são especificados pelo conjunto de características que devem apresentar ao fim do projeto (igual ao modelo de [KÄF 90]). A cada objeto da hierarquia de projeto é associado um conjunto de tarefas. Através da execução destas tarefas, o conjunto de características do objeto vai, passo-a-passo, sendo materializado.

Após o projeto ser dividido de forma recursiva, em um conjunto de sub-projetos, estes últimos são respectivamente associados a grupos de projetistas. Com isso, cada grupo deve, então, detalhar as características previstas, assim como os planos de implementação dos objetos de seu projeto.

Analisando-se o modelo de Iochpe, fica claro que ele está centrado nas ordens parciais de passos de projeto que geram objetos de projeto e nas características destes objetos. Os projetistas dividem o projeto global em sub-projetos e associam, a cada um destes, um grupo de projetistas. Cada grupo fica responsável pela execução de, pelo menos, um sub-projeto.

3.6.4 Modelo de cooperação proposto para o *Collaborative Service*

No modelo proposto para o módulo de cooperação [SAW 2002], o projeto é dividido em blocos de projeto, semelhante aos sub-projetos descritos nos modelos de Käfer, Nodine e Skarra, e Iochpe. Porém, nesse modelo, os blocos de projeto não trabalham com dependências e sua representação é constituída por conjuntos de objetos (blocos) interligados no contexto de um objeto em comum (projeto). Isso faz com que todo o grupo de projetistas interajam entre os todos blocos de projeto.

Diferente do modelo de Käfer, o modelo de Sawicki possibilita, além da interação de mais de um projetista por cada bloco de projeto, sua participação em mais de um bloco ao mesmo tempo. Porém, para cada bloco de projeto, apenas um projetista tem a permissão de alteração.

Os blocos de projeto são atualizados separadamente dos demais e executados dentro de pequenas transações. Cada bloco de projeto constitui uma sessão cooperativa e, conseqüentemente, vários blocos de um projeto constituem-se em várias sessões cooperativas. Nesse sentido, além de possibilitar-se a interação entre inúmeros projetistas, o modelo trabalha com um sistema de percepção relacionado com cada bloco de projeto. Com isso, no momento que um projetista entrar ou sair de um bloco de projeto, os demais participantes receberão uma notificação.

Esse modelo não trabalha com banco de dados, mas com conceitos de espaço compartilhado de objetos, o que permite o armazenamento de inúmeros objetos e vários formatos. O modelo de Sawicki relaciona outro objeto com o bloco de projeto, responsável pela comunicação entre os projetistas. Nesse sentido, cada bloco de projeto é composto também por sessões de *chat* envolvendo grupos de projetistas. Esse relacionamento pode ser visto na figura 3.4 logo adiante.

Em cada projeto existe um coordenador, diferente do modelo de Iochpe, no qual cada grupo fica responsável pela execução de, pelo menos, um sub-projeto. Em outras palavras, existe um membro responsável pelo projeto (chefe), com a função de coordenar os integrantes do grupo, além de organizar a inserção de novos participantes.

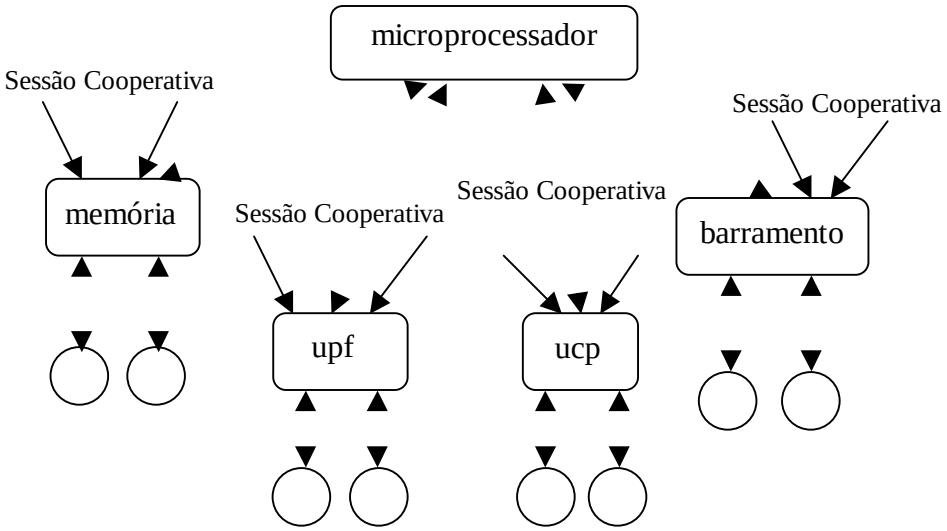


Figura 3. 4 - Organização e interação nos blocos de projeto

Esse modelo serviu como base para a implementação do *Collaborative Service*, no âmbito do Projeto Cave – descrito no capítulo 2. A criação do modelo de cooperação está descrita com mais detalhes no capítulo 5.

3.7 Ferramentas cooperativas

Os conceitos e idéias da área de trabalho cooperativo proporcionaram o desenvolvimento de diversos produtos e protótipos. Dentre alguns, incluem-se sistemas de mensagens, de conferência e videoconferência, sistemas de reunião, de co-autoria e *workflow.*

A seguir são apresentados alguns exemplos de sistemas de edição cooperativa, engenharia de *software*, destacando-se, de forma sucinta, suas principais características.

3.7.1 SEPIA – *Structured Elicitation and Processing of Ideas for Authoring*

O SEPIA é um sistema de autoria cooperativa de hiperdocumentos. Permite que um grupo de autores distribuídos possa produzir um documento hipermídia, tanto no modo síncrono e assíncrono quanto individual. Nele, o hiperdocumento é fragmentado em um conjunto de objetos (nós), que são armazenados em um banco de dados, podendo

ser acessado por todos os membros do grupo. Com o uso da tecnologia de hipertextos se torna mais fácil permitir alterações de forma concorrente e incremental.

Existe, também, suporte à comunicação síncrona entre os autores por intermédio de canais de áudio e videoconferência. Não há imposição de papéis aos autores, tampouco são armazenadas informações sobre qual autor tenha feito uma determinada alteração no hiperdocumento.

3.7.2 PREP – *Work in Preparation*

O PREP é um sistema de edição cooperativa assíncrono. Fornece suporte a interações sociais (comunicação de planos, definição de regras sociais e de comentários), opera sobre estações *Macintosh* ligadas por uma rede local.

Todos os colaboradores acessam um documento compartilhado que está armazenado em um banco de dados. Esse documento está organizado em segmentos e não equivale à divisão de um hipertexto em nós, pois não possui referências cruzadas ou internas, somente seqüenciais.

A área de trabalho é dividida em colunas, cada uma com informações diferentes do texto. O PREP utiliza atribuições de papéis (redator e revisor) sobre essas colunas para delimitar o controle de acesso. Por exemplo, um documento pode ser dividido em três colunas. A primeira pode conter o planejamento do segmento, a segunda o conteúdo, e a terceira o comentário. O PREP implementa um controle de versões, que impede que modificações feitas destruam por completo o conteúdo anterior do segmento. Uma desvantagem é que o PREP não implementa um mecanismo de comunicação entre os autores.

3.7.3 QUILT

O QUILT é um sistema de edição cooperativa de hipermídia. Combina aspectos de conferência por computador e permite através de uma rede UNIX que autores distribuídos editem um documento de forma assíncrona. Implementa um mecanismo de conferência assíncrono, através de troca de mensagens textuais e de voz, por canais de áudio. O QUILT disponibiliza mecanismos de atribuição de papéis (Leitor e Revisor), controlando o acesso ao documento [DIE 96].

O documento QUILT é formado basicamente por um nó base e nós de anotações ou comentários, associados por ligações hipertextuais, formando uma estrutura de árvore. Além disso, possui um mecanismo de versões, com a finalidade de manter um histórico das atividades. Implementa, também, um sistema de gatilho baseado na ocorrência de determinados eventos ou tempo decorrido, disparando um mecanismo de notificação.

3.7.4 CoWeb

O CoWeb é um sistema para suporte ao trabalho cooperativo via Web que permite que vários usuários distribuídos na Internet trabalhem sobre um mesmo documento HTML, especialmente *Forms*. Possibilita que usuários geograficamente distribuídos vejam e usem um mesmo documento HTML simultaneamente, de modo que os mesmos podem preencher cooperativamente campos de um formulário ou até mesmo marcar trechos importantes do documento.

O CoWeb converte elementos individuais do HTML em *applets* Java, que possibilitam a cooperação simultânea. De acordo com [JAC 2001], a heterogeneidade é um fator crucial para o sucesso da *Web* e novas aplicações somente terão sucesso se essa heterogeneidade não for derrubada.

Um dos objetivos do *CoWeb* é permitir uma comunicação de baixa velocidade, por redes de pequena largura de banda, que são bastante comuns na Internet. A arquitetura *CoWeb* tem um servidor que modifica os documentos HTML, substituindo, através de um *parser*, os elementos HTML por *applets* Java. O servidor *CoWeb* age também como elemento central da cooperação.

3.7.5 BSCW – *Basic Support for Cooperative Work*

O BSCW *Shared Workspace* é um sistema de armazenamento e recuperação de documentos que realiza suporte ao trabalho de grupos geograficamente dispersos. Contém um servidor que mantém um número de *workspaces* contendo objetos que são compartilhados pelos mesmos grupos, através de clientes WWW.

Segundo [BEN 98], a gerência de versões é realizada através de *check-in/check-out*, de modo a garantir que os membros sejam alertados do estado atual do documento. Quando um membro tenta recuperar um documento em *check-out*, recebe uma notificação de alerta informando quem está modificando o documento.

3.7.6 DISKEDIT

O DISKEDIT é um *toolkit* com a funcionalidade de estender editores mono-usuários para o campo de aplicações multi-usuário. Para [DIE 96], essa extensão é obtida através de pequenas alterações no código fonte dos editores e com o fornecimento de algumas primitivas genéricas que trabalham com as diferenças entre os editores.

Uma das suas vantagens reside no fato do grupo de projeto não precisar abandonar seus editores preferidos e, mesmo assim, conseguirem cooperar entre si. O DISKEDIT mantém uma cópia do estado do *buffer* para cada usuário, formando assim, uma arquitetura replicada. Além disso, possui um mecanismo de exclusão mútua, no qual cada usuário deve adquirir o *lock* sobre uma determinada região do texto antes de editá-la, garantindo que não haverá dois usuários editando a mesma região do texto. Uma das suas desvantagens está na ausência de mecanismos de comunicação síncrona (anotações, comentários) entre os autores.

3.7.7 MUT – *(Multiuser Talk)* & OCE – *(Graphic Object Cooperative Editor)*

O OCE é um editor gráfico cooperativo que permite a edição de figuras geométricas e anotações de texto de forma síncrona. A ferramenta MUT é um *talk* que suporta conferência baseada em texto. Foram desenvolvidos sobre a plataforma CORBA ORBIX, e ISIS de IONA (gerenciamento e difusão de mensagens) com mecanismos de reflexão computacional.

Para [FRI 97], o projeto dessas aplicações foi baseado em paradigmas de orientação a objetos para representar os elementos envolvidos na cooperação. Nessas ferramentas, cada participante de um estágio da atividade cooperativa atua como um cliente através de um ORB de um servidor distribuído de objetos de desenho (OCE) ou mensagem (MUT).

3.7.8 WebDAV – *Distributed Authoring and Versioning on the Web*

O *Web*DAV consiste em um grupo de trabalho formado pela IETF – *Internet Engeneering Task Force* e pelo W3C – *World Wide Web Consortium*, que trata da redação distribuída e da gerência de versões sobre a WWW.

Seu principal objetivo é estender o protocolo HTTP para que possa suportar as características acima, e com isso fornecer uma arquitetura aberta a nível de protocolo para o desenvolvimento de aplicações para cooperação assíncrona de documento *Web*, tanto em HTTP, quanto em XML.

O *Web*DAT inclui controle de acesso, *lock*, mecanismos de autenticação, criação, modificação, redirecionamento e recuperação de recursos e gerência de versões e mecanismos de *check-in / check-out* [IET 98].

3.7.9 ALLIANCE

O *Alliance* é um editor cooperativo que permite que usuários distantes e conectados pela Internet possam editar de forma assíncrona documentos estruturados. Um ponto importante no *Alliance* é o suporte a documentos estruturados, baseado na API THOT e no editor GRIF [DEC 95].

3.7.10 GROVE

O GROVE é um sistema de edição cooperativa de documentos estruturados na forma de árvore. Em uma rede local UNIX, permite que autores possam editar documentos de forma síncrona e assíncrona. O GROVE utiliza uma arquitetura replicada, com cópias dos documentos em cada estação operante e um coordenador central.

O GROVE permite que cada autor possa manipular uma ou mais visões de uma região do texto compartilhado através de um conjunto de janelas. A alteração realizada por um co-autor é transmitida para os demais de forma síncrona [SOU 96].

Além disso, possui um canal de áudio para comunicação informal entre os membros do grupo.

3.7.11 PROSOFT

O ambiente PROSOFT (desenvolvido no Instituto de Informática da UFRGS) tem como objetivo principal apoiar o desenvolvimento formal de *software*, fornecendo integração de dados, controle e de apresentação entre suas ferramentas. Sua construção foi influenciada pela estratégia *data-driven*, modelos e tipos abstratos de dados, orientação a objetos, método algébrico, dentre outros conceitos [REI 98].

A especificação do PROSOFT cooperativo suporta o uso concorrente de objetos de *software* (diagramas, código, etc), permitindo que um mesmo objeto seja manipulado por vários desenvolvedores. O PROSOFT cooperativo consiste de um conjunto de ATOs (Ambiente de Tratamento de Objetos) especificados para gerenciar a manipulação síncrona de objetos cooperativos do PROSOFT, permitindo a construção de aplicações *groupware* para o ambiente [REI 2000].

3.8 Ferramentas cooperativas na área de EDA *(Electronic Design Automation)*

Um ambiente CAD cooperativo é essencial para uma melhor consistência e produtividade no desenvolvimento de projetos envolvendo grupos de projetistas distribuídos. Esta seção descreve de forma sucinta algumas ferramentas cooperativas aplicadas em ambientes CAD na área de EDA.

3.8.1 STAR

O ambiente STAR foi desenvolvido em conjunto com a UFRGS e com o Centro Científico Rio da IBM Brasil, que aproveita a experiência desses dois grupos e objetiva implementar um ambiente que reúna as características mais importantes esperadas em um ambiente efetivamente aberto à integração de ferramentas destinadas a diferentes aplicações, arquiteturas e tecnologias de fabricação. STAR é baseado num modelo de dados derivado do ambiente GARDEN [VIE 89, QUI 90] e oferece recursos especiais para a gerência de metodologias de projeto e para o suporte à cooperação entre projetistas [WAG 91].

O modelo de dados adotado no STAR é uma versão ligeiramente modificada e estendida do modelo de dados GARDEN. A idéia básica era continuar com a existência de um modelo flexível e poderoso, combinado com mecanismos de gerenciamento de versões que independem da metodologia de projeto. O STAR oferecia um recurso não disponível até o momento em outros ambientes de projeto, a definição de hierarquias de metodologias de projeto, possível em função de sua flexibilidade no modelo de dados [WAG 91].

Segundo [WAG 2001], o ambiente STAR suporta um sistema de base dados estruturado hierarquicamente em diversos níveis. O nível superior representa a Base de Dados Pública, na qual estão armazenados objetos acessíveis a quaisquer usuários (projetistas ou grupo de projetistas) que possuam acesso ao ambiente. O nível de folhas da hierarquia é constituído pelas Bases de Dados Privativas, em que cada usuário armazena seus objetos, os quais, a princípio, não serão vistos por outros usuários. É importante destacar, que um modelo de cooperação define as regras que permitirão a visibilidade de objetos em diferentes bases de dados.

3.8.2 CollabTop

A ferramenta CollabTop, proposta por Konduri [KON 99], é uma extensão da ferramenta distribuída WebTop. O seu objetivo é criar um ambiente em que projetistas possam ter uma visão consistente do projeto em seu ambiente de trabalho. Foi desenvolvida como um *Applet* Java e usa mecanismos CGI para a comunicação com ferramentas. O servidor é um programa *stand-alone* Java que propaga um *broadcast* para os demais clientes.

Esse ambiente cooperativo faz com que um evento causado por um projetista se propague para os demais projetistas envolvidos no projeto. O CollabTop é capaz de juntar projetistas em sessões e várias sessões são capazes de rodar em paralelo como ilustra a figura 3.6.

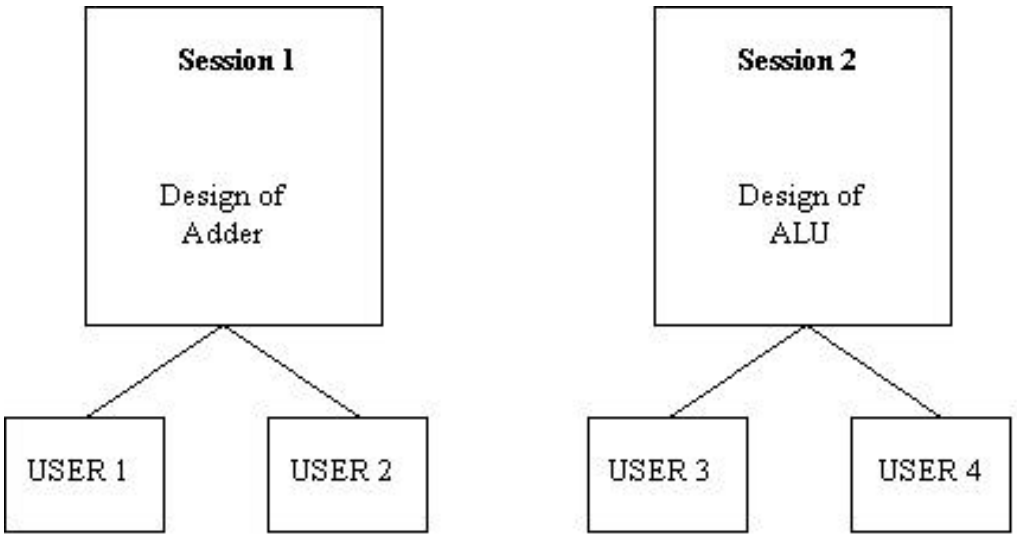


FIGURA 3. 5 - Colaboração entre várias sessões [KON 99]

Em sua arquitetura, quando um projetista edita um esquemático, vários eventos são gerados. Esses eventos podem ser propagados para todos os outros projetistas *(peer-to-peer)* ou enviados para um servidor central que faz um *broadcast* para os outros participantes. Essa comunicação pode ser visualizada na figura 3.6. É importante destacar que CollabTop foi implementado usando o modelo cliente-servidor.

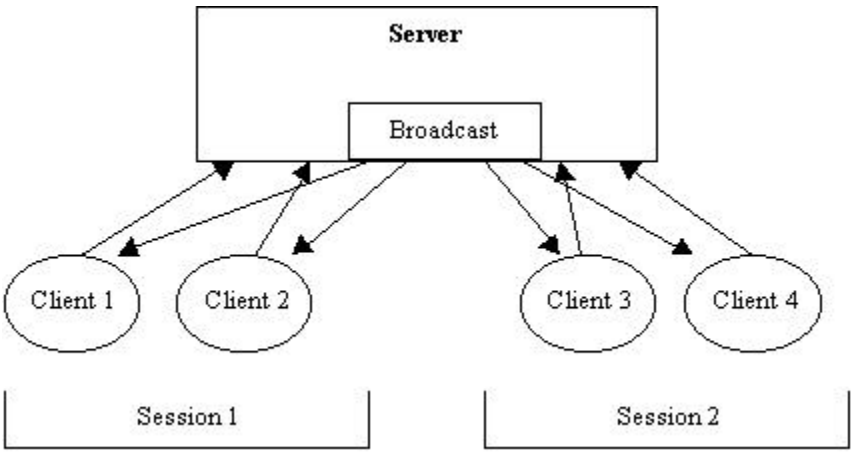


FIGURA 3. 6 – Eventos gerados pelo *broadcast*

3.8.3 Quants

Kirowski [KIR 2001] apresenta uma abordagem baseada em hipermídia, cujo propósito é auxiliar a grupos de projetistas que estão envolvidos na integração de blocos IP. A técnica para tal abordagem é baseada em mecanismos de fluxo de dados, chamados de semântica *multicast*, e o formato apresentado é denominado *quants*. O ponto chave da pesquisa de Kirowski é o *quant*, uma primitiva que encapsula informações multimídia, e *hiperlinks*. Os *quants* foram projetados para serem reutilizados para muitos propósitos.

Esta abordagem não apresenta muitas vantagens para suporte a projetistas nas suas atividades fim. O conceito de reusabilidade de *quants* é praticável somente quando seu potencial de reuso é realmente significativo, isso porque seu custo de criação é muito alto. Contudo, com a abordagem da semântica *multicast*, pode ser mais fácil a coordenação e a comunicação entre grupos de projetistas. Um ponto crítico, todavia, é o desempenho.

3.8.4 OmniFlow

Desenvolvido no *Collaborative Benchmarking Laboratory* na Universidade do Estado da Carolina do Norte, OminiFlow [LAV 2000, BRG 2001] compõe alguns mecanismos (linguagens de marcação, linguagens de descrição de *hardware* e programação estruturada) para se conseguir escalabilidade, bem como flexibilidade em sistemas de *workflow*.

Segundo [IND 2002] o modelo *workflow* é baseado nos conceitos das linguagens de marcação, expandido devido ao sucesso do HTML como a principal linguagem na construção de documentos WWW. OmniFlow usa XML para capturar a decomposição de todo fluxo dentro de uma hierarquia de tarefas. Um esquema XML – chamado cdtML – foi definido para permitir a validação dos modelos *workflow*

processados pelo OmniFlow. Baseado nesse esquema, um modelo *workflow* pode ser analisado e uma GUI pode ser dinamicamente criada por descrições XML. Com isso, o usuário pode ver, editar e executar o fluxo de tarefas.

3.9 Metodologias de cooperação

Essa seção descreve de forma sucinta duas metodologias de cooperação: WYSIWIS e *Pair-Programming.* A metodologia WYSIWIS trata da visualização da aplicação na forma compartilhada; a *Pair-Programming* é um estilo de programação em par, executado de forma local. Esse estilo de programação foi estendido para uma execução em máquinas remotas, encaixando-se no escopo desse trabalho.

3.9.1 Metodologia WYSIWIS (What You See Is What I See)

Ao se projetar uma interface multiusuário, além das dificuldades das interfaces monousuárias convencionais, aparecem novas dificuldades inerentes ao próprio trabalho em grupo. No contexto dessas dificuldades, está o conceito de percepção ou consistência entre os outros usuários *(user awareness).* Esse conceito altera radicalmente os paradigmas tradicionais para o projetos de interfaces gráficas, pois atualmente as interfaces monousuárias são projetadas para que os usuários não tomem conhecimento dos demais usuários que possam estar compartilhando, por exemplo, uma mesma base de informação.

A primeira questão associada à percepção dos usuários diz respeito à visualização da aplicação compartilhada. Uma possibilidade é usar interfaces WYSIWIS *(What You See Is What I See)* [STE 87], em que todos os usuários compartilham a mesma visão de interface. A grande vantagem do WYSIWIS é ser de simples implementação e fornecer um forte senso de contexto compartilhado. Por exemplo, é possível referir-se a algo pela sua posição na tela. No entanto, interfaces WISIWIS são inflexíveis e apresentam uma série de inconvenientes: não permitem que usuários tenham espaço privativo, em conseqüência, a tela fica cheia de janelas que só estão sendo usadas por outros usuários e os cursores dos outros usuários na maioria das vezes atrapalham o trabalho.

3.9.2 Metodologia *Pair-Programming*

A abordagem de um projeto cooperativo no Ambiente Cave é feita a partir da escolha de uma infra-estrutura tecnológica, o que envolve armazenamento de objetos e mecanismos de comunicação, entre outros, e também de uma metodologia de cooperação. O capítulo 3 mostra a tecnologia *Jini/Javaspaces* adotada para suportar a cooperação entre projetistas de CIs. Nesta seção, é apresentada a metodologia de cooperação chamada de *Pair-Programming*, a qual foi utilizada como estudo de caso para validar a interação dos grupos de projeto e estendida para um funcionamento em máquinas remotas.

Pair-Programming é um estilo de programação em que dois programadores trabalham lado a lado em um computador, colaborando continuamente num mesmo projeto, algoritmo, código ou teste. O uso dessa prática demonstra um aumento de produtividade e qualidade nos programas. Adicionalmente, programadores utilizando essa metodologia concordam que se adquire mais confiança na solução de problemas quando se trabalha em pares ao invés de sozinho. No entanto, muitos programadores estão condicionados a trabalharem sozinhos, oferecendo geralmente resistência a essa transição [WIL99, WIL2000].

Uma metodologia de desenvolvimento chamada *Extreme Programming* (XP) atribui seu grande sucesso ao *Pair-Programming.* A evidência desse sucesso é tão impressionante que fez despertar a curiosidade de muitos consultores e pesquisadores de Engenharia de Software altamente respeitados [WIL2000a, WIL2000b, COC 2000]. O maior exemplo é o sistema de compensação da Chrysler lançado em 1997. Após achar erros significantes em sua fase inicial, Kent Beck e Ron Jeffries – criadores da técnica XP – recomeçaram o processo utilizando os princípios de XP. Esse sistema paga aproximadamente 10.000 empregados por mês, possui 2.000 classes, 30.000 métodos e ainda está operacional hoje em dia. Adicionalmente, programadores da Ford gastaram quatro anos tentando construir o VCAPS *(Vehicle Cost and Profit System)* usando uma metodologia de cascata tradicional. Desenvolvedores XP implementaram satisfatoriamente em menos de um ano [WIL2000].

XP defende *Pair-Programming* com tanto fervor que mesmo a prototipação feita sozinha é jogada de lado e reescrita com um colega. Um elemento chave é que, trabalhando em duplas, um contínuo trabalho de revisão de código é produzido. Nota-se que é enorme o número de erros óbvios, mas não notados, tanto que são descobertos pela outra pessoa [WIL2000].

Pensamentos como “sou um programador ruim”, “se programarmos juntos você estará perdendo seu tempo”, ou “eu sou um programador excelente e estou fazendo par com esse cara ruim” não devem ter espaço nessa técnica. Senão, a relação de colaboração terá sido destruída. Nenhum de nós, independentemente de quão boa seja nossa habilidade, é infalível, fazendo com que as contribuições dos outros sejam insignificantes. John von Neumann reconhecia suas próprias inadequações e constantemente pedia para que outros revisassem seu trabalho. Ninguém duvida da genialidade de von Neumann. Sua habilidade de reconhecer a limitação humana o fez despontar entre um mar de pessoas. Em média, as pessoas podem ser treinadas para aceitar a sua humanidade – sua inabilidade de funcionar como máquinas – valorizando-a a ponto de controlá-la para o sucesso da programação [HER 2001a].

O Projeto Cave sofreu uma profunda evolução, como descrito da seção 2.2, em que focou sua pesquisa a partir do trabalho cooperativo. Muitas modificações foram realizadas para que o *framework* pudesse ter uma maior flexibilidade e melhor suporte para projetos de CIs complexos.

A abordagem descrita pela metodologia *Pair-Programming* enfatiza o trabalho em somente uma máquina. O parceiro de projeto pode contribuir com sugestões, detecções de erros, entre outras coisas. A proposta, inserida no contexto do Projeto Cave, estendeu essa metodologia tornando-a executável em máquinas diferentes, chamando-se de *Paar-Programming.*

3.10 Resumo do Capítulo

Este capítulo mostrou alguns conceitos básicos para o entendimento do que seja trabalho cooperativo. Esses conceitos servem como base para a criação do modelo de cooperação proposto para este trabalho. Nesse sentido, alguns aspectos foram destacados, tais como CSCW e *groupware,* tecnologias de suporte, ambientes e sistemas colaborativos, etc., além do estado da arte de ferramentas cooperativas relacionadas com EDA *(Electronic Design Automation)*. Além disso, este capítulo descreveu de forma sucinta as principais características de três modelos de cooperação propostos na literatura: Käfer, Nodine & Skarra, e Iochpe. Introduziu também um quarto modelo, criado por Sawicki [SAW 2002], no âmbito do Projeto Cave.

4 Tecnologias *Jini/Javaspaces*

4.1 Introdução

Este capítulo tem o objetivo de descrever a arquitetura das tecnologias *Jini* e *Javaspaces*. Essas tecnologias serviram como infra-estrutura para validar o modelo de cooperação proposto neste trabalho. Com o intuito de entender melhor seus mecanismos, vários pontos são destacados, tais como o funcionamento da arquitetura, conexões, mecanismos de comunicação, serviços, entre outros.

O módulo de cooperação utiliza-se de alguns serviços disponíveis na tecnologia *Jini*. Por exemplo, o armazenamento de objetos é realizado pelo serviço de persistência chamado *Javaspaces*. Esse serviço possibilita que o projetista armazene seu projeto em um repositório distribuído na rede e consiga recuperá-lo posteriormente. Esse serviço é descrito com mais detalhes na seção 4.10.

Neste contexto, o armazenamento de dados utiliza-se de outro serviço da tecnologia *Jini*, chamado serviço de transações *(Transaction Service)*, o qual implementa as quatro propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). A seção 4.11 descreve esse serviço com mais detalhes. Outros mecanismos de comunicação utilizados na arquitetura cooperativa são descritos nas próximas seções desse capítulo e foram base para validar o modelo de cooperação proposto.

4.1.1 *JINI – (Java Intelligent Network Internet)*

A tecnologia *Jini* foi criada para ser usada em todas as espécies de redes e dispositivos eletrônicos. Permite a conexão de qualquer aparelho de consumo ou de uso comercial em qualquer ponto da rede. Graças à sua simplicidade, remove as tradicionais barreiras de compatibilidade, confiabilidade e administração que impediram, no passado, a oferta de redes heterogêneas. A *Sun Microsystems*, empresa responsável pela linguagem de programação Java, relata que os lares inteligentes estão prestes a serem cenários reais. Lançada na Conferência Mundial de Analistas da *Sun*, a tecnologia *Jini* integra a próxima geração de produtos e aparelhos domésticos e profissionais que estão sendo desenvolvidos por mais de trinta líderes da indústria eletrônica [SUN 2002].

O que a tecnologia Java oferece para a independência de plataformas, a tecnologia *Jini* contribui pela interação entre dispositivos. Devido à independência de plataforma proporcionada pela linguagem Java, a tecnologia Jini não exige nenhum sistema operacional, processador ou ambiente de desenvolvimento específico. Essa tecnologia permite basear inúmeros aparelhos especialmente projetados para operar em rede. Além disso, está fundamentalmente relacionada a sistemas distribuídos, pois considera a inexistência de um servidor central. A falha de um dispositivo não afeta os demais, o que resulta em um ambiente mais flexível e adaptável.

Em uma rede que dispõe de *Jini*, cada dispositivo é reconhecido como um objeto, o que resulta num sistema distribuído que é entendido como um conjunto de serviços distribuídos. O mecanismo básico da tecnologia *Jini* é a habilidade de ligar, pesquisar na rede outros dispositivos e serviços em ação. Os dispositivos se identificam automaticamente, sem precisar de *drivers* de conectividade, e são aceitos no seu grupo, como uma espécie de *plug-and-play*, com capacidade de formar rede de equipamentos heterogêneos.

Um sistema *Jini* é baseado na idéia de *grupos* de usuários e de recursos por eles requeridos. A meta geral é tornar uma rede flexível, com ferramentas facilmente administradas, no qual os recursos possam ser encontrados sem nenhuma dificuldade. A tecnologia *Jini* é um conjunto de APIs Java e protocolos de rede que ajudam a construir sistemas distribuídos organizados como uma federação[1] de serviços. Os dispositivos podem ser vistos tanto como dispositivos de *hardware*, quanto programas ou a combinação dos dois. Seu foco principal é tornar a rede uma entidade dinâmica, refletindo melhor o trabalho em grupo com a habilidade de adicionar e retirar serviços de uma maneira flexível [EDW 99]. Um sistema *Jini* é formado pelas seguintes partes:

- ?? um modelo de programação que suporte e encoraje a produção de serviços distribuídos confiáveis;
- ?? um conjunto de componentes que possibilitam a infra-estrutura de agrupamento de serviços em um sistema distribuído;
- ?? serviços que podem fazer parte do sistema *Jini* e que ofereçam funcionalidade para qualquer outro membro do grupo;
- ?? habilitação dos usuários para que possam compartilhar serviços e recursos através da rede;
- ?? oferecer aos usuários acesso fácil aos recursos em qualquer lugar da rede.

O sistema *Jini* amplia o ambiente Java de uma única máquina virtual para uma rede de máquinas. O ambiente Java proporciona uma boa plataforma para computação distribuída, pois ambos, código e dados, podem locomover-se de máquina para máquina. O resultado é um sistema em que a rede suporta uma configuração em que objetos possam se mover de um lugar para outro, o quanto for necessário, e que possam invocar qualquer parte da rede para realizar as operações.

A arquitetura *Jini*, portanto, adiciona mecanismos ao ambiente Java, permitindo que os componentes fluam em um sistema distribuído, estendendo a movimentação para um sistema de rede. A infra-estrutura dessa tecnologia fornece um mecanismo para que dispositivos, serviços e usuários se conectem e desconectem da rede. Conectar e sair de um sistema Jini é uma tarefa simples, pois são sistemas muito dinâmicos, ao contrário das redes de hoje, em que a configuração é uma tarefa centralizada e feita à mão.

[1] Termo utilizado para expressar conjuntos de serviços relacionados

Cada dispositivo, para ser compatível com *Jini*, deve ter alguma memória e poder de processamento. Mas, mesmo dispositivos sem essas características podem ser conectados a um sistema Jini através de um *proxy*. É importante destacar que componentes do sistema Jini são implementados através da linguagem Java [EDW 99]. As *camadas* da tecnologia Jini estão ilustradas na figura 4.1 a seguir.

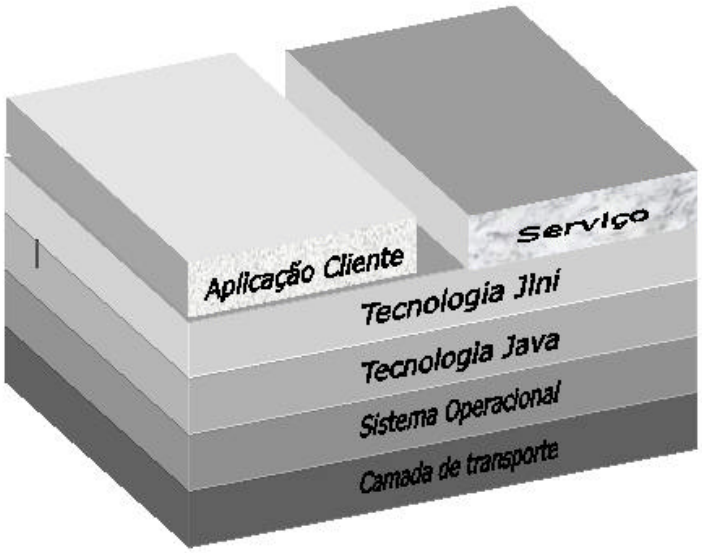


FIGURA 4. 1 - Camadas da tecnologia *Jini*

A tecnologia *Jini* é um conjunto de APIs que auxiliam a construção de sistemas distribuídos organizados como uma *federação* de dispositivos. Esses serviços estão disponíveis a clientes (um programa, outro serviço ou usuário) e podem ser combinados para atingir um determinado objetivo. Um serviço pode ser qualquer coisa na rede que possa realizar uma tarefa útil. Dispositivos de *hardware*, *software* e até mesmo usuários podem ser serviços. Uma impressora compatível com *Jini*, conectada na rede, pode ser um serviço de impressão.

O cliente utiliza vários serviços disponíveis para realizar uma determinada tarefa. Para entendermos melhor como *Jini* funciona, podemos verificar que no centro da tecnologia *Jini*, está localizado o *Lookup Service* (descrito nesse trabalho com o TLS – Tabela de Locação de Serviços). Para fazer parte de uma rede *Jini* (comunidade *Jini*) é usado o protocolo de descoberta *(discovery protocol)* para encontrar o *Lookup Service*. Logo após, é utilizado o protocolo de junção *(join protocol)* para registrar e tornar os serviços disponíveis para aplicações clientes. Logo que os serviços e os clientes forem unidos, não se necessita mais da assistência do *Lookup Service*, pois é possível trabalhar diretamente com quem requisitou o serviço.

A tecnologia *Jini* serve como um *link* entre a aplicação cliente e qualquer serviço externo disponível na rede (acessar CD ROM, impressão, etc). Por exemplo, um *drive* de CD ROM, quando conectado na rede, é visto pelo serviço de *Lookup*, pois recebe um pedido do dispositivo para ser registrado (no grupo CD ROMs). Quando o

Lookup Service aceita o pedido, o *drive* envia um arquivo *bytecode*, o qual estabelecerá a comunicação entre a aplicação cliente e o *drive*. O arquivo é armazenado em uma tabela que contém os dados de todos os serviços do *Lookup Service* disponíveis na rede *Jini*.

Quando uma aplicação cliente requer o uso de um *drive* de CD ROM, o *Lookup Service* envia uma lista de serviços de CD ROMs disponíveis. O cliente escolhe um ou outro, notificando ao *Lookup Service* sobre a sua escolha. O *Lookup Service* envia, logo após, o *bytecode* previamente obtido pelo CD ROM e o transfere para a aplicação cliente. A aplicação cliente usa esse código para se comunicar diretamente com o CD ROM e dispensa o *Lookup Service* [MOR 2000]. Essa representação pode ser vista na figura 4.2 a seguir.

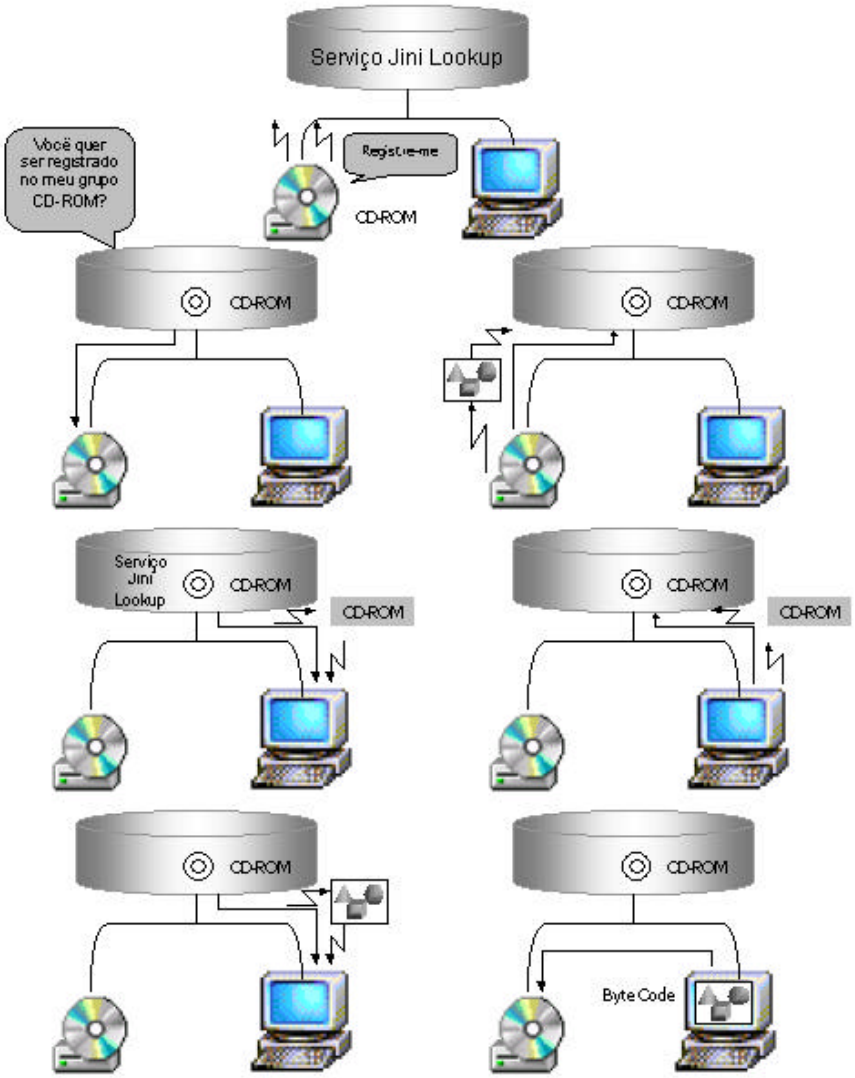


FIGURA 4. 2 - Registrando serviço do *Lookup Service*

A idéia por trás dos grupos de trabalho *(federação)*, é baseada na visão *Jini* de rede, na qual não existe uma autoridade central de controle. Como nenhum serviço está no comando, o conjunto de serviços disponíveis na rede formam uma *federação*. Ao invés de uma autoridade central, a infra-estrutura Jini fornece uma maneira para que os clientes e serviços se achem (através do *lookup*, que armazena a lista de serviços disponíveis). Quando dois ou mais serviços se encontram, ficam independentes do sistema. O cliente e seus vários serviços realizam suas tarefas não dependendo mais da infra-estrutura Jini. Caso aconteça algum problema com o *Lookup Service*, todos os outros sistemas distribuídos que foram montados antes do problema vão continuar funcionando [EDW 99].

4.2 O Funcionamento da tecnologia *Jini*

O sistema *Jini* define uma infra-estrutura de *runtime* que fica situada na rede e fornece mecanismos que possibilitam que se coloque, retire, localize e acesse serviços. Essa infra-estrutura aparece na rede em três lugares: nos serviços de *lookup*; nos provedores de serviços (dispositivos compatíveis com *Jini*) e nos clientes. Quando novos serviços se tornam disponíveis na rede, registram-se no *Service Lookup* (TLS). Quando um cliente precisa localizar um serviço para realizar uma determinada tarefa, consulta o serviço de *lookup.*

A infra-estrutura *runtime* de *Jini* utiliza um protocolo ao nível de rede chamado *discovery* e outros dois protocolos ao nível de objetos chamados: *join* e *lookup.* O *discovery* faz com que os clientes e serviços localizem o *lookup.* O *join* possibilita que os serviços se registrem no serviço de *lookup*; o protocolo *lookup* permite que um cliente procure os serviços registrados para que o auxiliem a realizar algum tipo de objetivo [SUN 99b]. A figura 4.3 ilustra a requisição de serviços em uma arquitetura *Jini*:

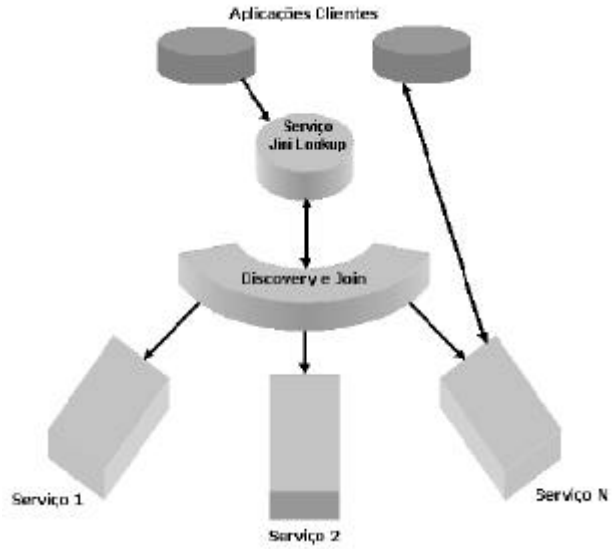


FIGURA 4. 3 - Requisição de serviços em uma rede *Jini*

4.3 Conexão *unicast*

Para definir uma conexão *unicast*, precisa-se saber o endereço IP e o número da porta onde o serviço de *lookup* está sendo executado. O dispositivo envia um pacote *unicast* para descobrir o número IP e a porta. Em resposta, o serviço de *lookup* enviará um pacote de anúncio de *unicast* ao dispositivo depois que o pacote de descoberta seja reconhecido e aceito. O aplicativo Jini usa a classe *net.jini.core. Discovery. LookupLocator* para especificar o reconhecimento IP e a porta utilizada. Para isso, usa-se uma *string* chamada Jini URL.

O formato *Jini* URL é jini://MachineIP:PORT. Depois de criar um *LookupLocator* com a URL exigida, é preciso chamar o método *getRegistrar()*, o qual concede um serviço *lookup* conhecido como um objeto *ServiceRegistrar*. Todos os protocolos de descoberta *unicast* fundamentais são encapsulados dentro da classe *LookupLocator*. Com isso, não é preciso se preocupar sobre como essa tarefa será realizada.

O fragmento de código que adquire o objeto *ServiceRegistrar* é mostrado abaixo. O objeto de *LookupLocator* é criado baseado na *string* passada em *Jini* URL. Se a *string Jini* URL não tiver o número de porta mencionado, por *default* o número 4160 é assumido. A seguir são destacados alguns passos relativos à conexão *unicast* e na caixa de diálogo é descrita a chamada do método *getRegistrar()*:

a) instanciar um objeto *LookupLocator* para o serviço de *lookup* indicado;
b) usar o IP e número de porta passados. (Se nenhum número de porta é especificado, a porta 4160 é que será usada);
c) invocar o *getRegistrar()* no *LookupLocator* para executar uma conexão *unicast* no serviço de *lookup*. O serviço de *lookup* é indicado pelo *host* e o número da porta passado ao construtor.

```
ServiceRegistrar registrar = LookupLocator.getRegistrar();
```

4.4 Conexão *multicast*

A conexão *multicast* é usada quando os locais de serviços *Jini Lookup* não são conhecidos. A rede deve habilitar *multicast* configurando o roteador para localizar o serviço *lookup* de *Jini* fora de seu domínio de rede.

A conexão *multicast* é uma espécie de radiodifusão controlada; os pacotes para uma conexão *multicast* podem cruzar *gateways*. Fixando um TTL *(Time-To-Live)*, o

parâmetro pode controlar a gama de pacotes *multicast*. Este parâmetro de TTL especifica o número de *gateways* que os pacotes podem cruzar. Por *default*, o valor de TTL é 15. Enviando pacotes de conexão *multicast* em cima de uma gama de rede designada pode-se descobrir um serviço *lookup* de Jini ou, se necessário, dentro de grupos de *multicast* específicos. Uma vez que os pacotes *discovery* são detectados, os serviços de *lookup* enviarão pacotes de anúncio de *multicast* na rede que poderia obter uma aplicação, a qual age como um *DiscoveryListener*. Um *DiscoveryEvent* será enviado ao *DiscoveryListener* que, em troca, provê um conjunto de objetos *ServiceRegistrar*. Nota-se que o dispositivo pode obter respostas de mais de um serviço *Jini Lookup*.

Os passos para uma conexão *multicast* são os seguintes: primeiro cria-se um objeto de *LookupDiscovery* e uma implementação do objeto *DiscoveryListener*. Então, o método *addDiscoveryListener()* é invocado para fazer o *DiscoveryListener* escutar os pacotes dos serviços de *lookup*. O *LookupDiscovery*, ao receber os pacotes, chama o método de *DiscoveryListener* apropriado. O método *discared* será chamado quando um serviço de *lookup* sair da rede.

4.5 Processo de *discovery*

Imaginando que exista um disco rígido habilitado para Jini e que ele forneça um serviço de armazenamento, no momento em que este dispositivo for conectado na rede, manda uma mensagem anunciando sua presença através de um pacote *multicast* para uma porta já conhecida do dispositivo. Nesta mensagem de presença, ele inclui um número IP e um número de porta, no qual o disco poderá ser encontrado pelo serviço de *lookup*.

O serviço de *lookup* fica monitorando a porta conhecida por todos, esperando por pacotes de anúncio. Quando um serviço de *lookup* recebe um pacote, ele o analisa para saber se deve ou não responder ao serviço que o enviou. Se esse for o caso, o *lookup* contata o serviço fazendo uma conexão TCP com a porta e o endereço IP contidos no pacote. Usando RMI, o serviço de *lookup* envia um objeto, chamado *ServiceRegistrar*. A finalidade desse objeto é facilitar comunicações futuras com o serviço de *lookup*. Usando métodos desse objeto o serviço que enviou o pacote de anúncio pode realizar o registro *(join)* no serviço de *lookup* [EDW 99].

4.6 Processo *join*

Depois que o provedor de serviços recebe o objeto *registrar*, que é o produto final do processo *discovery*, ele está pronto para registrar o serviço *(join)*, ou seja, está pronto para se tornar parte de uma *federação* de serviços que estão registrados no serviço *lookup* (TLS). Para realizar um *join*, o provedor de serviço deve invocar o método *register()* no objeto *registrar*, passando como parâmetro um objeto chamado *ServiceItem*, que descreve o serviço. O método *register()* envia uma cópia do seu argumento para o serviço *lookup*, onde é guardado. Quando este procedimento acaba, o processo *join* está terminado e o serviço está registrado na *federação* (grupo de serviços).

O *ServiceItem* contém vários objetos, incluindo um objeto chamado *service*, que o cliente pode usar para interagir com o serviço. O *ServiceItem* pode conter também um certo número de atributos que podem ser qualquer objeto. Os atributos podem ser ícones, classes que fornecem GUIs para o serviço e objetos que possuem mais informação sobre o serviço.

Os objetos *service* normalmente implementam uma ou mais interfaces por onde os clientes podem interagir com o serviço. Por exemplo, o serviço *lookup*, é um serviço Jini, e o seu objeto *service* é o *ServiceRegistrar*. O método *register ()*, invocado pelo provedor de serviço durante um *join*, é declarado na interface *ServiceRegistrar*, que todos os objetos *ServiceRegistrar* implementam. Clientes e serviços conversam com o serviço *lookup* por este objeto invocando seus métodos declarados na interface [EDW99].

4.7 Processo de *lookup*

Depois que o serviço se registrou na TLS *(Service Lookup)* através de um *processo join*, este serviço está disponível para clientes que o procurem na tabela de locação de serviços. Para construir um sistema distribuído composto de serviços que cooperam para atingir um objetivo, um cliente deve localizar e solicitar a ajuda de cada serviço. Para achar um serviço, o cliente deve procurá-lo em um serviço de *lookup* através de um processo também chamado de *lookup.*

Quando realizamos um *lookup*, o cliente invoca o método *lookup()* em um objeto *registrar*. O cliente passa como argumento para o *lookup()*, um objeto que serve como critério para a procura. Esse objeto pode incluir um *serviceID*, que identifica um serviço único e atributos, que devem ser iguais aos do provedor de serviço. Pode-se usar também máscaras para os campos. Uma máscara no campo ID vai trazer qualquer serviço daquele tipo. O método *lookup()* envia o seu argumento para o serviço *lookup* que faz a procura e devolve zero ou mais serviços que satisfazem as condições. O cliente recebe a referência aos objetos dos serviços através do valor de retorno do método *lookup()*.

Quando um cliente procura um serviço através de um tipo Java, geralmente busca por uma interface. Por exemplo, se um cliente quer usar uma impressora, ele vai procurar pela interface genérica da impressora que é padronizada. Todas as impressoras devem implementar esta interface. O serviço *lookup* irá retornar um ou mais objetos de serviço que implementam esta interface. Atributos podem ser incluídos para diminuir o número de serviços encontrados. O cliente, dessa maneira, usará os serviços da impressora chamando-os em seu objeto de serviço os métodos declarados na interface padrão de impressora.

4.8 RMI (Remote Method Invocation)

4.9.1 Introdução

A área de redes de computadores sempre se preocupou com duas aplicações fundamentais. A primeira delas é a transferência de arquivos entre diferentes computadores, atualmente resolvida através de diversos protocolos como *smtp, ftp, http* e outros. A segunda aplicação é permitir que um *host* execute programas em outro *host*. Neste tópico, será destacado o *Remote Method Invocation (RMI)*, um mecanismo sofisticado para comunicação entre objetos Java distribuídos e que permite que programas Java chamem determinados métodos num servidor remoto.

A interface RMI permite que objetos Java em diferentes *hosts* comuniquem-se uns com os outros. Um objeto remoto reside no servidor. Cada objeto remoto implementa uma interface remota *(Remote Interface)* que especifica quais dos seus métodos podem ser executados pelos clientes. Os clientes podem executar os métodos do objeto como executariam métodos locais. Embora o conceito de RMI possa trazer uma visão de objetos se comunicando uns com os outros, o RMI é mais usado em um cenário cliente/servidor tradicional, no qual um único aplicativo servidor recebe conexões e pedidos de vários clientes. O RMI é simplesmente o mecanismo pelo qual o cliente e o servidor se comunicam.

4.9.2 Arquitetura RMI

O RMI tem como objetivos integrar um modelo de objeto distribuído na linguagem Java, sem quebrar e nem romper o modelo de objeto existente. Além disso, visa tornar a integração com um objeto remoto tão fácil quanto a integração com um objeto local.

Segundo [LEM 98], podemos ser capazes de usar objetos remotos exatamente da mesma maneira como se utilizássemos objetos locais - atribuí-los a variáveis, passá-los como argumentos para métodos, etc. A chamada de métodos em objetos remotos deve ser realizada da mesma maneira que as chamadas locais. Além disso, o RMI inclui mecanismos mais sofisticados para a chamada de métodos em objetos remotos para passar objetos inteiros ou parte de objetos por referência ou por valor, assim como exceções adicionais para o tratamento de erros de rede que possam ocorrer enquanto uma operação remota esteja acontecendo.

Um objeto remoto é um objeto a partir do qual métodos podem ser chamados por uma máquina virtual Java *(JVM – Java Virtual Machine)* diferente do local em que o próprio objeto reside; geralmente em uma JVM que está rodando em um computador diferente.

4.9.3 Camadas RMI

Em [LEM 98], as camadas RMI são divididas da seguinte maneira:

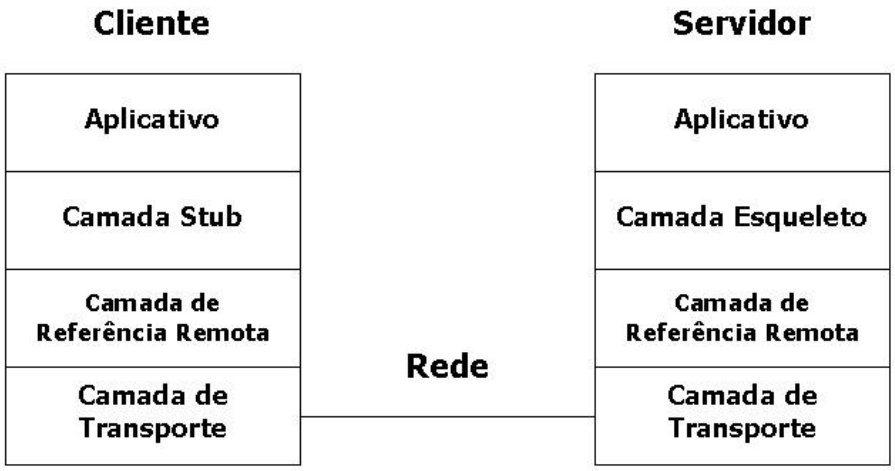


FIGURA 4. 4 - Camadas RMI

Ao observar-se as camadas *stub* e *esqueleto* no cliente e no servidor, respectivamente, vê-se que se comportam como objetos substitutos em cada lado, ocultando da implementação das classes a característica remota da chamada de objeto. O s*tub* é um substituto local do objeto remoto. A camada de referência remota, que trata do empacotamento de uma chamada de método e de seus parâmetros, retorna valores para o transporte através da rede. A camada de transporte representa a conexão de rede real de um sistema para outro.

A existência de três camadas para RMI permite que cada uma seja controlada ou implementada independentemente. Os *stubs* e *esqueletos* permitem que as classes de cliente e servidor se comportem como se os objetos com que estão lidando fossem locais e utilizem exatamente os mesmos recursos da linguagem Java para acessar esses objetos. A camada de referência remota separa o processamento do objeto remoto em sua própria camada, que pode então ser otimizada ou novamente implementada, independentemente dos aplicativos que dependem dela. A camada de transporte é utilizada independentemente das outras duas, de modo que se pode usar diferentes tipos de conexões de *socket* para RMI (TCP, UDP ou TCP com algum outro tipo de protocolo).

Então, quando um aplicativo cliente faz uma chamada de métodos remota, a chamada passa para o *stub* e depois para a camada de referência, que empacota os argumentos necessários e em seguida, passa através da camada de rede para o servidor. A referência do lado servidor desempacota os argumentos e os transmite para o *esqueleto* e depois para a implementação servidora. Sendo assim, os valores de retorno da chamada de métodos fazem o caminho de volta para o lado cliente.

Detalhes do estabelecimento de conexão entre *hosts* e a transferência de dados ficam escondidos nas classes RMI. Um *host* que permite RMI pode limitar a ação dos clientes remotos. Um objeto *SecurityManager* verifica todas as operações para certificação se elas são permitidas pelo servidor.

O empacotamento e a passagem de argumentos de métodos são um dos aspectos mais interessantes do RMI, pois os objetos precisam ser convertidos em algo que possa ser passado através da rede. Essa conversão é chamada de *serialização*. Desde que um objeto possa ser serializado, o RMI pode utilizá-lo como parâmetro de método ou como um valor de retorno. Os objetos que podem ser serializados incluem todos os tipos de primitivas Java, objetos Java remotos e quaisquer outros objetos que implementem a interface *Serializable*. Então pode-se dizer que serialização de objetos é um esquema que converte objetos em um fluxo de *bytes* que é transmitido para outras máquinas. Estas, por sua vez, reconstroem o objeto original a partir dos *bytes*. Subclasses da classe serializável também são serializáveis.

Uma tecnologia mais antiga, também desenvolvida pela Sun, é o *Remote Procedure Call (RPC)*, que faz muito que o RMI faz. O RPC é independente tanto de linguagem, quanto de processador, já o RMI é naturalmente independente de processador, mas limitado aos programas escritos em Java. RPC só pode enviar tipos primitivos de dados, enquanto que o RMI pode enviar objetos inteiros.

4.9 Javaspaces

4.10.1 Introdução

A arquitetura da tecnologia *Javaspaces* foi projetada para ajudar a resolver dois problemas relacionados: persistência distribuída e o projeto de algoritmos distribuídos. *Javaspaces* usa o serviço RMI e também a serialização de objetos, que caracterizam a linguagem de programação Java a realizar estas tarefas [FRE 99, HUP 2001]. Neste trabalho, *Javaspaces* serve como base de dados para o serviço de cooperação, garantindo o armazenamento e recuperação de projetos criados com as ferramentas CAD do Ambiente Cave.

É importante destacar que sistemas distribuídos são mais difíceis de se construir, pois requerem um pensamento cuidadoso dos problemas, se comparados com uma computação local. Os problemas primários são falhas parciais de *hardware*, latência e compatibilidade de linguagens. Com *Javaspaces*, pode-se trabalhar com vários repositórios distribuídos pela rede, formando um sistema tipicamente distribuído. A comunicação através da rede é realizada com o mecanismo RMI *(Remote Method Invocation)* tratado na seção 4.9.

4.10.2 Modelo de aplicação *Javaspaces*

Um serviço *Javaspaces* possui entradas. Uma entrada é um grupo tipado de objetos, expresso em uma classe para a plataforma Java, e que implementa a interface *net.jini.core.entry.Entry*. Uma entrada pode ser escrita em um serviço de *Javaspaces*, o qual cria uma cópia desta entrada no *Space* (serviço que armazena objetos Java) e que pode ser recuperada em operações futuras.

As entradas em um serviço *Javaspaces* usam *templates* (modelos), que são objetos de entrada que têm algum ou todo o seu conjunto de campos como sendo uma espécie de chave de recuperação. Assim, o *template* e os campos armazenados devem ser exatamente iguais para que o objeto armazenado seja recuperado posteriormente.

Há dois tipos de operações de *lookup*: *read()* e *take()*. Uma leitura *(read)* requisita um objeto que tenha o mesmo *template* passado por parâmetro para um repositório, e o mesmo devolve a entrada igual ao modelo no qual a leitura foi requerida ou uma indicação de que nenhum objeto foi encontrado. Um pedido *(take)* é uma operação semelhante à leitura. Porém, se o modelo foi encontrado, esta entrada é retirada do repositório. Também podemos pedir para um serviço *Javaspaces* notificar *(notify)* quando uma entrada de um modelo especificado é escrita.

São executadas todas as operações que modificam um serviço *Javaspaces* dentro de uma transação, tornando as operações seguras e confiáveis. Em outras palavras, cada entrada em um repositório pode ser transmitida no máximo uma vez. Porém, note que duas ou mais entradas em um repositório podem ter exatamente o mesmo valor [FRE 99].

A arquitetura da tecnologia *Javaspaces* suporta uma transação simples que permite atualizações de múltiplas operações em múltiplos espaços (repositórios). O modelo de transação utiliza o protocolo *commit* duas-fases, que é definido no pacote *net.jini.core.transaction* [EDW 99]. Esse mecanismo é descrito com mais detalhes na seção 4.11. Entradas escritas em um serviço *Javaspaces* são gerenciadas por um tempo *(lease)*, definidos no pacote *net.jini.core.lease* [FRE 99, KEN 99].

4.10.3 Persistência distribuída usando *Javaspaces*

A implementação da tecnologia *Javaspaces* provê um mecanismo para armazenar um grupo de objetos relacionados e recuperá-los com base em um *template* (modelo). Isto permite usar um serviço *Javaspaces* para armazenar e recuperar objetos em um sistema remoto [EDW 99].

Existem duas implementações do serviço *Javaspaces*. Uma transiente *(transient)* e outra persistente *(persistence)*. A diferença entre as duas é que, no serviço transiente, os dados são armazenados enquanto o serviço *Javaspaces* está sendo executado. Qualquer objeto armazenado será perdido caso o serviço seja finalizado ou haja uma falha na máquina onde o serviço está armazenado. No serviço *Javaspaces* persistente, os objetos são armazenados em disco. Com isso, pode ser recuperado seu estado atual, mesmo com a finalização ou falha no servidor *Javaspaces* [EDW 99].

A classe que implementa o serviço *Javaspaces* transiente é chamada *com.sun.jini.outrigger.TransientSpace.* A classe que implementa o serviço *Javaspaces* persistente é a *com.sun.jini.outrigger.FrontEndSpace.*

A implementação transiente está localizada no JARfile *transient-outrigger.jar* e a implementação persistente em *outrigger.jar*.

4.10.4 Benefícios

Os serviços *Javaspaces* são ferramentas para construir protocolos e aplicações distribuídas. São projetados para trabalhar com aplicações que podem se modelar como fluxos de objetos de um ou mais servidores. Se uma determinada aplicação pode ser modelada desse modo, a tecnologia *Javaspaces* pode prover inúmeros benefícios.

O *Javaspaces* pode prover um sistema de armazenamento distribuído seguro para objetos Java. A estratégia de segurança do armazenamento de objetos é tratada pelo serviço de transações que pode ser utilizado em conjunto com o serviço *Javaspaces*. Os acessos simultâneos de múltiplos clientes são resolvidos no serviço *Javaspaces*. Em outras palavras, esse serviço armazena e recupera automaticamente as entradas provendo mecanismos para transações distribuídas.

O serviço *Javaspaces* pode proporcionar persistência distribuída de objetos Java. Este é o grande beneficio de armazenar objetos, não só dados, de modo que se tem um repositório acessível para a computação cooperativa distribuída.

4.10.5 Metas e exigências

As metas para o projeto da tecnologia *Javaspaces* são:

- prover uma plataforma para projetar sistemas de computação distribuídos que simplifique o projeto e implementação desses sistemas;
- minimizar o número de classes do lado do cliente e manter o modelo no lado cliente simples para carregar rapidamente as classes;
- proporcionar uma implementação pequena do lado cliente, pois será executado em computadores com memória local limitada;
- possibilitar uma variedade de implementações, incluindo armazenamento de base de dados relacional e orientada a objetos;
- possibilitar reproduzir um serviço *Javaspaces*.

As exigências para aplicações *Javaspaces* clientes são:

- ser possível escrever puramente para um cliente na linguagem de programação Java;
- cuidar os detalhes da implementação do serviço por parte do cliente. As mesmas entradas e modelos têm que trabalhar do mesmo modo, não importando que implementação seja utilizada.

4.10.6 Algumas operações

Existem quatro tipos primários de operações que se pode invocar em um serviço *Javaspaces*. Cada operação tem parâmetros que são entradas, incluindo alguns que são modelos *(templates)* do tipo de entrada *(entry)*. Neste tópico são mostradas algumas operações de manipulação do serviço Javaspaces [FRE 99]:

??***write():*** escrever uma determinada entrada no serviço *Javaspaces*;
??***read():*** ler uma entrada do serviço *Javaspaces* que se iguala ao modelo;
??***take():*** ler uma entrada do serviço *Javaspaces* que se iguala ao modelo, removendo-a do *JavaSpaces*;
??***notify():*** notificar um objeto especificado quando as entradas se igualam ao modelo determinado são escritas no serviço *JavaSpaces*.

Tal como usado neste texto, o termo *operação* refere-se a uma única invocação de método; por exemplo, duas diferentes operações *take ()* e *read()* podem ter modelos diferentes *(takeIfExist() e readIfExit())*.

4.10 Transações Distribuídas usando Javaspaces

4.11.1 Introdução

O acesso a diversos itens de dados em um sistema distribuído é normalmente acompanhado de transações que têm que preservar as propriedades ACID – *atomicidade*: ou todas as operações da transação são propagadas corretamente no banco de dados ou nenhuma será; *consistência*: a execução de uma transação isolada (ou seja, sem a execução concorrente de uma outra) preserva a consistência; *isolamento:* cada transação não toma conhecimento de outras transações concorrentes no sistema; *durabilidade:* depois que uma transação completou com sucesso, as mudanças persistem, até mesmo quando houver falhas no sistema [KOR 91].

Há dois tipos de transações que devemos considerar. As transações locais, que são aquelas que mantêm acesso e atualizam diversas bases de dados locais; e as transações globais, que são aquelas que mantêm acesso e atualizam diversas bases de dados locais. No caso das transações globais, essa tarefa é mais complicada, pois diversos *sites* podem participar de sua execução. Uma falha de comunicação entre *sites* pode resultar em erros de processamento. No *Javaspaces*, é possível destacar quatro operações principais que permitem a manipulação de objetos: *write(), read(), take() e notify()*.

A transação é um conjunto de várias operações. Mas, sob ponto de vista do usuário, é uma só. Então, seguindo o exemplo de uma aplicação bancária, ao precisarmos fazer uma transferência entre duas contas, poderíamos naturalmente executá-la como duas operações em separado, mas agrupadas em uma transação. A primeira diminui da conta o valor desejado, e a segunda aumenta em uma conta diferente o mesmo valor.

Como estamos dentro de uma transação, o sistema protege as operações para que ambas se completem. Caso uma falhe, a transação não será completada, e os valores são restabelecidos na forma inicial. *Javaspaces* é o único serviço que suporta transações e, até mesmo no caso de *Javaspaces*, as transações são opcionais.

4.11.2 Como se efetua uma transação – protocolo duas fases

Primeiro, a transação gerente coleta todas as operações que compõem uma transação global. Estas são chamadas de participantes na transação. Depois, o gerente sinaliza cada um destes constituintes para a fase *precommit*. Isto significa que todos os participantes simplesmente guardam as operações em uma área temporária. Cada uma destas operações retorna um indicador para o gerente. O gerente, por sua vez, responde se a fase *precommit* obteve, ou não, sucesso.

A fase *precommit* é realmente uma requisição da transação gerente para cada um dos participantes individuais que devem estar preparados para um ou outro caminho – abortar ou efetuar a transação –. A Figura 4.5 mostra o que acontece na primeira fase: pode-se ver que a *Transaction Mannager* pergunta para cada uma das três operações participantes na transação se elas estão prontas para efetuar um *commit*. Neste caso, cada uma responde que está pronta.

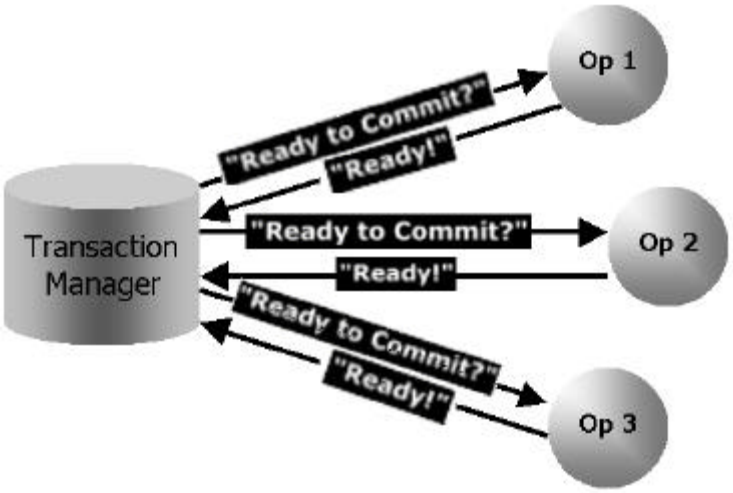


FIGURA 4. 5 - Fase *precommit*

A *Transaction Mannager* coleta todos os resultados dos participantes individuais para o *precommit*. Se algum destes participantes falha, o gerente aborta a transação. Isto significa que os participantes podem esquecer seus resultados temporários e não se armazena nada para o depósito fixo. Mas, se todos os participantes estão prontos para ir, o gerente faz com que cada um deles efetue a transação definitivamente *(commit)*. Ou todos os participantes abortam *(abort)*, ou todos efetuam a transação *(commit)*. A Figura 4.6 mostra, a seguir, este segundo estágio do protocolo, no qual o *Transaction Mannager* informa a cada operação participante que deve armazenar suas mudanças.

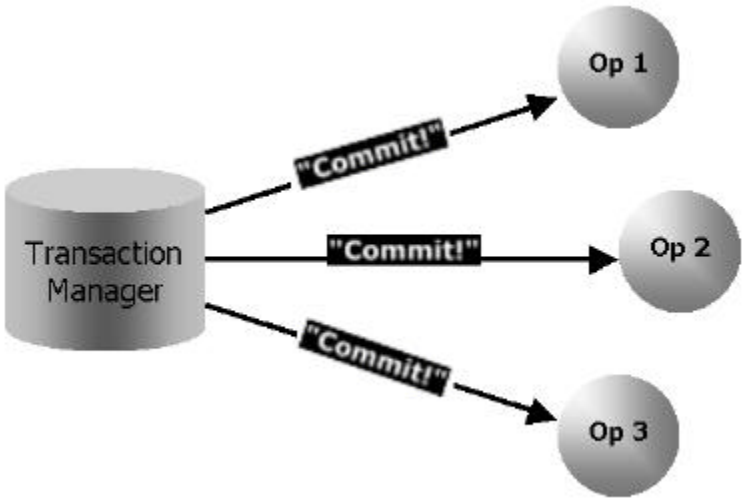


FIGURA 4. 6 - Fase *commit*

A *Transaction Mannager* é responsável por cobrir todos os participantes e controlar todos eles através de cada um dos estágios do processo – *precommit*, selecionando então, *abort* ou *commit*.

4.11.3 Transações e Javaspaces

Para se usar transações com operações baseadas em espaço *(based-space)*, tipicamente pergunta-se primeiro à *Transaction Mannager* para criar a transação e gerenciá-la por um determinado tempo *(lease)*. Então, a transação é passada para cada espaço da operação que gostaríamos que ocorresse. Admitindo não haver nenhum problema ao longo do caminho, então damos explicitamente um *commit* na transação, o que resultará em todas as operações completadas. Uma transação é ilustrada na figura 4.7 [EDW 99, KEN 99].

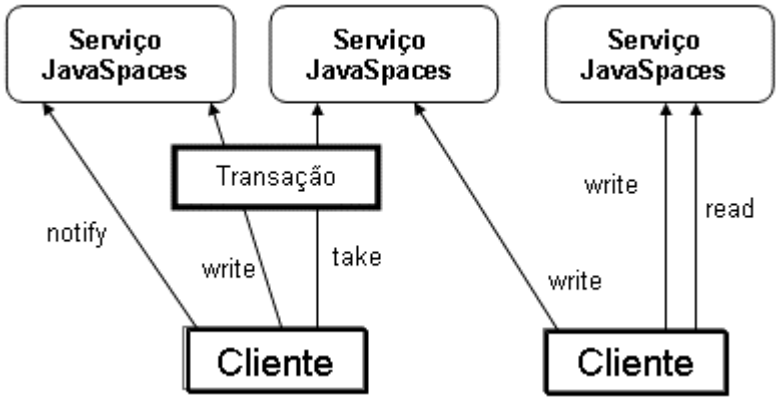


FIGURA 4. 7 - Transações entre diferentes *Javaspaces*

4.11.4 Participantes de uma transação

A tecnologia *Jini* provê um serviço de transação que gerencia um conjunto de participantes por meio de um processo de transação. O serviço de transação comanda os participantes até o *protocolo commit 2-fases*, que é um protocolo padrão que garante que todos os participantes completem suas respectivas operações na transação. No caso de falha em um participante, nenhum deles a completará. Se qualquer problema ocorrer, a transação é abortada, e será deixado o espaço *(space)* inalterado. A transação também pode ser abortada por meio da *Transaction Manager* se, por exemplo, o *lease* da transação for expirado.

Uma entrada *(entry*, visto na seção 4.10.2*)* é escrita dentro de um *space* sob uma transação. O *entry* é invisível a qualquer cliente que tente lê-la, pegá-la ou notificá-la fora da transação. Se a entrada for levada para o interior da transação, ela nunca será vista de fora dela. Se a transação abortar, o *entry* é descartado. Uma vez que a transação executa o *commit*, o *entry* estará disponível para leituras, recebimentos e notificações de fora da transação. Para maiores detalhes, veremos a semântica para se usar transações com as operações *space* [FRE 99].

4.11.5 Usando a *Transaction Manager*

Para fazer uso de transações, primeiro é preciso acessar a *Transaction Manager*, para criá-las e preservá-las. Como todos os serviços *Jini*, o serviço *lookup* retorna um *proxy*, objeto para *Transaction Manager*. Neste caso específico, fica-se esperando por um serviço que execute a interface *TransactionManager*. A seguir está expresso um “trecho” de código que demonstra a aquisição de uma Transaction Manager:

```
TransactionManager mgr = TransactionManagerAcessor.getManager();
```

O método estático *getManager()* da classe *TransactionManagerAcessor* é chamado, retornando para a *TransactionManager* o objeto *proxy*. Com esse *proxy* em mãos, podemos criar a transação que irá gerenciar um conjunto de operações. Vê-se a seguir um exemplo de como criar uma transação:

```
Transaction.Create trc = null
try {
   trc = TransactionFactory.create( mgr , 300000 );
}           catch           (Exception           e)           {
   System.err.println( “Could not create transaction” + e );
}
```

Primeiro, é declarada uma variável do tipo *Transaction.Create*, a qual é o tipo do objeto que será retornado quando se requisita a *Transaction Manager* para criar

uma nova transação. Para criar uma transação, usa-se a classe *TransactionFactory* que chama o método estático *create*, a qual leva à *Transaction Manager* e um tempo *(lease, em* milisegundos) como parâmetro, criando a transação que irá ser coordenada pelo gerente e que também cuidará do tempo dado (*lease)*. Se a chamada para *create* for bem sucedida, o objeto *Transaction.Create* é retornado e nomeado para a variável "trc". Se alguma coisa houver de errado durante a criação da transação, uma exceção será lançada no lugar.

4.11.6 Um Exemplo de Transação

Para demonstrar uma transação baseada em espaço é removida uma *entry* (entrada) de um *Javaspaces* e escrita em outro *Javaspaces* dentro de uma transação. Primeiro, define-se uma classe chamada *Message* que será usada para instanciar a entrada *(Entry)*, a qual será movida. Isso é o que vemos a seguir:

```
public class Message implements Entry {
  public String content;
  //a no arg-constructor
  public Message(){}
}

//o método que irá gravar uma entrada ( Entry Message //) em um
Javaspaces.

private void createMessage() {
 Message msg = new Message();
 msg.content = "test";
 try {
   sourceSpace.write(msg,null,Lease.FOREVER);
  } catch ( Exception e ) {
   System.err.println("Cant write message" + e );
  }
}
```

O próximo passo é escrever o método *relayMessage()* que removerá a mensagem do *space* fonte e escreverá no *space* destino dentro de uma transação. A primeira coisa a fazer neste método é chamar o método estático *getManager()* da classe *TransactionManagerAccesor*, como explicado anteriormente, e que irá retornar a *TransactionManager*. Depois são definidas duas variáveis da classe *Message*: uma para servir como um *space* origem e outra para ser o *space* destino. Esse processo está representado a seguir:

```
private void relayMessage() {
 TransactionManager mgr = TransactionManagerAccessor.getManager();
 Message template = new Message();
 Message msg = null;
 Transaction.Created trc = null;
try {
 trc = TransactionFactory.create( mgr, 300000 );
} catch ( Exception e ) {
 System.err.println("Could not create transaction" + e );
}
 Transaction txt = trc.transaction;
try {
 try {
  template.content = "test" ;
  msg = ( Message )sourceSpace.take(template,txn,Long.MAX_VALUE);
  targetSpace.write(msg,txn,Lease.FOREVER);
} catch ( Exception e ) {
  txt.abort();
  return;
}
  txn.commit();
} catch ( Exception e ) {
  System.err.println("Transaction Failed");
  return;
}
}
```

4.11.7 Tecnologia *Javaspaces* e Banco de Dados

Um serviço *Javaspaces* pode armazenar dados persistentemente, os quais poderão ser recuperados posteriormente. O *Javaspaces* é um serviço projetado para ajudar a resolver problemas em computação distribuída, não sendo primariamente usado como um repositório de dados, embora existam muitos dados usados para aplicações *Javaspaces*. Os modelos de Bancos de Dados mais difundidos são:

Banco de Dados Relacional: armazena e manipula os dados diretamente via linguagem *query.* As entradas no Javaspaces somente podem ser recuperadas pelo tipo e a forma de cada campo;

Banco de Dados OO: provê uma imagem orientada a objetos de armazenamento de dados, pode ser modificada e usada quase como se existisse memória provisória; *Javaspaces* não provê uma camada provisória, trabalham somente com cópias de entradas.

As diferenças entre esses repositórios existem porque o serviço *Javaspaces* foi projetado para um propósito diferente do que um banco de dados relacional ou orientado a objetos. Um serviço *Javaspaces* pode ser usado para armazenar dados persistentemente, tal como armazenar dados preferenciais dos usuários e recuperá-los mais tarde através do identificador do objeto usuário e de seus atributos [FRE 99].

4.11 Resumo do *c*apítulo

Este capítulo descreveu as tecnologias *Jini/Javaspaces* que servem como base para a implementação do modelo de cooperação proposto nesse trabalho. A existência de vários serviços ligados ao tratamento e comunicação entre objetos qualificaram essa tecnologia como pertinente para servir como infra-estrutura para o *Collaborative Service.*

Foram abordados aspectos ligados ao armazenamento de objetos através do serviço de persistência *(Javaspaces)* com sua forma de armazenamento, recuperação e notificação. Destacou-se também o tratamento das atualizações, baseadas em *update/notify*, protegidas em uma transação *(Transaction Service)*, além de mecanismos de comunicação entre objetos representados pelo protocolo RMI *(Remote Method Invocation)*, entre outros mecanismos voltados para aplicações distribuídas.

A modularidade requerida pelo serviço de cooperação pode ser aplicada dentro dessas tecnologias e representada por modelos orientado a objeto. A especificação do modelo criado para o *Collaborative Service* está descrita no capítulo 5 e visa abordar abstrações principalmente através de notações UML *(Unified Modeling Language)*.

5 Inserção de uma Abordagem Cooperativa para o Ambiente Cave

5.1 Motivação

Como visto na seção 1.1, algumas abordagens para o suporte ao trabalho cooperativo foram anteriormente apresentadas na área de EDA *(Eletronic Design Automation)* com [LAV 97, BRG 2001, KIR 2001]. Estes trabalhos focavam compartilhamento de IPs, treinamento remoto ou modelagem do fluxo de tarefas [IND 2002]. A abordagem apresentada neste trabalho cria um modelo de cooperação genérico o suficiente para suportar a cooperação sobre diferentes modelos de projetos. As seções 5.3 e 5.4. respectivamente descrevem a metodologia de cooperação e a infra-estrutura adotada para projetar a arquitetura cooperativa.

Para permitir a interação entre vários projetistas, mesmo quando separados geograficamente, o Ambiente Cave passou a trabalhar com metodologias de armazenamento baseadas em objetos ao invés de hiperdocumentos. Isso permite uma maior interação entre os projetos, acessos multiusuários, permissões, percepções e tratamento de erros.

A abordagem deste trabalho baseou-se em conversas informais e no *feedback* dado por empresas do ramo de projeto de CIs quando investigadas sobre seus interesses quanto à cooperação entre projetistas. Observou-se nestas pesquisas que a cooperação é desejada em níveis de abstração mais altos, nos quais utilizam-se representações gráficas que facilitam a troca de informações entre os projetistas. É com esse intuito que o projeto do módulo de cooperação tenta se imbuir. Pretende-se criar um modelo que seja flexível o bastante para se adaptar a qualquer tipo de ferramenta no Ambiente Cave.

É importante ressaltar que a idéia subjacente ao trabalho cooperativo é fazer com que a produção do grupo seja maior do que a produção de cada um dos membros individualmente. Esse ideal representa um aumento significativo de produtividade, o que vem sendo apontado como um importante fator para a sobrevivência das empresas no mercado. Dentro dessa realidade, viu-se a necessidade de incorporar ao *Framework* Cave esses conceitos. Assim, os projetistas que utilizarem esse ambiente terão a possibilidade de interagir com os demais participantes do projeto sem a necessidade de deslocamentos (no caso de uma equipe geograficamente distribuída).

Este capítulo descreve o modelo adotado para criar a estrutura de cooperação. Essa estrutura baseou-se em modelos UML *(Unifield Modeling Language)*, uma linguagem de modelagem unificada utilizada nas diversas fases do desenvolvimento de *softwares* e um padrão para modelagem envolvendo objetos. São abordadas também algumas estratégias de armazenamento e comunicação utilizadas no tratamento e criação

do módulo de cooperação, além da implementação de uma arquitetura cooperativa, chamada *Service Space.*

5.2 Serviço Cooperativo Baseado em Conceitos de Orientação a Objetos

Tal como visto no capítulo 2, a construção do *Framework* Cave baseia-se fortemente em conceitos de orientação a objetos [PRE 92, MEY 97, YOU 90, RUM 94, FUR 98, BOO 2000, BOO 2001]. Seguindo essa mesma linha de projeto, foi criado um serviço que engloba várias classes projetadas para dar suporte à cooperação. Esse conjunto de classes forma o módulo de cooperação *(Collaborative Service)* que é requisitado sempre que houver a necessidade de algum tipo de interação entre projetistas. Assim, o módulo é encapsulado (ocultação de informação) e reutilizado para todo o ambiente sempre que houver necessidade.

A inserção de trabalho cooperativo no Ambiente Cave atual (Cave2) não utiliza armazenamento de dados baseados em documentos. Como visto na seção 2.3, o foco do ambiente está na estruturação e armazenamento de dados baseado em objetos. Então, o resultado de um projeto, que anteriormente resultava em um documento, agora resulta em um objeto.

5.3 Armazenamento de objetos

Um dos pontos principais para se ter um bom grau de interação entre grupos de projetistas é o armazenamento dos dados. A estratégia de representação dos dados e a arquitetura do repositório de dados são pontos chaves para o desenvolvimento de um ambiente de projeto que trate também de cooperação. Nesse sentido, foram levantadas algumas possibilidades de armazenamento de objetos.

Como descrito na seção 2.3.1, o RDBMS (*Relational Database Management System*) foi criado por Codd [COD 70]; trata-se de modelos que podem ser apropriados para inúmeras aplicações, mas não são considerados apropriados para a modelagem de dados manipulados por ferramentas de CAD, principalmente por causa do uso de tabelas [YOR 90]. Utilizar tabelas para modelar os dados em uma ferramenta, como, por exemplo, um editor de esquemáticos, é inviável devido à complexidade dos dados manipulados por essa ferramenta.

Podem ser encontrados também alguns trabalhos que consideram esse modelo não apropriado para suportar colaboração, principalmente devido à sua modelagem e estratégias de acesso aos dados, e também seu armazenamento [IND 2002].

Os OODBMS (*Object-Oriented Database Management System*), vistos na seção 2.3.2, têm seu foco em aplicações comerciais e outros são resultados de pesquisas. Esses sistemas podem ser divididos quanto à estratégia de modelagem e quanto ao acesso a dados.

Analisando-os quanto à modelagem dos dados, ambas estratégias seguem os conceitos do paradigma orientado a objetos, que oferecem uma semântica rica para modelar, como, por exemplo, tipos de dados complexos. Esses modelos de dados são comuns tanto em ferramentas de síntese, como em ferramentas de edição de esquemáticos. Porém, alguns OODBMS requerem características especiais para os objetos que são armazenados no repositório, tais como uso específico de superclasses e declaração explícita de métodos que alteram o estado dos objetos [MUE 2000]. Em muitos desses casos, a implementação destas características são simples, mas pode haver alguma restrição na modelagem, principalmente quando se utilizam linguagens que não suportam herança múltipla.

Alguns testes foram realizados usando OODBMS para implementação de um protótipo. Como estudo de caso, utilizou-se o banco de dados Ozone [MUE 2000], um *software* livre. Ozone trabalha com instâncias únicas de objetos. Isto significa que as aplicações não trabalham com cópias locais dos dados armazenados no repositório, mas com referências para os objetos armazenados. Quando um dado é requerido por uma aplicação, um método remoto do objeto é chamado e o dado é retornado.

Outra base de persistência pesquisada utiliza o conceito de espaço compartilhado de objetos, que foi introduzido por *Gelernter* [GEL 85] em 1980 e foi recentemente estendido pelo grupo *Jini* da *Sun Microsystems* [EDW 99, FRE 99 e MOR 2000]. Essa tecnologia provê um serviço de persistência que trabalha sem a complexidade de RDBMS ou OODBMS, utilizando vários protocolos e serviços que viabilizam a construção de uma arquitetura cooperativa.

Atualmente, a implementação do serviço de cooperação, que é apresentada neste trabalho, utiliza como infra-estrutura principal a tecnologia *Jini/Javaspaces* [SUN 99a, SUN 99b, EDW 99, FRE 99]. Essa tecnologia implementa um serviço de armazenamento com poucas restrições, trabalha com mecanismos de comunicação entre objetos, transações, e possibilita conceitos de orientação a objetos, utiliza a linguagem Java e suporta o armazenamento de objetos Java.

Todas essas características referidas qualificaram a tecnologia *Jini* como base para implementação de uma infra-estrutura cooperativa no Ambiente Cave. Uma abordagem mais detalhada sobre essa tecnologia foi procedida no capítulo 4.

5.4 Metodologia de Cooperação *(Pair-Programming- PP)*

Como visto na seção 3.8 do capítulo 2, *Pair-Programming* é um estilo de programação em que dois programadores trabalham lado a lado em um computador, continuamente colaborando no mesmo projeto, algoritmo, código ou teste. O uso dessa prática demonstra um aumento de produtividade e qualidade nos programas. Adicionalmente, os programadores, ao utilizarem essa metodologia, concordam que se adquire mais confiança na solução de problemas, pois trabalham em pares ao invés de

sozinhos. No entanto, como já mencionado, muitos programadores estão condicionados ao trabalho individual e geralmente oferecem resistência a essa transição [WIL99, WIL2000].

A caracterização da metodologia *Pair-Programming* no módulo de cooperação pode ser demonstrada através de cores que simbolizam as permissões. Como *Pair-Programming* não foi projetada para a execução em máquinas remotas, algumas características foram acrescentadas. Por exemplo, na sua origem *Pair-Programming* é executada localmente, alternando a permissão de escrita através da substituição do projetista que está no teclado por outro. A extensão do *Collaborative Service* trabalha com requisições remotas e passagem da permissão através da rede, não trabalhando com pares, mas sim com equipes de projeto. A modificação no conceito original da metodologia para uma execução em máquinas remotas passou a se chamar *paar-programming*. A palavra "paar" é originária da língua alemã e tem o mesmo significado que *pair* (inglês), ou seja, par ou dupla.

A comunicação original de *Pair-Programming* [WIL 2000] é realizada verbalmente. No módulo de cooperação para a comunicação é feita através de um *chat* de texto. O modelo desenvolvido para o *chat* está descrito com mais detalhes na seção 5.5.3. Além dessas características, o projetista pode armazenar seu projeto em um repositório de dados e recuperá-lo sempre que for preciso. Pode também optar pelo trabalho em equipe ou por um trabalho individual, bastando apenas não acrescentar participantes no projeto.

5.5 Arquitetura do serviço de cooperação

A arquitetura do módulo de cooperação baseou-se na tecnologia *Jini/Javaspaces*. Com um modelo de dados baseado em *Tupla-Spaces* [FRE 99, GEL 85, EDW 99], um conjunto de classes foi desenvolvido para tratar tanto do armazenamento dos projetos, quanto de seu acesso compartilhado. Essas classes foram implementadas seguindo-se o modelo proposto neste capítulo e descrito pela notação UML.

O entendimento de *modelo*, nesse caso, é muito importante. Afinal, *modelo* é uma abstração de alguma coisa, cujo propósito é permitir que se conheça essa coisa antes de construí-la, e a abstração é uma capacidade fundamental humana que permite lidar com coisas complexas. Nesse caso, deve-se abstrair diferentes visões do sistema, construir modelos com a utilização de alguma notação, verificando se os modelos satisfazem os requisitos para depois transformá-los em uma implementação.

Nesse sentido, conforme descrito no capítulo 4, a tecnologia *Jini* implementa vários serviços. O armazenamento de objetos é realizado através do serviço de persistência *(Javaspaces)*. Esse serviço permite interagir com o serviço de transações e as suas requisições são realizadas de forma transparente sempre que houver a necessidade de um trabalho em grupo. Essa interação é ilustrada na figura 5.1 a seguir e serve como base para a construção do módulo de cooperação.

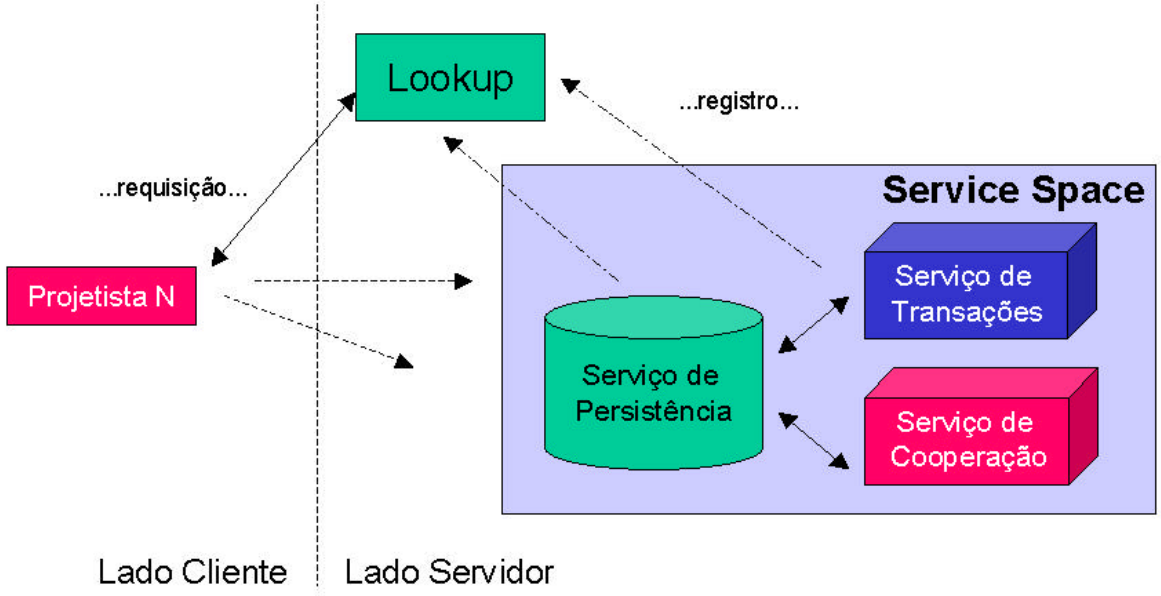


FIGURA 5. 1 - Interação entre os serviços e o projetista

5.5.1 TLS (Tabela de Localização de Serviços)

Um ponto importante dessa estrutura é o *Service Lookup* – que chamamos nessa arquitetura de TLS (Tabela de Localização de Serviços). Trata-se de uma tabela que armazena a localização dos serviços que vão ser utilizados pelo processo de cooperação. Pode haver várias TLS's espalhadas entre várias máquinas, com as referências da localização de vários servidores e serviços. A figura 5.2 ilustra uma TLS com três serviços registrados.

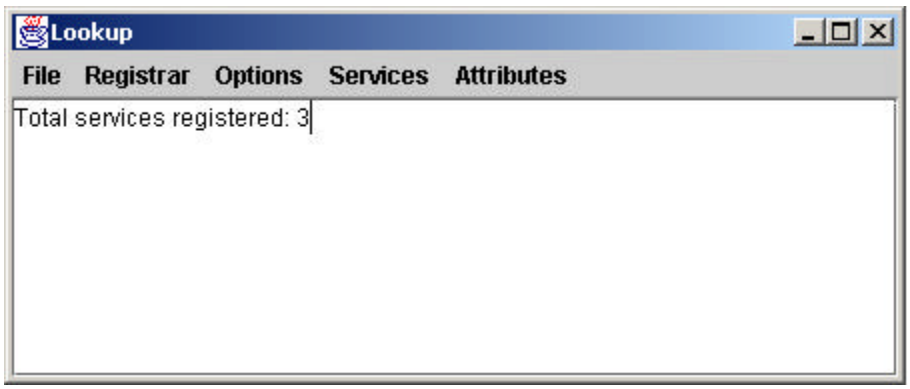


FIGURA 5. 2 - Tabela de localização de serviços

A localização de uma TLS é realizada através de uma conexão *unicast* até a máquina que hospeda esse serviço – conexões *multicast e unicast* estão descritas com mais detalhes nas seções 4.4 e 4.5. Pode haver mais de um servidor de projetos cadastrados em uma TLS. Com isso, o projetista pode escolher em qual repositório deseja armazenar seus projetos. O processo de busca e localização é simples; o projetista

abre uma conexão *unicast* até a TLS, que retorna os serviços disponíveis. Depois de fechada a conexão, o projetista se desvincula da TLS e trabalha diretamente com os serviços requisitados. A figura 5.2 ilustra uma TLS com três serviços registrados.

5.5.2 Modelagem da estrutura de classes usando UML

Como visto na seção 5.2, a modelagem da estrutura do projeto Cave está totalmente incorporada à metodologia baseada em objetos. Nesse contexto, o módulo de cooperação foi todo projetado a partir do uso da UML *(Unified Modeling Language)* [FUR 98, TEX 97, BOO 2001, LAR 2000].

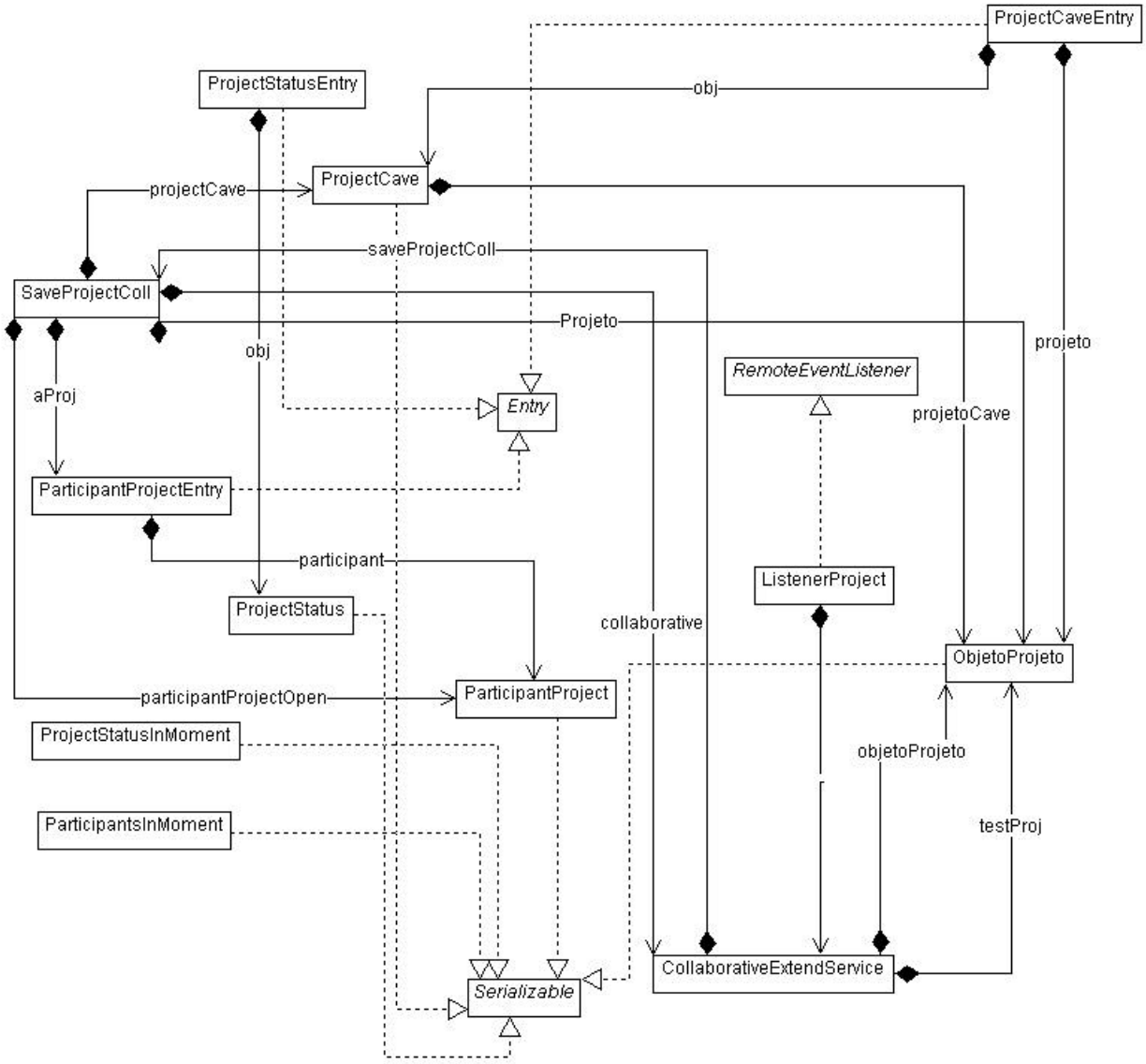


FIGURA 5. 3 - Diagrama de classes: armazenamento de projetos e participantes

O diagrama de classe ilustrado na figura 5.3 representa o modelo estrutural de dados implementado e aplicado no desenvolvimento do *Collaborative Service*, basicamente na parte que trata de projetos e participantes. Essa modelagem mostra o conjunto de associações, agregações e generalizações inseridas no modelo de classes com o objetivo de representar graficamente toda a interação do modelo. São abordados com mais detalhes respectivamente nas sessões 5.5.4, 5.5.5 o armazenamento dos projetos e o armazenamento dos participantes, todos em representações simplificadas desse modelo.

5.5.3 Modelo de Classes da Estrutura do *Chat*

A estrutura de classes que representa o *chat* tem como um de seus objetivos o reuso. Com isso, o módulo de *chat* pode ser reutilizado em qualquer ponto do *framework* e não somente nas sessões cooperativas. O módulo de *chat* compõe o pacote *collaborative* e seu modelo de dados é representado pelo diagrama de classes ilustrado na figura 5.4. A interface de comunicação para o reuso desse recurso é realizada através da criação de uma instância da classe *ChatFrame* e a indicação de quatro parâmetros que são: a referência do repositório, o nome do projeto, o nome do bloco de projeto e o nome do projetista que irá entrar na seção de *chat*.

A partir da entrada desses parâmetros, a interface gráfica é montada reutilizando alguns métodos herdados pela classe *Frame*. A classe *ChatClient*, além de implementar a interface *WindowListener*, responsável pelos eventos da janela gráfica, é formada também pela composição da classe *ChatFrame* a qual armazena todos os componentes gráficos da tela.

A classe *ChatMessage* é uma classe serializada que armazena os dados passados como parâmetro na criação da instância da classe *ChatFrame*. O único atributo adicional é a mensagem, que vai ser armazenada e propagada aos demais participantes da sessão de *chat*.

A classe *ChatMessageEntry* é a classe que vai ser armazenada, acessada e atualizada no repositório. Incorpora a classe *ChatMessage* e é necessária, pois, para uma classe ser armazenada no repositório precisa, obrigatoriamente, implementar a interface *Entry*. A interface *Entry* é descrita com mais detalhes na seção 5.10.2.

Pode-se perceber que esse modelo pode ser aplicado em qualquer ponto do *framework*. Sua aplicação depende somente de decisões de projeto. Essas decisões são discutidas pelo grupo Cave do GME (Grupo de Microeletrônica da UFRGS) durante o andamento dos trabalhos com o intuito de tornar o *framework* mais robusto e rico em novos recursos.

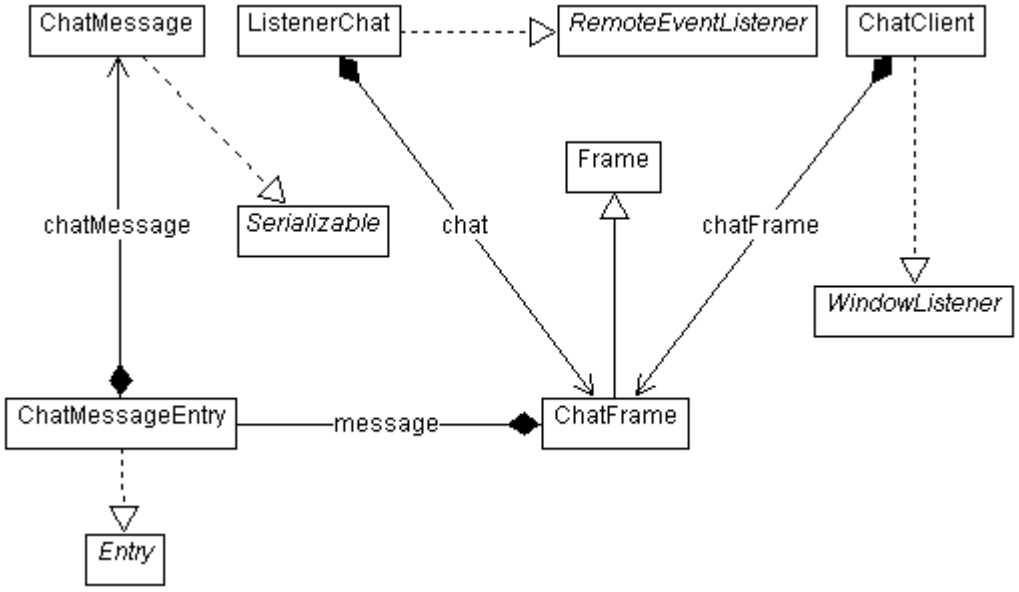


FIGURA 5. 4 - Diagrama de classes: *Chat*

5.5.4 Armazenamento dos projetos

A sessão 5.4.2. mostrou a estrutura de classes que tratam do armazenamento do projeto e seus participantes. Com esse cenário, a criação de uma sessão cooperativa implica a criação dos blocos de projeto e a inserção dos seus participantes. Com isso, a estrutura de classes do projeto é serializada e armazenada no repositório. É importante ressaltar que não basta somente serializar o objeto e armazená-lo no repositório. Para que o objeto possa ser armazenado no repositório, ele precisa implementar a interface *net.jini.core.entry.Entry.*

A figura 5.5 ilustra as classes que compõem a base para o armazenamento dos projetos. A classe *ObjetoProjeto* engloba o projeto a ser armazenado, que pode ser um esquemático se o projetista estiver trabalhando com o *Blade* [SAW 2002a, SAW 2002b], ou um texto se a ferramenta for o Homero [SAW 2001, SAW 2001a]. Um ponto importante a destacar é a possibilidade de armazenar qualquer tipo de objeto Java, independente do tipo de saída das ferramentas.

A classe *ProjectCave* armazena informações pertinentes ao projeto, como o nome do responsável pelo projeto, os blocos envolvidos e quais são os projetistas que interagirão com o projeto. A classe *ProjectCaveEntry* implementa a interface *Entry* e engloba a classe *ProjectCave*. Então, no repositório, são armazenadas várias instâncias da classe *ProjectCaveEntry*, que são os blocos de um projeto.

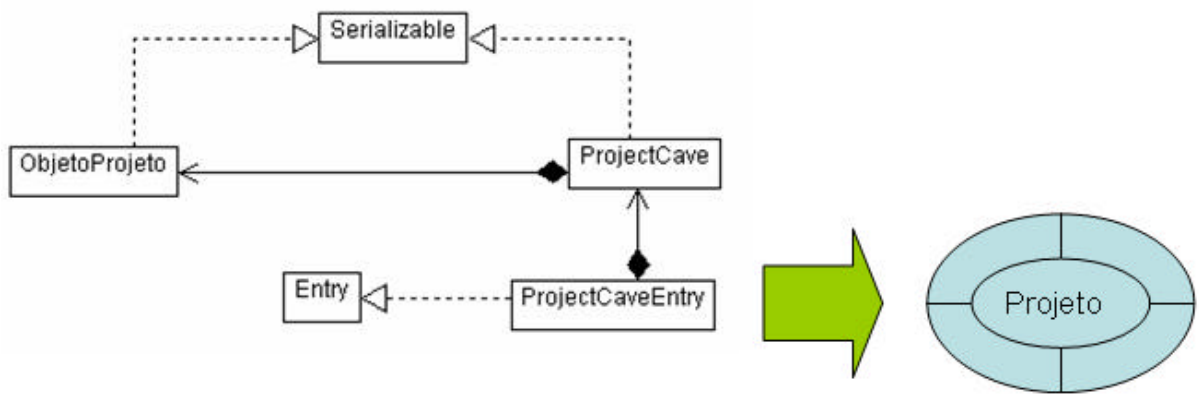


FIGURA 5. 5 - Classes do projeto a serem armazenadas

5.5.5 Armazenamento dos participantes

Cada bloco de projeto inserido no repositório cria um novo objeto com informações de participantes *on-line/off-line*, mantendo visível suas permissões de escrita ou leitura. A figura 5.6 mostra as classes que compõem as informações relativas aos participantes dos blocos de projeto.

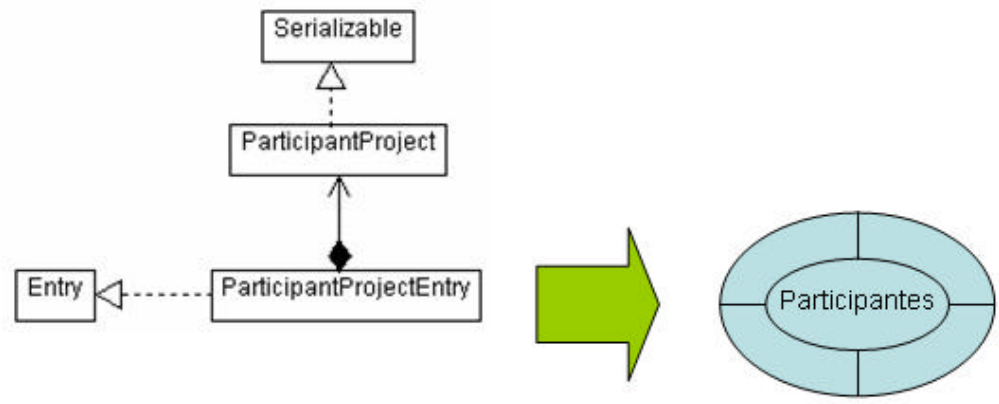


FIGURA 5. 6 - Classes de participantes a serem armazenadas

Na classe *ParticipantProject*, estão modelados os atributos referentes ao estado do projetista participante. Esses podem ser:

?? *on-line – write permission*
?? *on-line – read permission*
?? *off-line.*

A opção *on-line – write permission* permite que o projetista altere o bloco de projeto, com atualizações periódicas ao repositório. A opção *on-line – read permission* torna o projetista um participante sem interação direta na edição do projeto; sua interação é através de sugestões que são propagadas através de um *chat* aos demais participantes da

sessão cooperativa. O *chat* foi implementado utilizando as tecnologias *Jini/Javaspaces* e está descrito com mais detalhes na seção abaixo.

5.5.6 Armazenando mensagens do *chat*

A interação entre projetistas é realizada com troca de idéias através de um *chat* de texto. Pesquisas estão sendo feitas para ampliar esse *chat* de texto a fim de suportar comunicações em áudio/vídeo através do JMF *(Java Media Framework)* [SUN 2002].

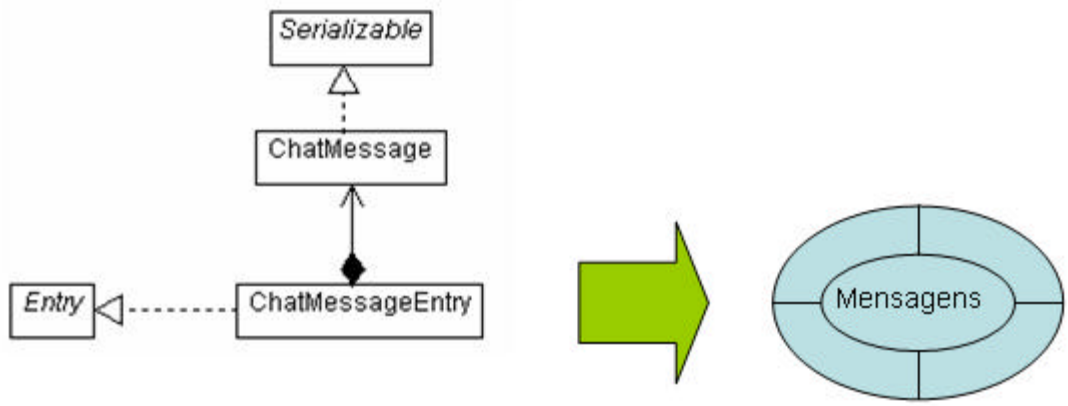


FIGURA 5. 7 - Estrutura de classes do *chat*

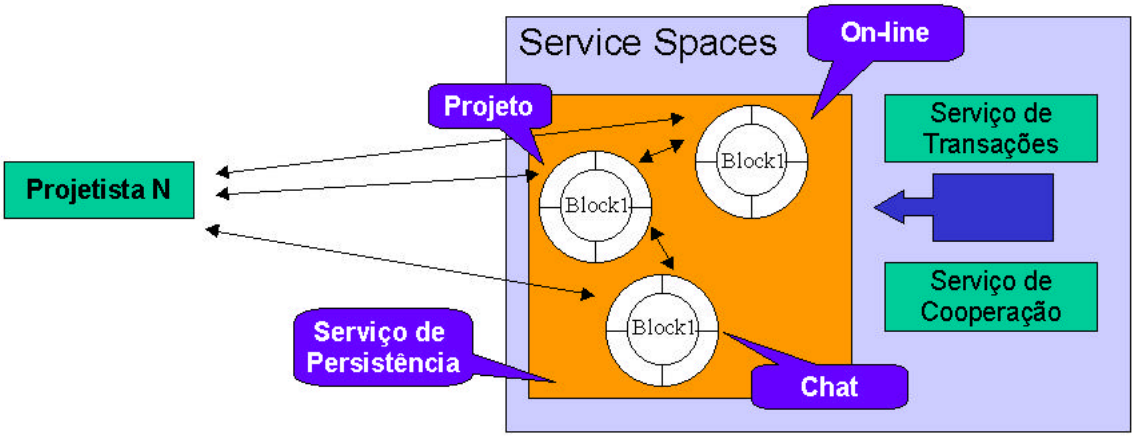
A figura 5.7 simplifica o modelo de classes associadas ao *chat* de texto. A seção 5.4.3 descreve esse recurso com mais detalhes. É importante destacar que o que vai ser armazenado, na realidade, são instâncias da classe *ChatMessageEntry*. Essa classe implementa a interface *Entry* e encapsula a classe *ChatMessage*, responsável pelo armazenamento dos atributos relacionados com a recuperação e notificação dos participantes.

5.5.7 Comunicação entre objetos (projeto, participantes e *chat*)

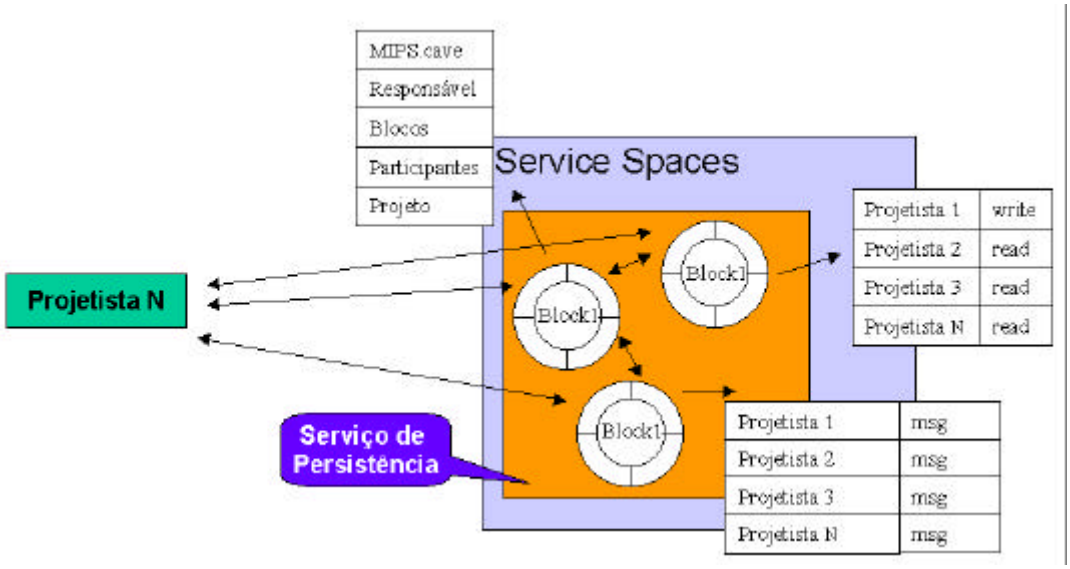
Outro ponto importante dessa arquitetura é a relação de objetos com um bloco de projeto. Sendo assim, a criação de um novo projeto cooperativo resulta na criação e armazenamento de três objetos para cada bloco: um objeto que armazena o bloco de projeto *(ProjectCaveEntry)*; outro, os participantes *on-line* do bloco *(ParticipantProjectEntry)*; e um objeto que armazena as mensagens transmitidas pelo *chat* de texto *(ChatMessageEntry)*.

A figura 5.8 ilustra o armazenamento dos três objetos no serviço de persistência e a interação entre eles. Esses três objetos compõem um bloco de projeto. Com isso, a interação se dá sobre cada um deles independentemente. Isso diminui a carga de atualizações no caso de centralizar todos os dados em somente um objeto.

O mecanismo de cooperação que mostra a interação entre os projetistas e os blocos de projeto está descrito com mais detalhes na seção 5.5.8. A recuperação desses objetos é representada através dos atributos de entrada de uma classe que implementa a interface *Entry* que se igualam com os atributos da classe armazenada no repositório.



(a)



(b)

FIGURA 5. 8 - Objetos referentes a um bloco de projeto

A figura 5.8 (a), identifica os três objetos armazenados no serviço de persistência que representam um bloco de projeto: *projeto, projetistas on-line e mensagens do chat*. Esses objetos interagem diretamente com o serviço de cooperação e o serviço de transações. A recuperação e notificação dos dados de projeto são realizadas através de combinações de atributos, que servem como uma espécie de chave de

recuperação. Em outras palavras, se o projetista entrar com atributos que se igualem aos atributos declarados no objeto armazenado, o mesmo é recuperado. Alguns atributos de cada um dos objetos são ilustrados na figura 5.8 (b).

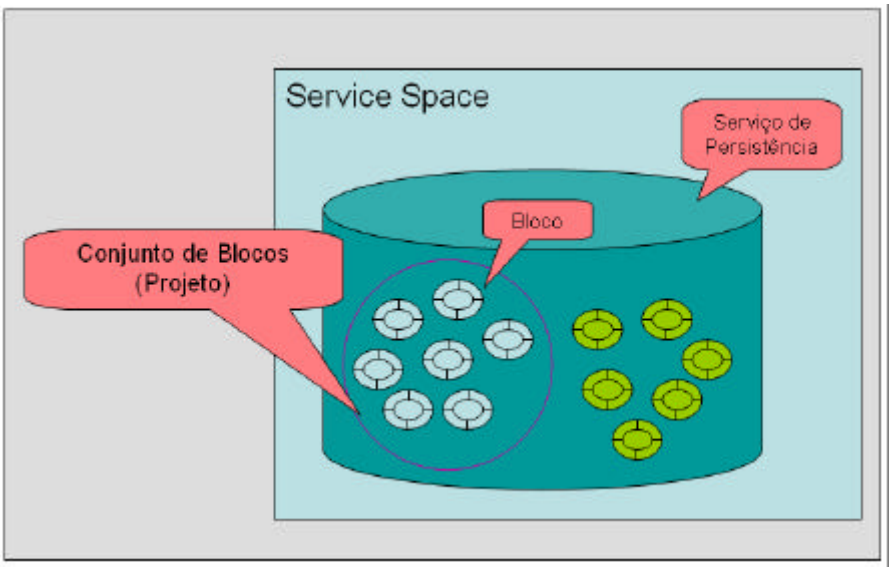


FIGURA 5. 9 - Blocos de projeto

A figura 5.9 mostra os blocos de projeto armazenados no serviço de persistência. Todos esses blocos trabalham em conjunto, pois cada um é parte integrante do projeto. Por exemplo, o projeto de um microprocessador pode ser dividido em vários blocos, tais como: ALU, Memória, Entradas e Saídas, etc. Cada um desses blocos podem ser sessões cooperativas. Depois do término da construção de todos os blocos, os mesmos podem ser carregados nas ferramentas para fins de simulação e/ou execução.

5.5.8 Atualizando projetos

As atualizações são realizadas pelo projetista que detém a permissão de escrita. Neste momento, o projeto permite que as atualizações sejam realizadas de forma assíncrona, mas pesquisas estão sendo feitas com o intuito de englobar cooperação síncrona para a infra-estrutura do *framework*. Esse modelo é baseado em atualizações e notificações em intervalos de tempo. Assim, sempre que um bloco de projeto é atualizado no repositório, os demais projetistas envolvidos recebem uma notificação de alteração e buscam atualizar a visualização do bloco de projeto em sua ferramenta.

Essa atualização é realizada dentro de uma transação, garantindo assim a atomicidade, consistência, isolamento e durabilidade do projeto, como detalhado na seção 4.11. A figura 5.10 mostra um projetista atualizando um bloco de projeto dentro de uma transação através das primitivas de atualização *take ()* e *write ()*.

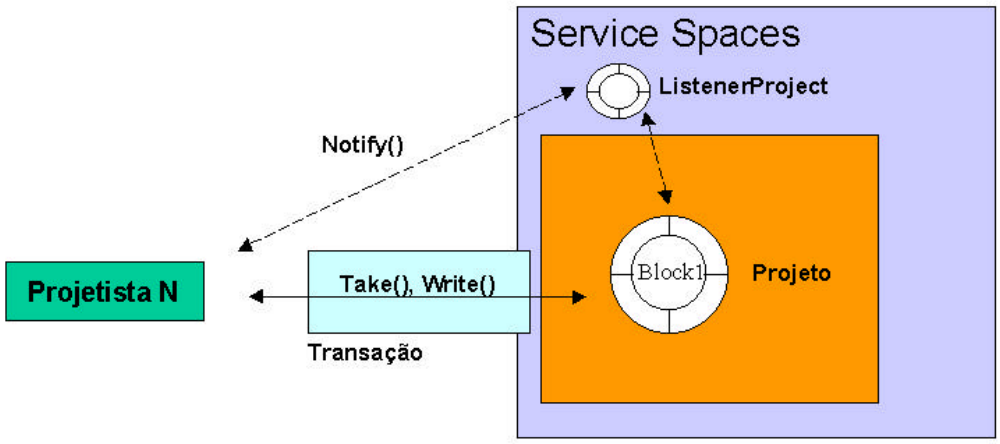


FIGURA 5. 10 - Atualizando o projeto

Existem alguns pontos importantes a destacar nessa arquitetura, como o *Event Source* e o *Event Listener*. O *Event Source* é qualquer objeto que dispara um evento. Eventos podem ser disparados por qualquer mudança interna do objeto. *Event Listener* é um objeto que recebe (escuta) eventos disparados pelo *Event Source.*

Quando usamos eventos distribuídos baseados em *space,* como é o caso dessa arquitetura, o *space* é um *Event Source* que dispara um evento remoto (objeto) para o *Event Listener* remoto sempre que alguma alteração acontecer no repositório. O *Event Listener* remoto, dispara o método de notificação para todos os projetistas envolvidos com o mesmo bloco de projeto. A figura 5.10 ilustra o *Listener Event* remoto chamado de *ListenerProject*, que dispara o método de notificação sempre que alguma alteração ocorrer dentro do repositório e que se iguale com o modelo de busca imposto pelos projetistas envolvidos no mesmo bloco de projeto. Esse *listener* tem a função de notificar o grupo de projetistas relacionados com o bloco de projeto alterado e disparar somente métodos de atualização do projeto.

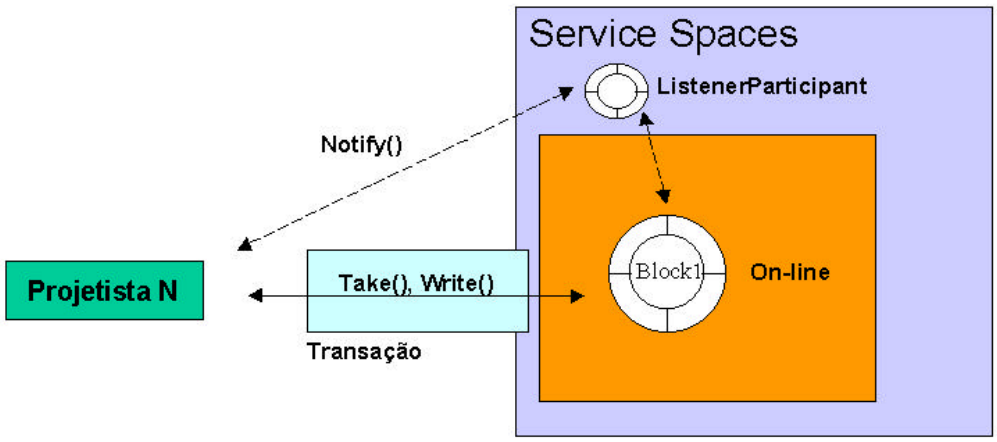


FIGURA 5. 11 - Atualizando os projetistas participantes

Existe outro *listener* que participa das atualizações de projeto. Por exemplo, a figura 5.11 ilustra a atualização de um objeto que armazena os projetistas envolvidos no mesmo momento em um bloco de projeto, mostrando suas permissões. Essa atualização também é realizada dentro de uma transação para garantir todas as propriedades ACID descritas anteriormente. Métodos que servirão como atualização dos objetos são disparados dentro de uma transação. A função deles é de remover o objeto, atualizá-lo e gravá-lo novamente no repositório, disparando um evento que será reconhecido pelo *ListenerParticipant*. Com isso, esse *listener* executa seu método de notificação e mostra para todos os projetistas que estão trabalhando no bloco de projeto nesse exato momento (percepção ou *awareness*[2]).

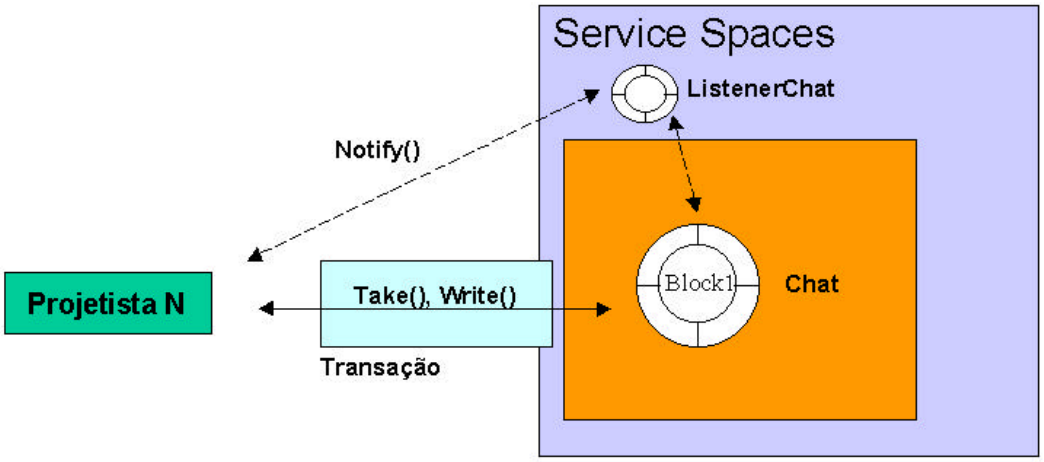


FIGURA 5. 12 - Atualizando as mensagens do *chat*

Um último objeto envolvido nas atualizações do projeto pode ser visualizado na figura 5.12, chamado de *ListenerChat*. Com visto na seção 5.5.3, a interação entre os projetistas é feita através de troca de idéias expressadas através de um *chat* de texto entre todos os projetistas envolvidos no projeto. Com isso, cada bloco de projeto tem um objeto responsável pela propagação das mensagens entre os projetistas. Esse objeto é atualizado sempre que um projetista enviar uma nova mensagem para os participantes de um bloco de projeto. O *ListenerChat* é executado quando houver uma modificação no objeto responsável pelo *chat* de um determinado bloco, executando seu método de notificação para todos os projetistas envolvidos nesse bloco.

Depois das notificações, o lado cliente busca o objeto atualizado no repositório e o materializa em sua tela. A tela da ferramenta cliente expressa várias informações, por exemplo, quais são os participantes do projeto, quais dos participantes estão *on-line* e suas permissões, qual projetista detém o direito de escrita, além de um *chat* ativo entre os projetistas *on-line*.

[2] *Awareness* é o conhecimento geral sobre as atividades e sobre o grupo.

As notificações são disparadas através da rede para todos os participantes *on-line* que tiverem a entrada (chave) igual a entrada (chave) do objeto armazenado no repositório.

5.5.9 Transmitindo *token* de escrita

A arquitetura proposta utiliza como metodologia de cooperação uma extensão de *Pair Programming*, visto na seção 5.4. A abordagem dessa metodologia permite que somente um indivíduo altere o projeto. Os demais podem contribuir com idéias ou requisitar a permissão de escrita ao projetista que a detém.

A ação de requisição acontece no objeto que obtém a informação sobre o estado de cada projetista. Com isso, o projetista que requisita a permissão de escrita acessa o objeto e emite um sinal *(token)* através da rede, o que indicará ao projetista escritor quem o está requisitando. Em caso de muitos projetistas optarem pela requisição, vários sinais serão emitidos e materializados na tela de quem detém a permissão de escrita. Com isso, o projetista escritor pode escolher quem será o novo escritor do projeto. Isso acontece também com a passagem de um sinal ao objeto que informa os estados do projetista.

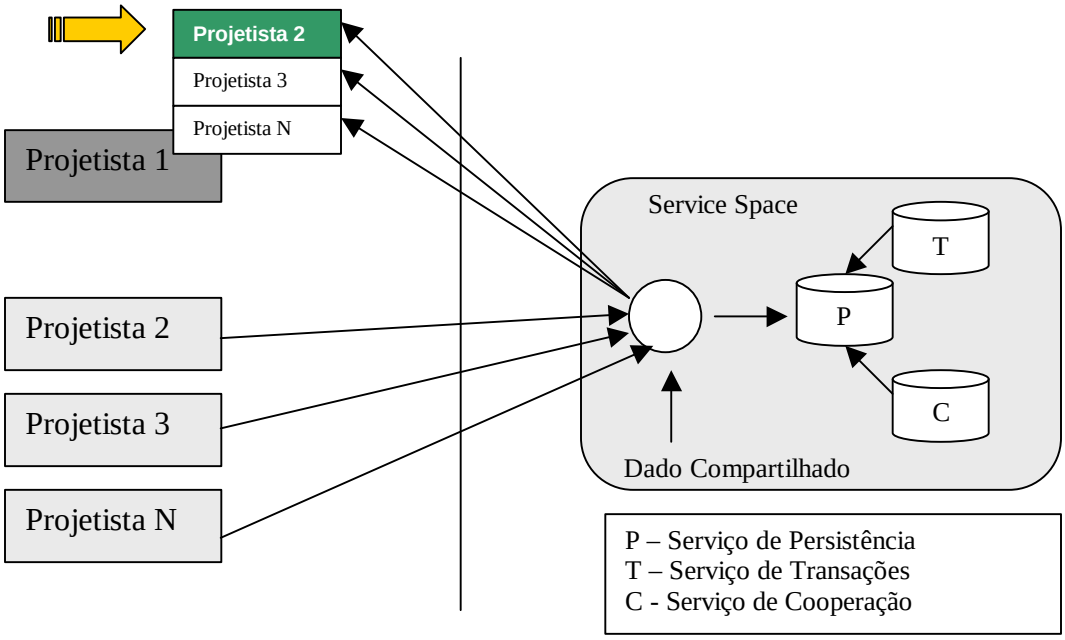


Figura 5. 13 – Requisitando permissão de escrita

As figuras 5.13 e 5.14, respectivamente, representam a requisição de permissão e a autorização da permissão. Na figura 5.13, três participantes requisitam a permissão de escrita ao projetista "escritor" (representada pela lista de requisições). Nesse exemplo, o projetista 2 foi o escolhido para assumir as alterações do projeto. A transmissão da permissão e demais notificações podem ser visualizadas na figura 5.14.

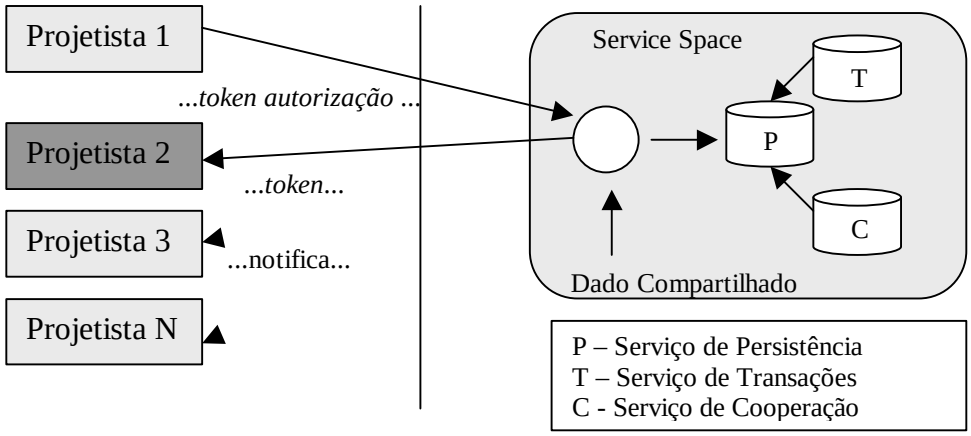


Figura 5. 14 – Transmitindo permissão de escrita

5.6 Casos de uso envolvendo *Collaborative Service*

Os casos de uso são conjuntos de cenários unidos por um objetivo em comum. Com isso, é possível criar alguns cenários envolvendo o *Collaborative Service*, entendendo *cenário* como uma seqüência de passos que descrevem uma interação entre o usuário e o sistema.

É possível verificar a situação de um projetista de circuitos integrados que utiliza o módulo de cooperação para elaborar seu projeto. Por exemplo, todos são projetistas, ou seja, tem características que são pertinentes a um projetista, porém, podem assumir funções diferentes de acordo com cada uma das situações.

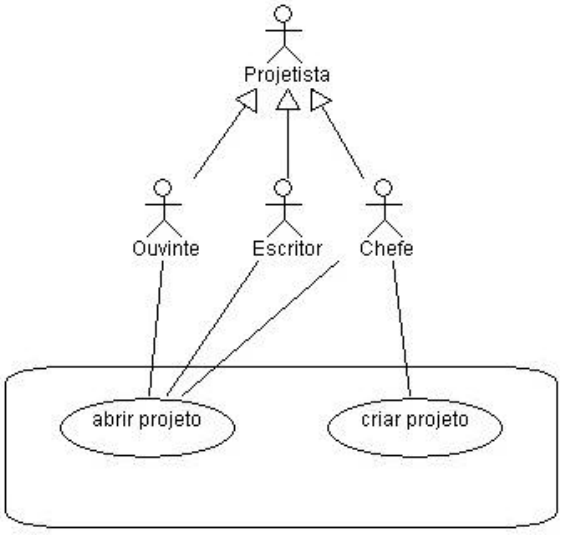


FIGURA 5. 15 – Caso de uso envolvendo projetistas

O projetista pode ser um ouvinte (participante), ou um escritor (pode alterar o projeto). Um coordenador ou chefe do projeto que pode assumir a posição de ouvinte ou escritor. A diferença é que somente um chefe (coordenador) pode criar um novo projeto e/ou adicionar novos participantes. Essa situação pode ser visualizada através do caso de uso da figura 5.15.

Dá-se uma outra situação pela conexão da ferramenta cliente com o módulo de cooperação. Essa opção é de extrema importância, pois é através da requisição desse serviço que o módulo de cooperação começa a atuar. Esse caso de uso passa por nós intermediários, até efetuar a conexão completa com o projetista. Em outras palavras, a comunicação é realizada até a tabela de locação de serviços (TLS), a qual disponibiliza a referência ao *Collaborative Service*. Esse caso de uso está representado na figura 5.16 a seguir:

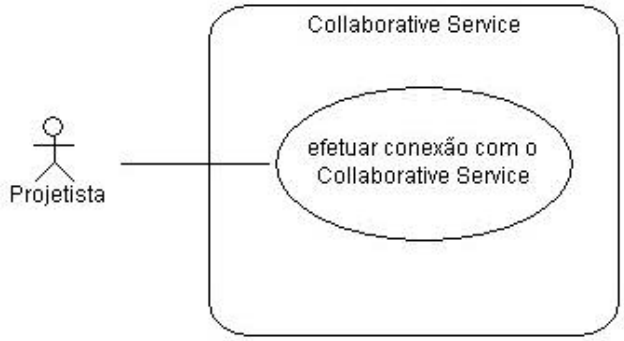


FIGURA 5. 16 - Caso de uso da requisição do *Collaborative Service*

Após a conexão com o *Collaborative Service*, o projetista pode criar um novo projeto, inserindo dados referentes aos blocos de projeto, participantes do projeto, entre outros, e o armazenando no repositório para que outros projetistas possam recuperá-lo. O projetista pode também abrir um projeto já existente. Para que isso aconteça, ele tem que estar listado como participante do projeto e saber a senha do projeto para poder carregá-lo em sua ferramenta. O caso de uso da figura 5.17 ilustra essa situação:

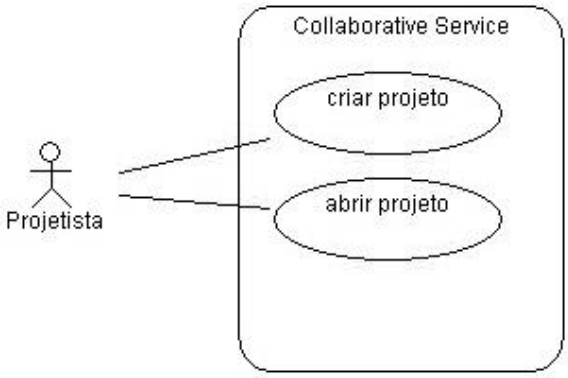


FIGURA 5. 17 - Caso de Uso: cenário após a conexão

O próximo cenário, ilustrado na figura 5.18, representa a situação de um projetista após a criação de um novo projeto ou após a abertura de um projeto já existente. Este cenário habilita as funções que representam a interação do projetista com o serviço de cooperação e, conseqüentemente, com os demais participantes do grupo de projeto. Funções do tipo atualizar projeto, sair da sessão, requisitar permissão, enviar permissão, chamar o *chat*, executar a ajuda *on-line* compõem o cenário do projetista que utiliza o módulo de cooperação para editar seus projetos.

Nesta seção foram abordados alguns casos de uso envolvendo o *Collaborative Service* com projetistas, utilizando-se da notação UML. Neste capítulo, serão explorados outros diagramas dessa mesma notação, com o intuito de apresentar e especificar com mais detalhes o modelo proposto para o serviço de cooperação.

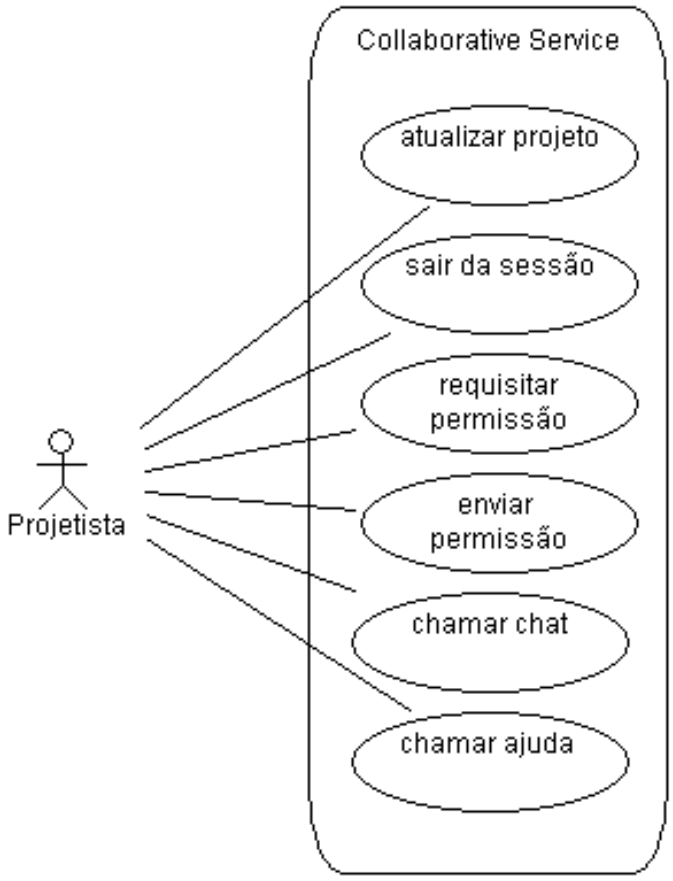


FIGURA 5. 18 - Caso de uso: cenário após abertura ou criação de um projeto

Neste contexto, é importante lembrar que casos de uso representam uma visão *externa* do sistema. Como tal, não têm relação alguma com as classes do sistema. Como descritos anteriormente, casos de uso são somente conjuntos de cenários relacionados por um objetivo comum do usuário.

5.7 Diagramas de estado envolvendo o *Collaborative Service*

Essa sessão procura representar, através de diagramas de estado, o comportamento do *Collaborative Service*. A função desse diagrama é descrever todos os estados possíveis que um objeto particular pode ter e mostrar como o estado do objeto muda com o resultado de eventos que o atingem.

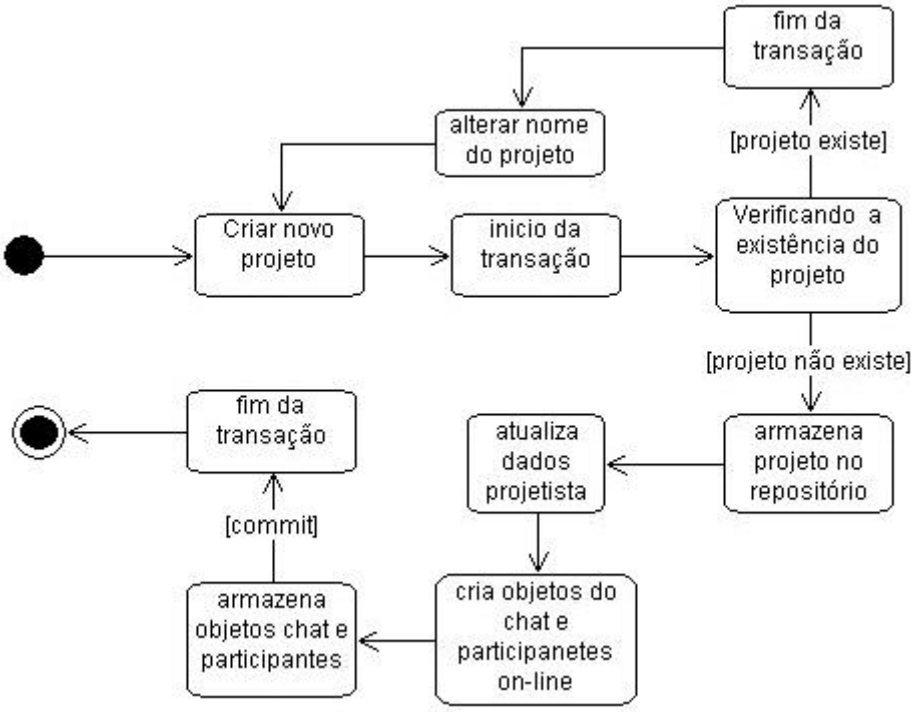


FIGURA 5. 19 - Estados da criação de um novo projeto

O digrama da figura 5.19 mostra os estados possíveis da fase de criação de um projeto utilizando o módulo de cooperação. Esses estados são apresentados após a conexão completa da ferramenta com o *Collaborative Service*.

O estado de criação de projeto é a fase na qual o projetista insere os dados que servirão como referência para o armazenamento e recuperação dos projetos no repositório. Após a inserção desses dados, abre-se uma transação que transmite os dados para a criação da estrutura que será armazenada no serviço de persistência. Porém, antes que os dados sejam armazenados de fato, é necessária uma verificação da existência ou não do nome do projeto. Caso exista, a transação é abortada e o estado de criação do projeto é executado novamente. Caso os dados não existam no repositório, a estrutura do projeto criada é armazenada na base de dados. Essa estrutura consiste em objetos referentes aos dados do projeto, dados dos projetistas *on-line* e dados relacionados com o *chat*.

É importante ressaltar que a criação de um novo projeto pode ser realizada somente pelo projetista chefe (coordenador). Esse projetista tem a permissão de criar projetos, inserir e excluir participantes.

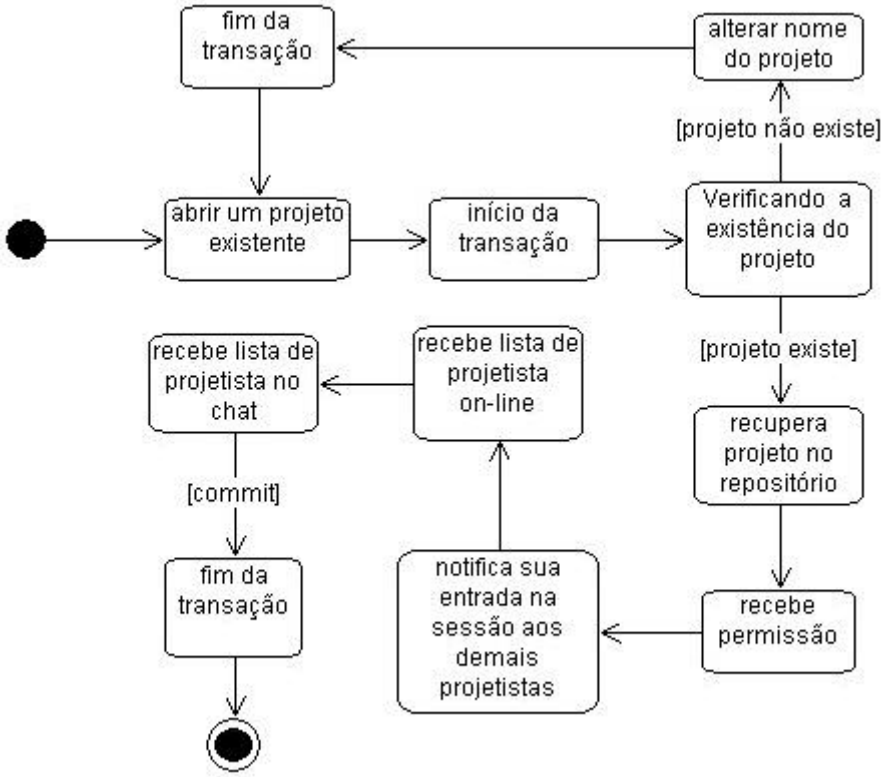


FIGURA 5. 20 - Estados da recuperação de um projeto

A figura 5.20 mostra um digrama de estados que representa a abertura de um projeto existente no repositório de dados. O primeiro estado inicia com a entrada de dados referentes ao projeto que será recuperado da base de dados. Esses dados representam, por exemplo, o nome do projeto ou nome dos blocos de projeto do qual o projetista faz parte e quer recuperar.

O próximo estado inicia com a criação de uma transação que transmite os dados através da rede para o repositório. Os dados transmitidos consultam a base de dados para verificar a existência ou não do projeto. Caso exista, a transação é abortada e retorna-se ao estado de entrada de dados.

No caso do projeto ser encontrado no repositório, o projetista recebe a permissão (escrita ou leitura). Conseqüentemente, os demais projetistas são notificados que mais um integrante do grupo está conectado ao projeto (percepção). Com isso, o projetista recebe a lista dos projetistas que estão *on-line* no momento, além de entrar na sessão de *chat* destinada ao projeto.

É de suma importância destacar que todos esses estados são executados dentro de uma transação, o que garante a execução *(commit)*, ou não execução *(abort)* dessas operações.

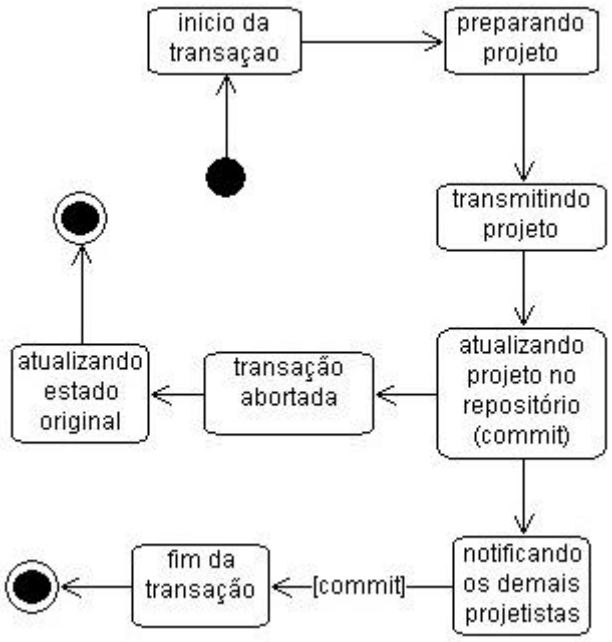


FIGURA 5. 21 - Estados da atualização do projeto

A figura 5.21 representa a atualização do projeto na base de dados por parte do projetista que detém a permissão de escrita. Como visto anteriormente, somente quem tiver o direito de alteração pode atualizá-lo no repositório.

A atualização inicia com a criação de uma transação que reúne todos os dados pertinentes do projeto (preparando o projeto). Após o término da preparação, os dados são transmitidos através da rede até a base de dados. Caso aconteça algum problema, a transação é abordada, os dados são retornados ao seu estado original, e uma nova tentativa de atualização pode ser iniciada. Em caso de perda da conexão, os projetos podem ser armazenados localmente na forma de sistemas de arquivo, o que não permite o compartilhamento dos dados com os demais participantes. No caso de retorno da conexão, os dados podem ser perfeitamente atualizados no repositório de dados.

No momento em que os dados de projeto são armazenados, os demais participantes são notificados de que uma nova versão do projeto está disponível, e a recuperam do repositório. Essa busca, após a atualização, é realizada de forma implícita, ou seja, o projetista não precisa saber como recuperar o projeto, pois isso é feito de forma automática pelo módulo de cooperação.

Após a atualização do projeto e das notificações aos demais participantes, a transação é encerrada. Na figura 5.22 a seguir são ilustradas a requisição de escrita e a transmissão do *token* de escrita a outro projetista.

A requisição de escrita é representada pelo projetista que detém a permissão de leitura. A interação começa com o estado inicial da transação que transmite o *token* de requisição pela rede até chegar no projetista destino. O projetista destino pode receber várias requisições que são tratadas (sincronizadas) e armazenadas em uma lista de requisições. Porém, somente uma requisição pode ser atendida; as outras serão descartadas. O estado final de requisição se encerra com o término da transação.

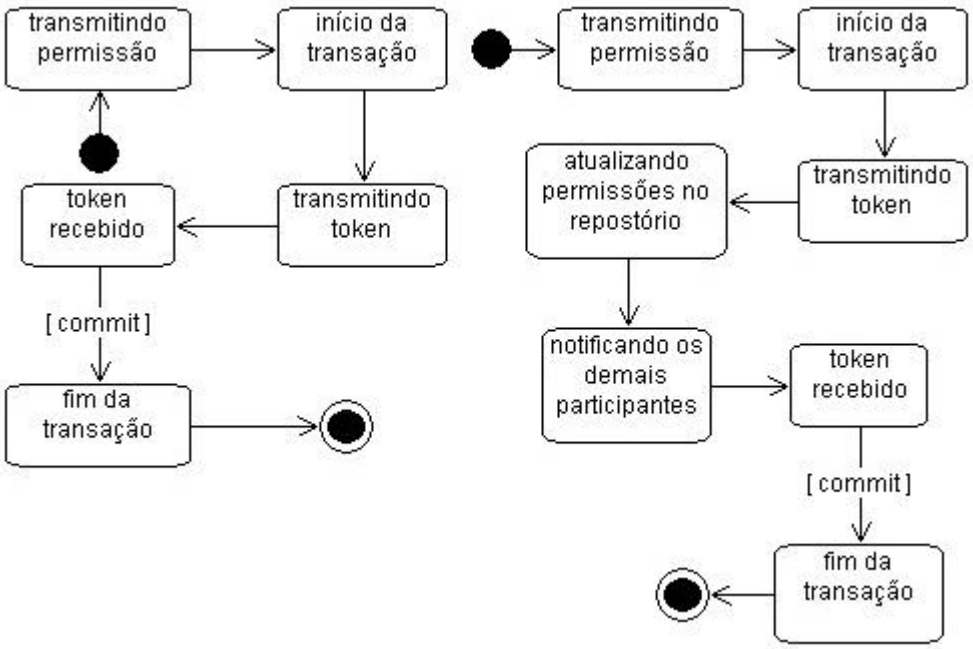


FIGURA 5. 22 - Transmissão de *tokens*

A resposta a uma requisição é realizada com a escolha do projetista que irá receber a permissão de escrita. Neste contexto, o estado inicial começa com a transmissão do *token* através da rede dentro de uma transação. Esse *token* executa métodos que atualizam as permissões no repositório e as replicam para o projetista destino e para todos os demais participantes da sessão. Com isso, as permissões são alteradas e informadas a todos os participantes do projeto.

5.8 Criação de pacotes

Seguindo a mesma estrutura de classes do projeto Cave, as classes implementadas estão localizadas em pacotes de acordo com sua semântica. Nessa estrutura são encontradas as classes desenvolvidas pelo Grupo de Microeletrônica (GME) da UFRGS que dão suporte ao *framework*. Fazem parte não só as classes genéricas a todas as ferramentas e ao *Tool Launcher*, mas também as que são integrantes das diferentes ferramentas disponíveis.

As descrições das classes a seguir estão no escopo do projeto cooperativo proposto. Em especial, serão tratadas aqui somente as classes que possuem ligação e

relevância para a arquitetura de projeto cooperativo. O diagrama ilustrado na figura 5.23 mostra o pacote *collaborative*, que é descrito com mais detalhes nesta seção.

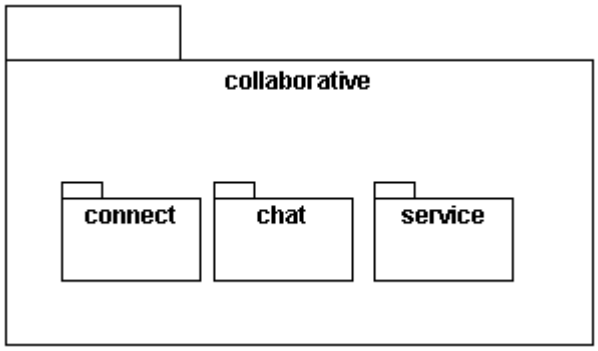


FIGURA 5. 23 - Pacotes que compõem o *Service Collaborative*

5.8.1 cave.collaborative.connect

Esse conjunto de classes tem o objetivo de conectar o lado cliente com a TLS (Tabela de Locação de Serviços). Depois de conectado, a ferramenta cliente recebe a referência da localização dos demais serviços que irão interagir com o *Service Collaborative*. São eles: *Persistence Service* (Serviço de Persistência), representado pelo *Javaspaces*, e o *Transaction Service* (Serviço de Transações). Toda essa interação entre diferentes serviços é realizada de forma transparente para o usuário.

5.8.2 cave.collaborative.chat

A criação desse pacote trata da comunicação entre os projetistas de um sistema ou circuito integrado. Esse conjunto de classes é executado no modo de cooperação especificado na interface gráfica do *Collaborative Service*. Como é implementado de forma modular, seguindo os conceitos de orientação a objetos, pode ser executado em qualquer ponto do *framework*. Em outras palavras, essa implementação pode ser reutilizada não somente no módulo de cooperação, mas como um recurso a mais do *framework*. Assim, projetistas que utilizam as ferramentas do Ambiente Cave podem se comunicar através desse recurso. Seu funcionamento foi detalhado a seção 5.4.5.

5.8.3 cave.collaborative.service

Esse pacote adiciona a maioria das funcionalidades do módulo de cooperação. Nele estão inseridas a metodologia de cooperação - representada por *Pair-Programming*, a interface gráfica de comunicação do *Collaborative Service* com a ferramenta e o modelo de classes que tratam do armazenamento, recuperação e notificação dos dados de projeto. O funcionamento dessas operações foram descritas com mais detalhes nas seções 5.4.3. e 5.4.6. A integração da interface gráfica do módulo de cooperação com a ferramenta é modelada e implementada através de um método estático chamado *createMenu()* pertencente a classe *CollaborativeExtendMenu*, que insere suas operações no menu das ferramentas do *framework*.

5.9 Contribuição deste trabalho para o Ambiente Cave

A figura 5.24 adiante mostra a estrutura do Ambiente Cave original. Baseava-se exclusivamente num modelo centrado em hiperdocumentos e sistema de arquivos. Apesar de todos avanços gerados na arquitetura original do Projeto Cave, algumas modificações foram necessárias para introduzir conceitos de trabalho cooperativo no *framework*. A partir dessa situação, o modelo orientado a objetos foi pesquisado com vistas a substituir o modelo de transmissão e armazenamento de dados baseado em hiperdocumentos e sistema de arquivos por um modelo baseado em objetos.

Essa substituição foi planejada quando se percebeu a necessidade de se ter projetistas cooperando em um mesmo projeto através de uma rede. O armazenamento de dados baseado em sistema de arquivos não se mostra muito eficiente no tratamento de acessos multiusuário e armazenamento de dados complexos. Então, algumas alternativas de armazenamento de dados foram pesquisadas para que o tratamento dos dados não fosse visão de somente um projetista, mas sim uma possibilidade para que vários projetistas pudessem interagir com seus projetos de uma forma organizada.

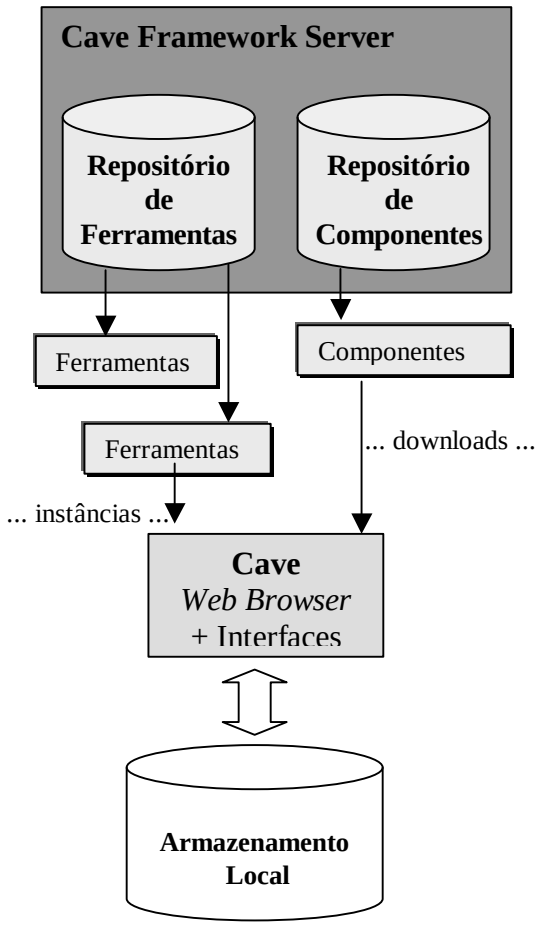


Figura 5. 24 – Estrutura da proposta original do Ambiente Cave

O resultado desse trabalho pode ser visualizado na figura 5.25, que mostra o módulo de cooperação *(Collaborative Service)* incorporado ao Ambiente Cave. Para que isso fosse possível, uma arquitetura de apoio teve que ser implementada, e se chamou *Service Space.* Com isso, é possível perceber a interação do serviço de cooperação com o repositório de dados, responsável por gerenciar todas as ações relacionadas com a cooperação, dentre elas: acesso a projetistas, criação e recuperação de projetos, agenciamento de transações, administração de permissões, entre outras.

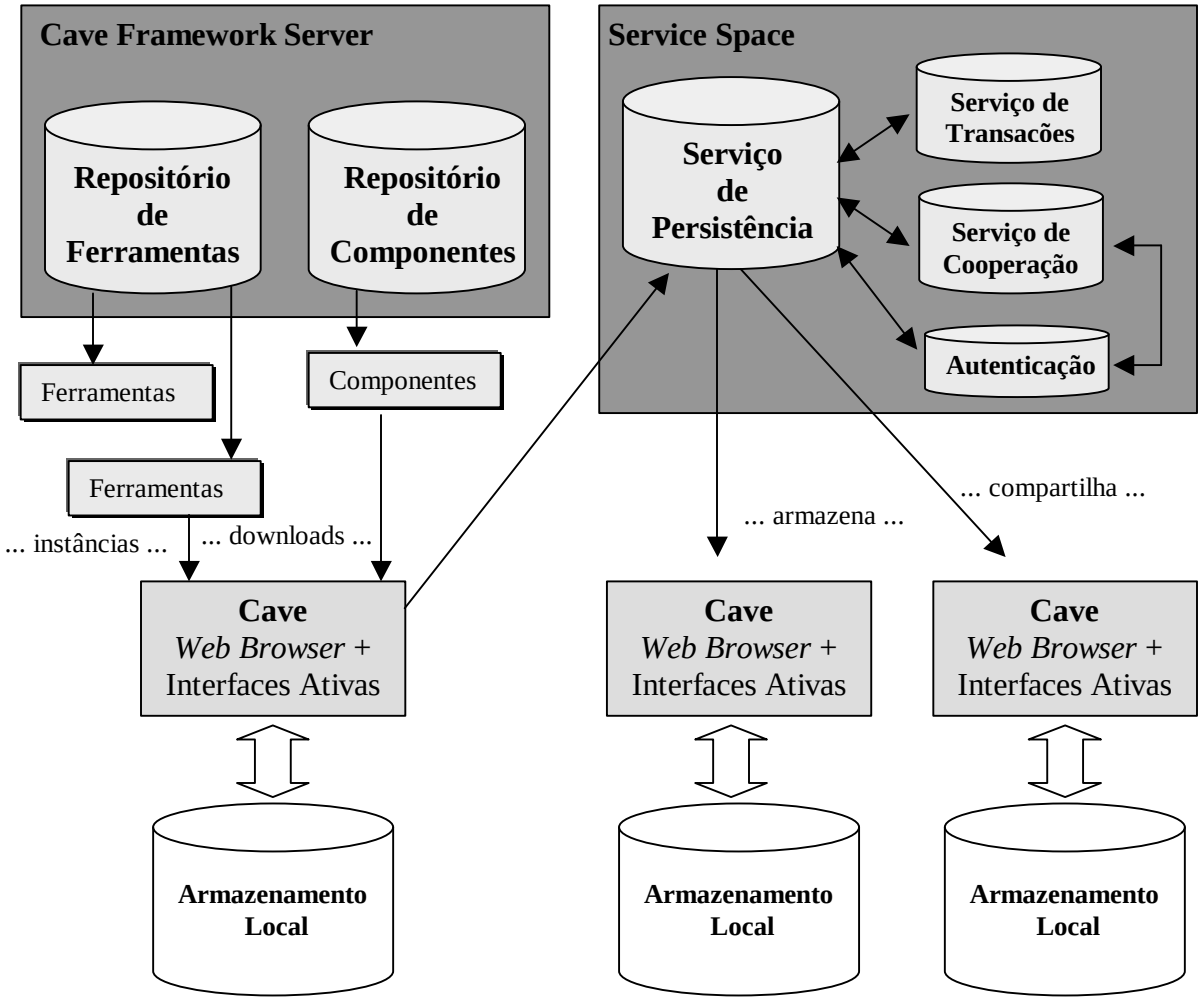


Figura 5. 25 – Ambiente Cave após a inserção do módulo de cooperação

5.10 Resumo de capítulo

Este capítulo detalhou o desenvolvimento do modelo utilizado para a criação do *Collaborative Service*. Esse modelo segue o padrão UML *(Unified Modeling Language)*, tal como ilustrado pelos diagramas desta proposta. Foram abordadas também, estratégias de armazenamento, comunicação, recuperação e notificação, modeladas e criadas para a arquitetura do módulo de cooperação. Para a implementação e validação desse modelo, foram criados e especificados três pacotes de classes que poderão ser

reutilizados por qualquer uma das ferramentas do ambiente Cave. Por último foi apresentado a contribuição deste trabalho no escopo do Ambiente Cave, tendo sido ilustradas as arquiteturas trabalhando em conjunto (Cave e *Collaborative Service incorporado ao Service Space*).

6 Estudo de Caso 1: *Blade*, Um Editor de Diagramas Esquemáticos Hierárquico

6.1 Introdução

Este capítulo mostra o *Collaborative Service* trabalhando em conjunto com a ferramenta *Blade*, um editor de diagramas esquemáticos hierárquico proposto por Brisolara [BRI 2002, BRI 2002a]. *Blade* é uma ferramenta gráfica para projeto de CIs que compõe o *Framework Cave*. Está voltada para ambientes distribuídos, permitindo a operação remota, visando também a especificação de sistemas através de representações gráficas.

Nesse capítulo demonstra-se a integração do *Collaborative Service* com a ferramenta *Blade*, sendo descritas suas características de aplicação no Ambiente *Cave*. Em outras palavras, esse estudo de caso mostra a cooperação em nível de diagramas através do módulo de cooperação utilizando-se apenas poucas linhas de código.

6.1.1 Característica da Ferramenta *Blade*

Um editor de diagramas no contexto deste trabalho é uma ferramenta que permite especificar um sistema graficamente através do uso de diagramas. A idéia do projeto é disponibilizar no *Cave* uma ferramenta que servirá como interface gráfica entre o projetista e outras ferramentas de CAD disponíveis no ambiente de projeto. Podendo essa ferramenta ser integrada a outras ferramentas, por exemplo, a um editor de descrições HDL, permitindo descrições tanto textuais como gráficas para um sistema.

Além disso, o editor de diagramas deve suportar diferentes tipos de representações gráficas, permitindo a entrada de um projeto utilizando descrições em diferentes níveis de abstração. Ao final da especificação, o editor gera uma descrição comportamental do sistema usando uma linguagem de descrição de *hardware* ou, ainda, uma descrição em mais alto nível usando uma linguagem de descrição de sistemas (SDL, do inglês, *Systems Description Language*).

O *Blade* suportará primeiramente somente diagramas em nível lógico. Porém, foi desenvolvido com o intuito de ser estendido a outras formas de diagramas e permitir o uso de especificações em outros níveis de abstração. Esta extensibilidade é um aspecto muito importante da ferramenta porque facilita o reuso de suas classes. Assim, o editor pode ser estendido a fim de permitir a especificação de sistemas usando novas classes de diagramas que venham a ser utilizadas na área de CAD e no projeto de CIs. Além disso, o reuso das classes do editor pode permitir o desenvolvimento de outros editores que utilizam outras representações gráficas, ou seja, outras formas de diagramas.

6.2 Integração do *Collaborative Service* com a ferramenta Blade

A integração do módulo de cooperação com a ferramenta *Blade* é simples. O desenvolvedor de ferramentas precisa apenas implementar a *interface DocumentListener* na classe *Handler* – a estrutura do *framework* padroniza a utilização de uma classe para manipulação de eventos. Nessa estrutura, a interface *DocumentListener* implementa três métodos abstratos chamados *insertUpdate(), changeUpdate() e removeUpdate()*. Esses métodos recebem as mensagens de notificação do módulo de cooperação tais como permissões de escrita/leitura, atualizações de projeto e participantes e as mensagens do *chat*.

O quadro abaixo mostra a alteração de código necessária para que o módulo de cooperação trabalhe em conjunto com a ferramenta *Blade*. A classe *Handler* da ferramenta *Blade* chama-se *HierarchicalBladeHandler*.

```
import cave.collaborative.connect.*;
import cave.collaborative.chat.*;
import cave.collaborative.service.*;

public class HierarchicalBladeHandler … implements … DocumentListener
{
…
   public void insertUpdate(DocumentEvent e) {}
   public void changedUpdate(DocumentEvent e){}
   public void removeUpdate(DocumentEvent e) {}
…
}
```

Na declaração da classe *Handler* é necessário que sejam declarados os pacotes do módulo de cooperação. Essa declaração é muito importante pois faz com que a ferramenta consiga localizar e reutilizar o conjunto de classes do *Collaborative Service*. A especificação dos pacotes e sua criação foram descritas com mais detalhes na seção 5.6. O quadro a seguir descreve as alterações no método *insertUpdate()*. Ele é quem recebe todas as notificações e atualizações dos projetos e de seus participantes.

```
public void insertUpdate( DocumentEvent e )
{
…
   collaboration.getNotificationProject()…
…
   collaboration.getProject()…
…
   collaboration.getNotificationParticipant()…
…
   collaboration.getParticipant()…
…
   collaboration.getWritePermission()…
…
}
```

A inserção das funcionalidades do módulo de cooperação altera somente o método *insertUpdate()*; os demais métodos devem ser somente declarados no corpo da classe, pois se tratam de métodos abstratos da interface *DocumentListener*.

O método *getNotificationProject()* recebe a notificação de que o projeto foi modificado. Sua função é de notificar o projetista através de uma mensagem de que houve atualização do projeto no repositório. Com isso, a última versão pode ser recuperada e atualizada na tela da ferramenta.

O método *getProject()* retorna o projeto atualizado do repositório. Ele é executado sempre que os projetistas forem notificados de que houve uma atualização do projeto no repositório. Em outras palavras, é executado sempre que o método *getNotificationProject()* for disparado.

As mensagens das atualizações dos participantes são recebidas pelo método *getNotificationParticipant()*. Nesse método recebe-se notificações sempre que um projetista entrar ou sair de uma sessão cooperativa ou quando houver mudança nas permissões de escrita/leitura.

Depois de recebida a notificação de atualização dos participantes, o método *getParticipant()* é disparado para atualizar a tela dos projetistas. A função principal desse método é ajudar a percepção dos participantes de uma sessão cooperativa, transmitindo ao grupo os projetistas que estão *on-line* e suas permissões.

O método *getWritePermission()* é disparado sempre que houver modificação no portador do direito de escrita. Assim, além de aparecerem as permissões na lista de projetistas *on-line*, aparecerá também na interface que serve como "sinaleira" para as permissões de escrita e leitura. Ou seja, verde para quem detém o direito de escrita e vermelho para quem detém o direito de leitura.

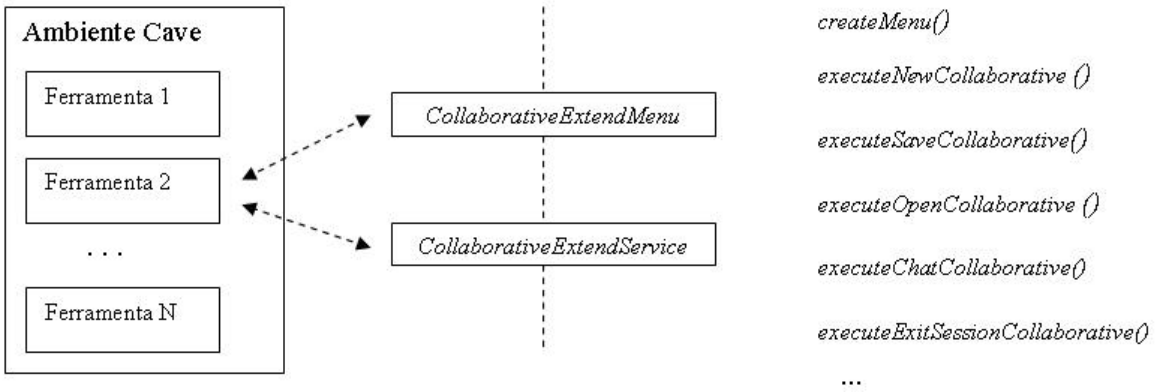


FIGURA 6. 1 - Interfaces intermediárias

Existem duas classes que servem de interface entre a ferramenta e o serviço de cooperação. São elas *CollaborativeExtendMenu* e *CollaborativeExtendService*,

ilustradas pela figura 6.1. A classe *CollaborativeExtendMenu* é a encarregada de adicionar a interface gráfica do módulo de cooperação. Para que isso aconteça, é necessário executar um método estático chamado *createMenu()*, passando como parâmetro o *menubar* padrão da ferramenta e sua interface *ActionListener*. A seguir está representada a linha de código adicional para a inserção do menu de cooperação.

*CollaborativeExtendMenu.**createMenu**(menubar,this);*

O resultado da inserção dessa linha pode ser visualizada na figura 6.2. Nesse caso, o menu foi inserido na ferramenta *Blade*.

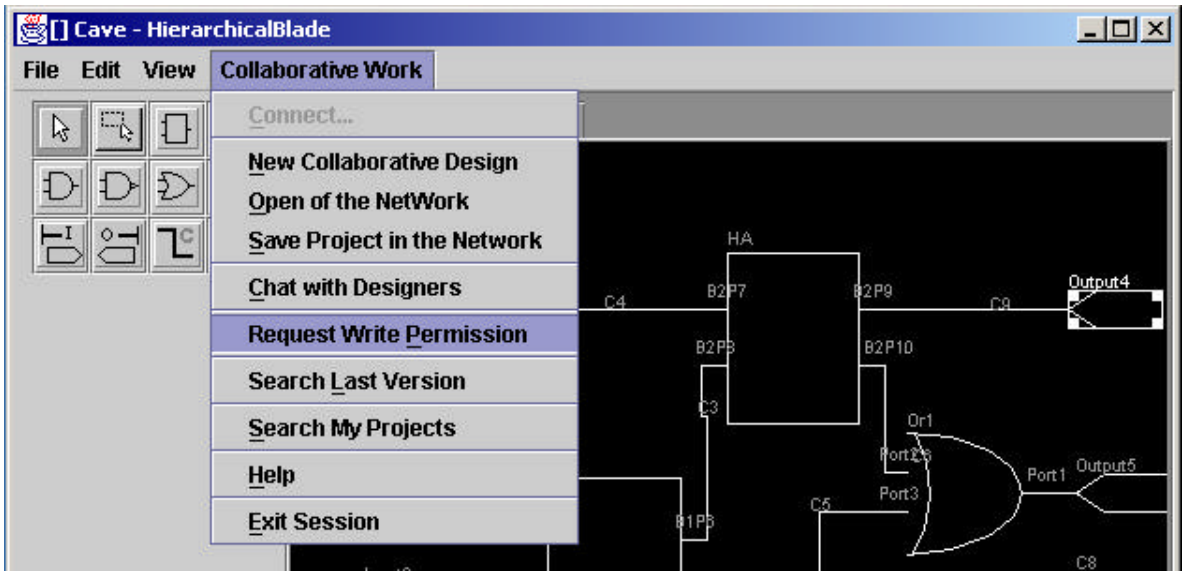


FIGURA 6. 2 - Menu de cooperação inserido na ferramenta *Blade*

Cada uma das opções descritas na figura 6.2 tratam especificamente do módulo de cooperação. A opção *Connect* faz a comunicação da ferramenta com a tabela de locação de serviços (TLS), que retorna a referência dos serviços de transação e persistência, deixando a ferramenta pronta para a cooperação. Essa comunicação é realizada através do método estático *executeServiceColl()*. A execução dessa opção habilita os demais recursos ligados a interação entre os projetistas. São eles:

?? ***New Collaborative Design:*** Essa opção cria um novo projeto cooperativo e o armazena no repositório de projetos *(Persistence Service)*. Esse procedimento, assim como as atualizações, são realizadas dentro de uma transação *(Transaction Service)*. Para realizar essa operação é chamado o método estático da classe *CollaborativeExtendService* chamado *executeNewCollaborative ()*, passando-se como parâmetro o nome do projeto e o nome de seus blocos. A figura 6.3 ilustra a criação de um novo projeto cooperativo, adicionando-se o nome do projeto, seu responsável, uma senha de acesso, os blocos que compõem esse projeto, e por último a inserção do grupo de projetistas que irão trabalhar nesse projeto. Detalhes no armazenamento foram abordados na seção 5.5.

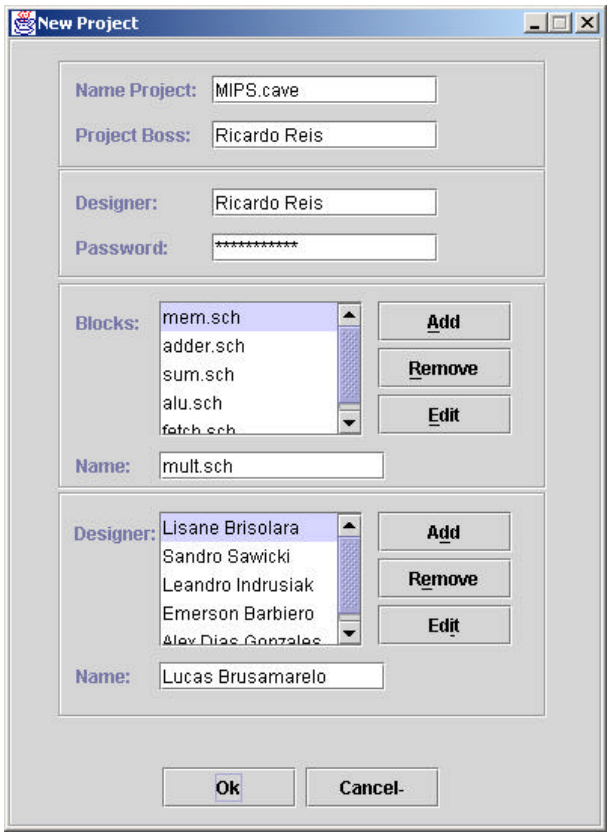


FIGURA 6. 3 - Criação de um novo projeto cooperativo

?? ***Open of the Network:*** Essa opção recupera projetos armazenados no repositório através da execução do método estático *executeOpenCollaborative()*. A recuperação de um projeto e seus blocos é realizada através de parâmetros inseridos pela interface gráfica.

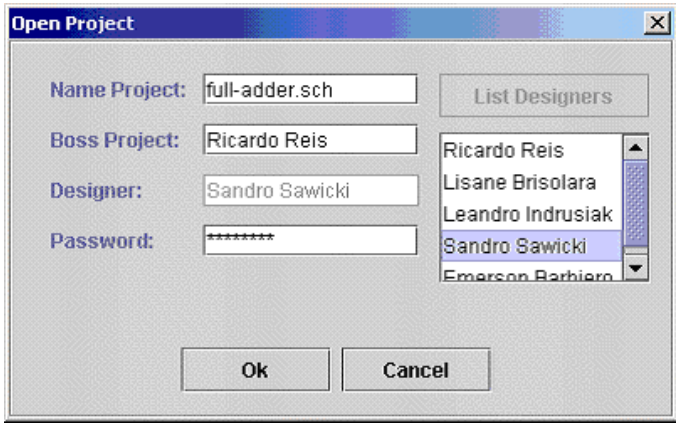


FIGURA 6. 4 - Abrindo um projeto compartilhado

?? ***Save Project in the Network:*** Essa opção atualiza um projeto no repositório. Somente estará disponível se o projetista estiver com a permissão de escrita.

Caso contrário, sua tela, além de não permitir a gravação através dessa opção, não permitirá a edição do projeto. Sua execução é realizada através do método estático *executeSaveCollaborative ()*, passando-se como parâmetro o bloco de projeto que será atualizado. É importante ressaltar, novamente, que todas as atualizações são realizadas dentro de uma transação.

?? ***Chat with Designers:*** Essa opção permite a interação entre os projetistas através de um *chat* de texto. A figura 6.5 mostra a conversa entre dois projetistas no intuito de auxiliar a conclusão no projeto. A execução do módulo de *chat* é realizada através do método estático *executeChatCollaborative()*. Como mencionado anteriormente, pesquisas no GME (Grupo de Microeletrônica da UFRGS) estão sendo realizadas para inserir voz na comunicação entre os projetistas.

FIGURA 6. 5 - Interação entre dois projetistas através do *chat*

?? ***Request Write Permission:*** Essa opção tem como função requisitar a permissão de escrita para outro projetista. Para que isso aconteça, o método *requestWritePermission()* é disparado uma mensagem chega até o projetista portador da permissão de escrita. Ele por sua vez, escolhe a quem deseja passar a permissão.

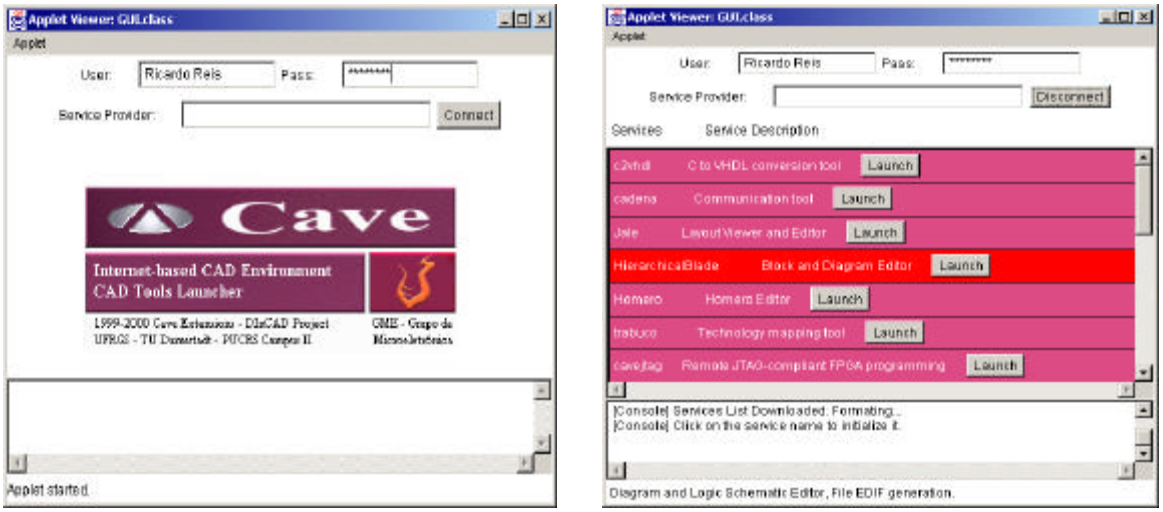
?? ***Search Last Version:*** Essa opção está disponível somente para o projetista que detém a permissão de escrita. Recupera a última versão do projeto executando o método *executeSearch LastVersionCollaborative()*.

?? ***Help:*** Essa opção executa um *help on-line*, no qual mostra como utilizar o *Collaborative Service*. Dispara o método *executeHelpOnLine Collaborative()*, que busca um formulário HTML que especifica o funcionamento detalhado do módulo de cooperação.

?? ***Exit Session:*** Essa opção tem como objetivo permitir que um projetista saia de uma sessão cooperativa. Ao sair da sessão, os demais projetistas são notificados e atualizados. Caso a saída da sessão aconteça com quem detém a permissão de escrita, a permissão é passada ao próximo projetista da lista.

6.3 Demonstrando a aplicação com o Collaborative Service

Como visto no capítulo 2, o Cave2 pode ser executado sobre a Internet usando o protocolo HTTP juntamente com *browsers* compatíveis com a linguagem Java, ou *standalone* através de uma conexão *socket*. A distribuição de recursos de projeto pode ser facilmente realizada, visto que a arquitetura suporta a cooperação entre vários *Framework Servers* e *Service Spaces*. Porém, atualmente utiliza-se uma abordagem não redundante, ou seja, tem-se um único *Framework Server* e um único *Service Space*. Assim, como mostra a seção 5.7 *(Service Space)*, toda a representação de projetos e de módulos de ferramentas de projeto estão armazenadas num único *Framework Server* e um protocolo interno é usado para o compartilhamento de tais recursos. Com isso, todos os serviços são disponibilizados num único *Service Space* e podem ser localizados através de um mecanismo de *lookup* descrito em [SAW 2002], é utilizado como um intermediário entre o cliente e as aplicações.



(a) (b)

FIGURA 6. 6 - Cave GUI e Tool Launcher

A figura 6.6 (a) e 6.6 (b) ilustram respectivamente as interfaces gráficas do Ambiente Cave e do *Tool Launcher*. Com isso, o projetista pode se conectar ao *Server Provider* e criar uma instância de *Tool Launcher*. Este, por sua vez, disponibiliza ao projetista uma lista de ferramentas para a sua utilização.

A figura 6.7 ilustra a ferramenta *Blade* trabalhando em um projeto cooperativamente. Nesta ilustração, é possível visualizar alguns pontos de destaque na interface cooperativa. São eles o menu de cooperação, a percepção dos projetistas que estão trabalhando *on-line* no projeto juntamente com suas permissões, a barra de *status* que indica o direito de escrita ou leitura, a interface de *chat* e o grupo de projetistas que estão trabalhando do projeto.

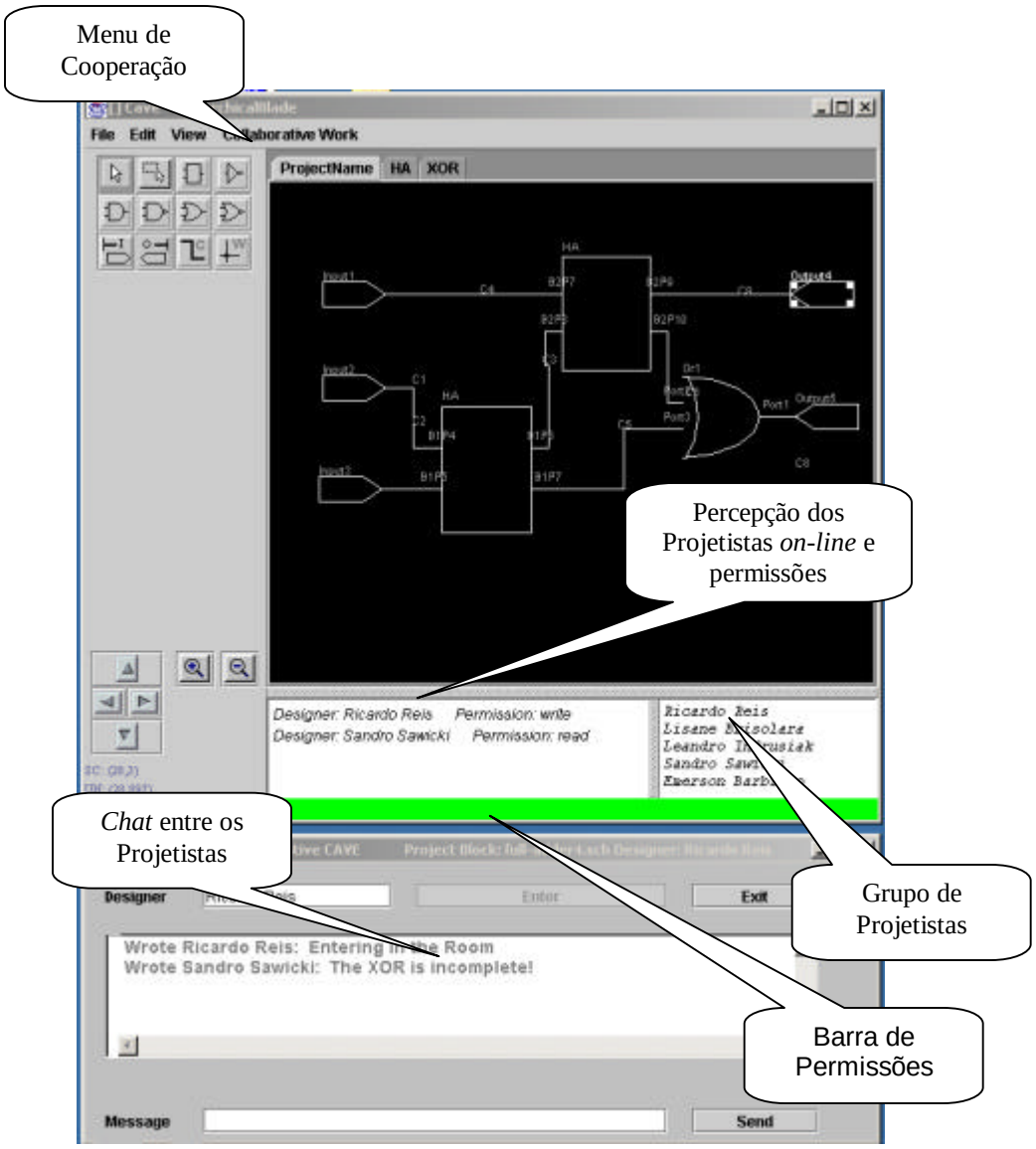
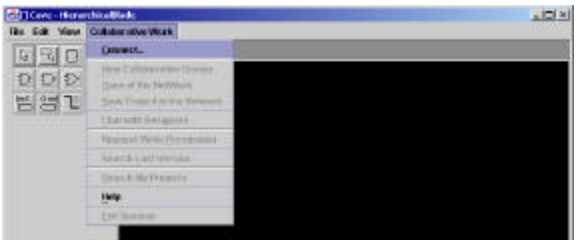
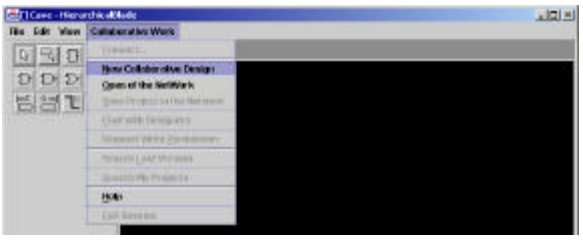


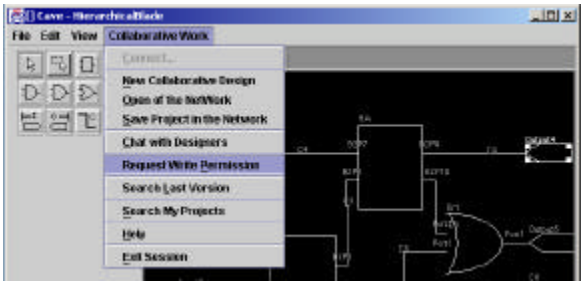
FIGURA 6. 7 - Pontos após a inserção do *Collaborative Service*



(a)



(b)



(c)

FIGURA 6. 8 - Etapas da comunicação com o repositório de dados

A seqüência de figuras acima demonstra as diferentes visões dos recursos gráficos, criadas para garantir a segurança do projetista em situações de inconsistência da aplicação. Na figura 6.8 (a), as opções de cooperação não estão disponíveis, com exceção do *Help e Connect*. Isso acontece por que a ferramenta ainda não se conectou com a base de dados e nem com o serviço de cooperação. Depois de conectado, como ilustra a figura 6.8 (b), as opções de criação e abertura de projeto tornam-se disponíveis. Logo após a criação ou abertura de um projeto existente, todas as opções ficam disponíveis ao projetista, como ilustra a figura 6.8 (c).

Conforme exposto, *Blade* é um editor de diagramas hierárquico, portanto permite a criação de projetos hierárquicos nos quais os projetistas podem definir diferentes níveis de hierarquia. Em termos de visualização, o editor permite a navegação através destes níveis, possibilitando seu armazenamento. Num projeto cooperativo, quando um projetista faz uma alteração em um dos níveis, os demais projetistas devem ser notificados. Este estudo de caso permitiu validar o serviço de cooperação utilizando diagramas simples, e até mesmo hierárquicos. A facilidade com que a integração do serviço com a ferramenta deu-se, comprova a flexibilidade desse serviço. Portanto, a hierarquia do *Blade* não precisa ser tratada necessariamente pelo serviço. Quando um projeto hierárquico é salvo no repositório, todos os níveis hierárquicos do projeto também são armazenados em um só objeto.

É importante ressaltar que os métodos construídos para o serviço de cooperação possibilitam também o armazenamento de níveis hierárquicos em separado. Com esse recurso, projetistas podem ter visões e interações em cada um dos níveis hierárquicos do projeto.

Na figura 6.9, seqüências (a), (b) e (c) são ilustrados três níveis de hierarquia para a representação de um *full-adder*. Nos painéis de cooperação estão ilustrados os projetistas participantes da sessão cooperativa que estão *on-line* e também suas permissões (percepção). No painel ao lado estão ilustrados todos os participantes do projeto. Nesse caso, a seqüência de figuras apresenta um projetista com a permissão de escrita, indicada pela cor verde na barra de *status.*

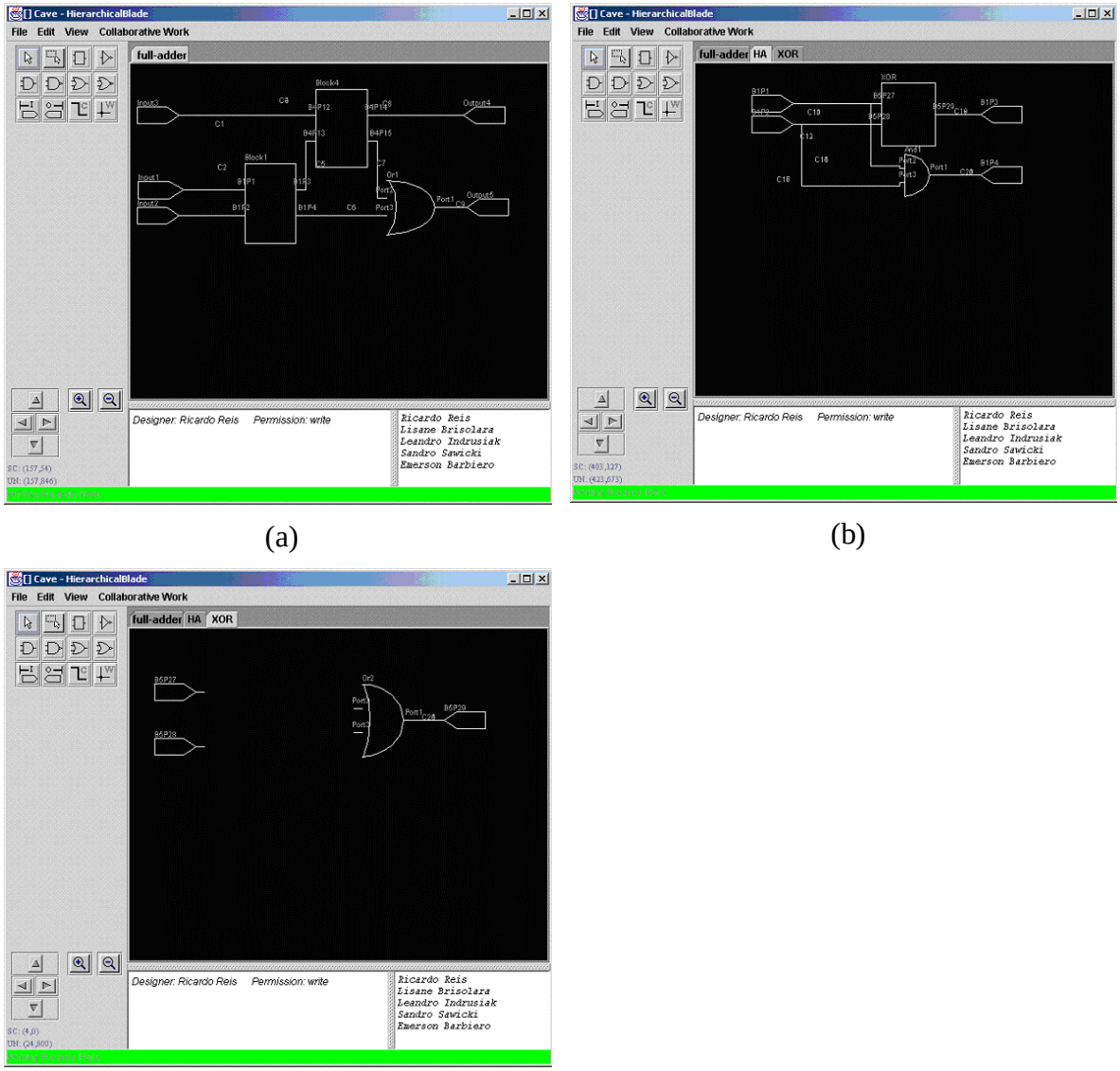


FIGURA 6. 9 - Representação dos níveis hierárquicos em um *full-adder*

Até o momento, somente um projetista está trabalhando em um bloco de projeto. A figura 6.10 ilustra dois projetistas interagindo em um projeto com níveis hierárquicos. É possível perceber que ambos têm a mesma visão do projeto e de sua hierarquia, o projetista que entrou na sessão cooperativa obteve a permissão de leitura,

ilustrada pela cor vermelha na barra de *status*. Os dois projetistas trocam idéias através do *chat* de texto, situado abaixo da ferramenta. É importante destacar que essa interação foi executada em uma máquina somente para fins ilustrativos, mas é possível executá-la em diferentes plataformas de hardware e/ou software espalhadas remotamente pela rede. A seqüência de figuras abaixo, mostra a interação entre dois projetistas, seguindo a metodologia de cooperação adotada, *Pair-Programming*.

As figuras 6.10 a 6.14 mostram a interação de dois projetistas usando como exemplo o projeto de um *full-adder*. Como se pode perceber, este projeto possui três níveis hierárquicos que podem ser acessados através de "abas" na parte superior da área de trabalho. O *full-adder* é um circuito bastante simples, para tal não seriam necessários vários níveis de hierarquia, porém utilizou-se uma abordagem hierárquica a fim de permitir a exploração desta funcionalidade por parte do editor e do módulo de cooperação.

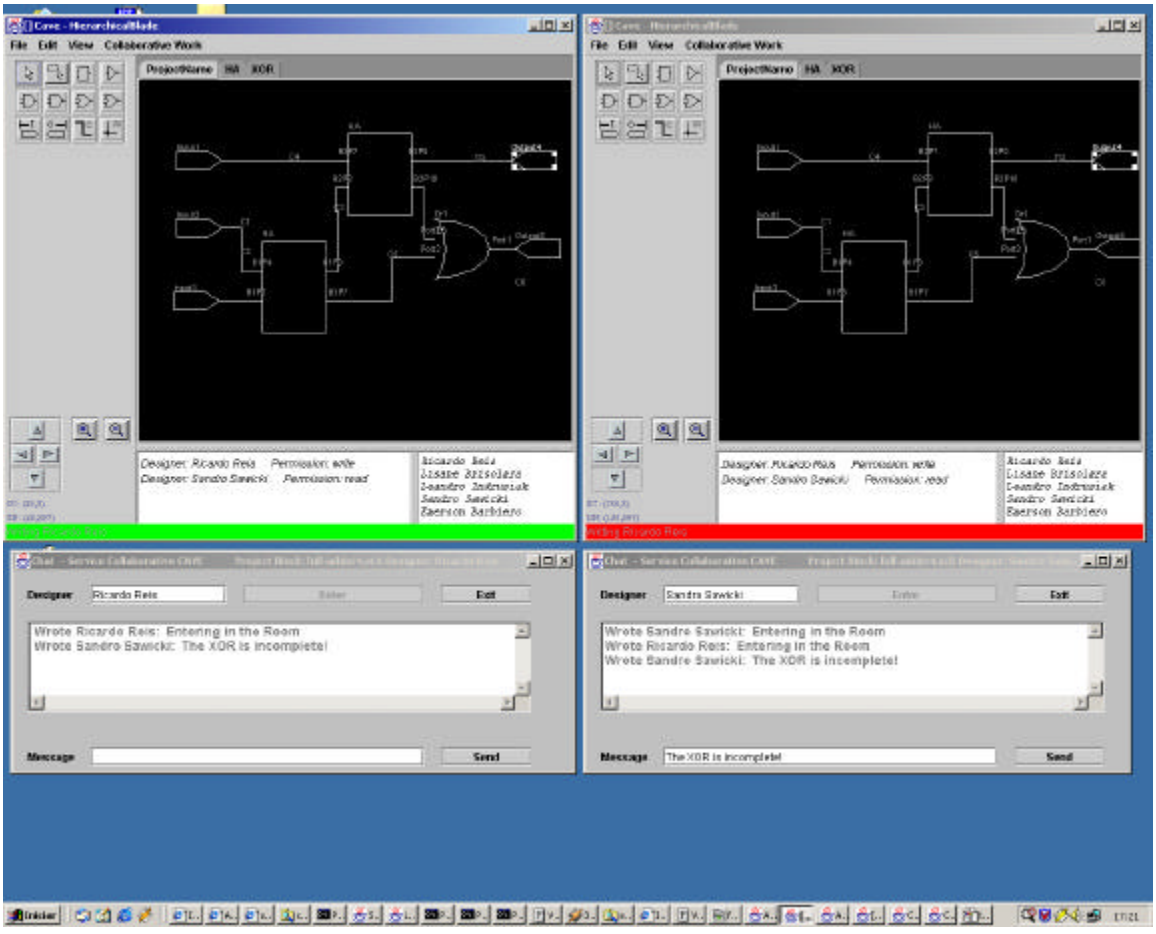


FIGURA 6. 10 - *Full-Adder* Nivel 1: interação entre dois projetistas

O primeiro nível representa o topo da hierarquia de projeto, sendo composto por dois blocos que representam *half-adders* e uma porta lógica OR, conforme pode ser visualizado na figura 6.10.

Na figura 6.11 vê-se o segundo nível da hierarquia de projeto, que representa o circuito correspondente a um *half-adder*. Este circuito é composto por uma porta AND e um outro bloco que representa uma porta lógica EX-OR. A porta EX-OR foi representada neste exemplo utilizando as portas lógicas OR, NOT e AND, somente para fins ilustrativos. O *half-adder* foi instanciado duas vezes para a composição do *full-adder* no nível superior da hierarquia do projeto. É possível perceber nesta ilustração, a fácil navegação através dos níveis hierárquicos desse circuito:

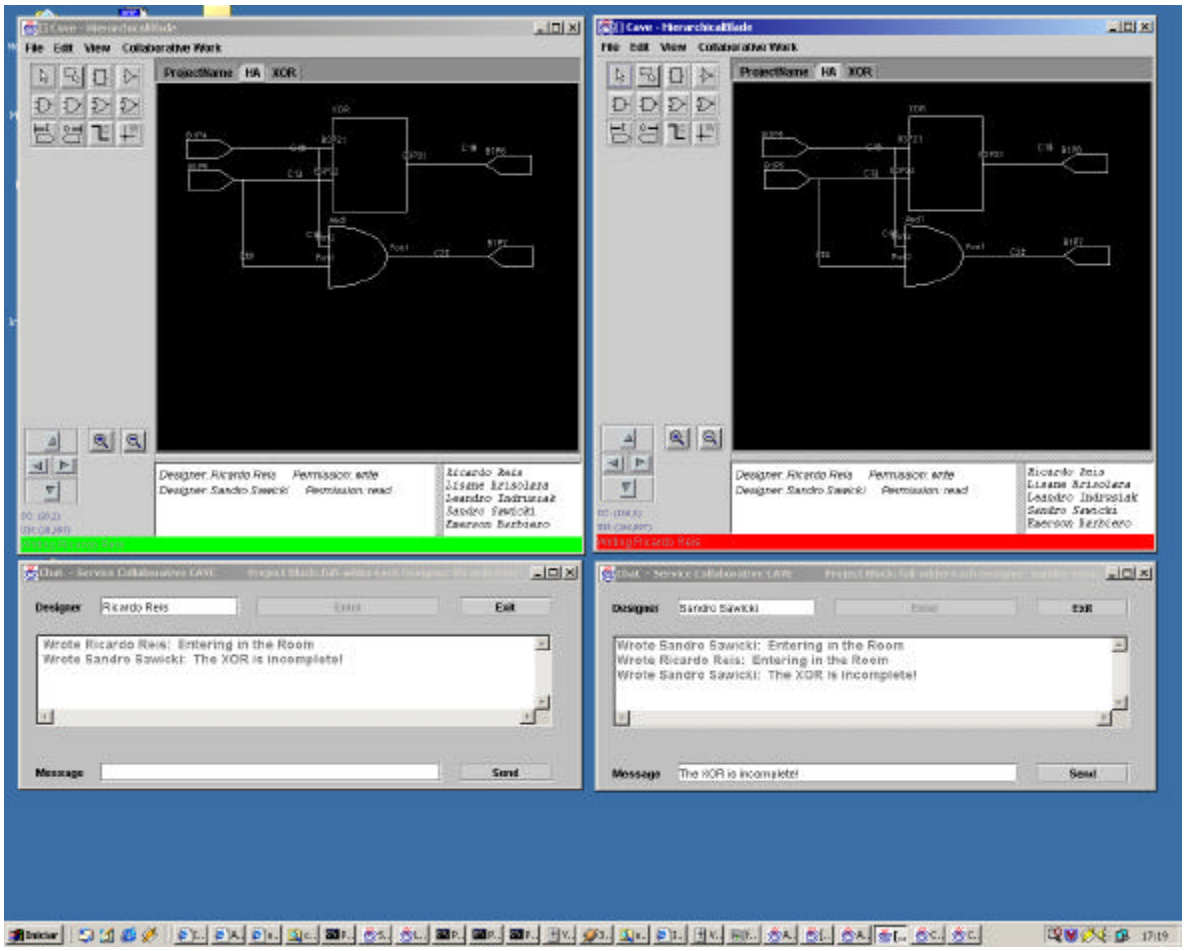


FIGURA 6. 11 - Nível 2: *Half-Hader* instanciada

A figura 6.12 adiante, mostra o nível mais baixo da hierarquia do projeto de um *full-adder*. Ele é representado pelo circuito correspondente à implementação de uma porta lógica EX-OR.

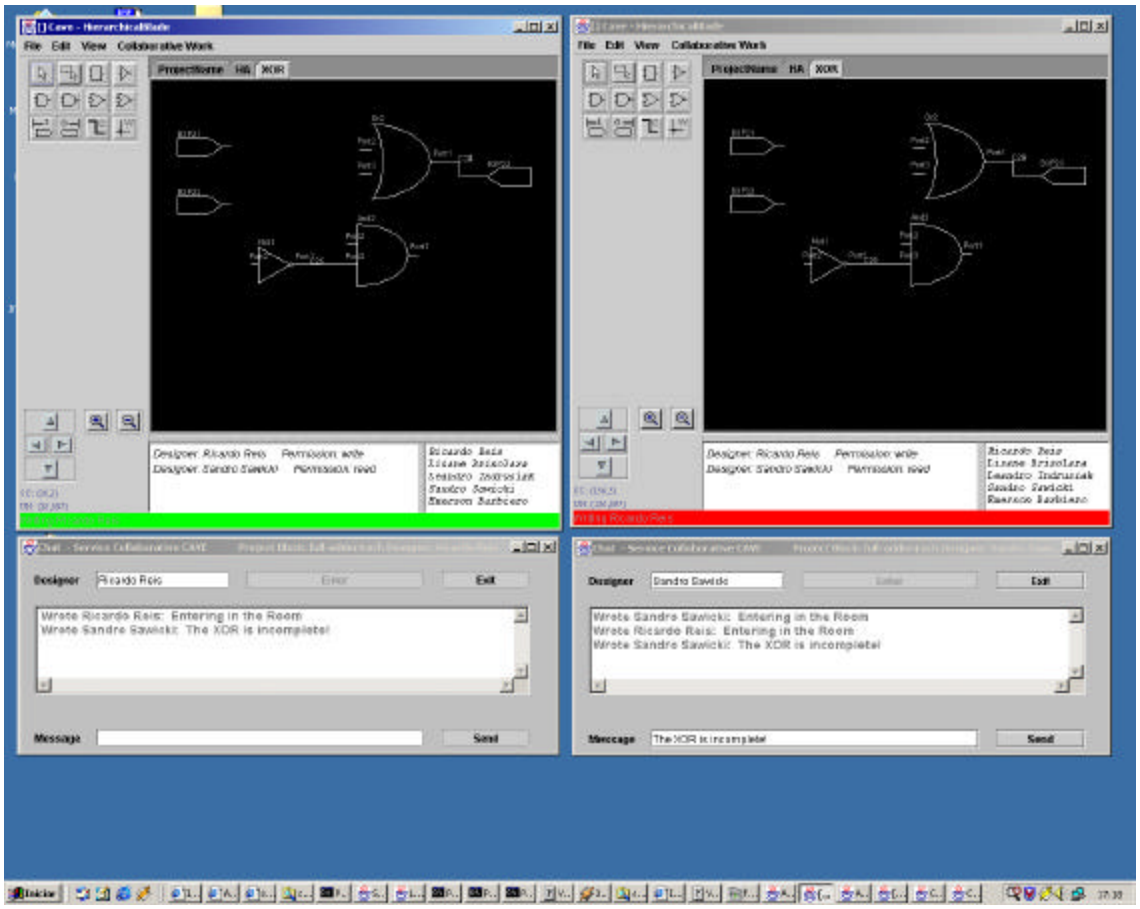


FIGURA 6. 12 - Nível 3: EX-OR inacabada

Qualquer alteração realizada pelo projetista que detém a permissão de escrita pode ser propagada aos demais participantes. Nessa situação, a figura 6.12 traz a a construção de uma porta lógica EX-OR, na qual ambos projetistas têm a mesma visão do circuito. No momento em que o projetista escritor alterar o projeto, ele pode atualizá-lo no repositório de dados, isso conseqüentemente, resulta em uma sessão de notificações a todo o grupo de projetistas envolvidos no projeto. A atualização, por sua vez, pode ser instantânea ou um aviso pode ser disparado mostrando ao projetista participante que uma nova atualização foi realizada. No momento, apenas a atualização instantânea foi implementada, mas o modelo permite que a notificação seja realizada através de avisos de atualização aos projetistas.

O estado de cada componente gráfico (objeto) é preservado no repositório de dados e visualizado por todo o grupo de projetistas. Na seqüência, a figura 6.13 mostra a seleção de um pino de saída realizado pelo projetista escritor. Nesse sentido, quando o projeto for atualizado, o mesmo componente selecionado pode ser visualizado por todo o grupo de projetistas. Esse é mais um fator que caracteriza esse modelo como sendo tipicamente orientado a objetos.

Nesse modelo, somente um projetista pode alterar o projeto; os demais participantes são ouvintes. Entretanto, caso um participante queira editar o projeto, pode requisitar a escrita ao projetista escritor. Nesse sentido, muitos projetistas podem requisitar a escrita, portanto, o escritor deve estar preparado para receber vários pedidos de permissão de escrita. O tratamento para esse caso foi detalhado na sessão 5.5.9.

A figura 6.13 mostra o pedido de escrita de um projetista ouvinte para um projetista escritor. Após a autorização do projetista escritor, as interfaces gráficas de ambos projetistas são alteradas, ou seja, o projetista escritor que tinha a barra de *status* verde (permissão de escrita – edição do projeto), passa a ser ouvinte (permissão de leitura – não edição do projeto) e sua barra de *status* muda para a cor vermelha.

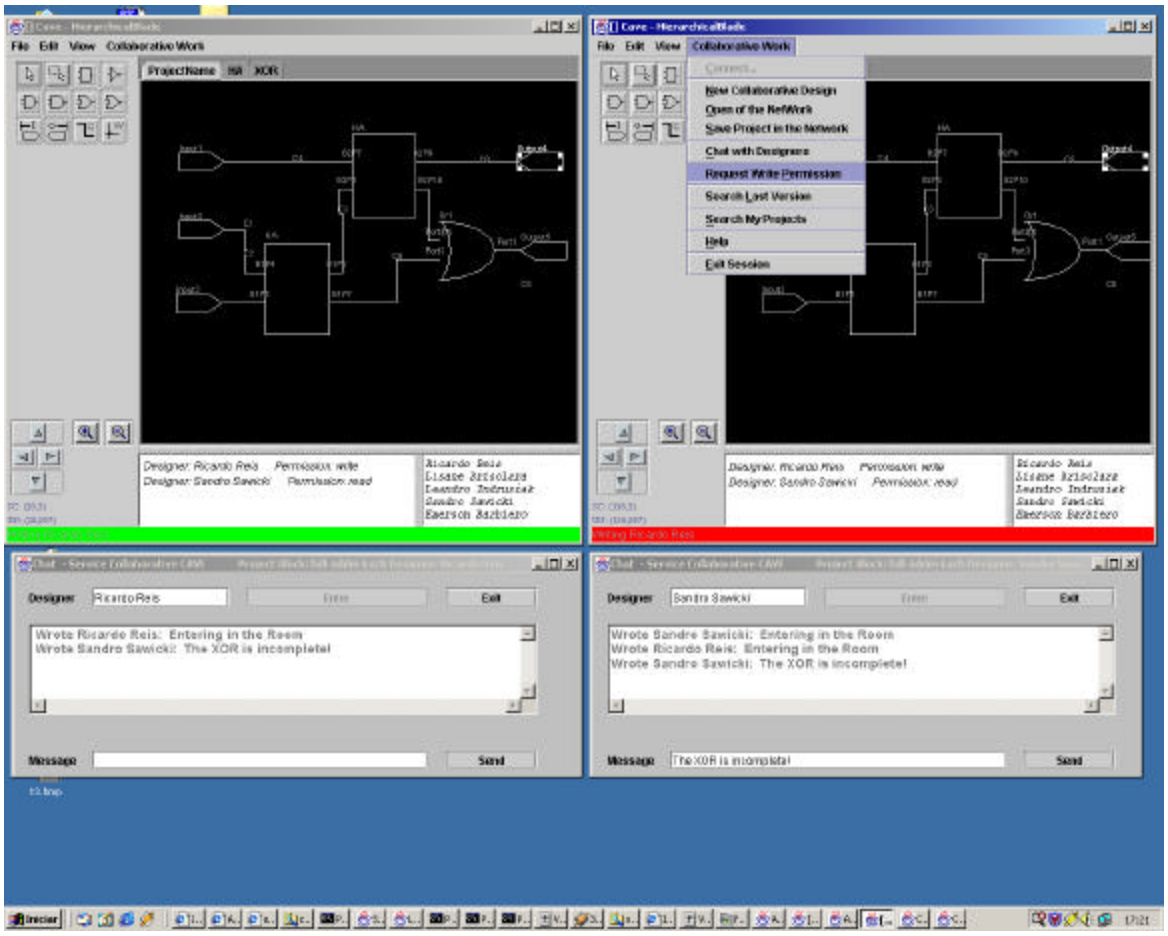


FIGURA 6. 13 - Requisição de escrita

A troca de permissões está ilustrada na figura 6.14 a seguir. Além, das mudanças na interface de cada ferramenta, o componente responsável pela percepção também é modificado e repassado aos demais projetistas. Em outras palavras, todo o grupo de projetistas envolvidos em um projeto comum sabe quem está interagindo com ele no momento, e quem é o projetista que detém o direito de escrita.

Um ponto importante a destacar nesse modelo é que, em cada bloco, existe um projetista com a permissão de escrita, mas nada o impede de atuar como escritor em outros blocos do mesmo projeto. Existe, sim, um único responsável pelo projeto que, além de coordenar os passos de cada projetista, criar projetos novos, remover projetistas, pode adicionar outros participantes ao de projeto.

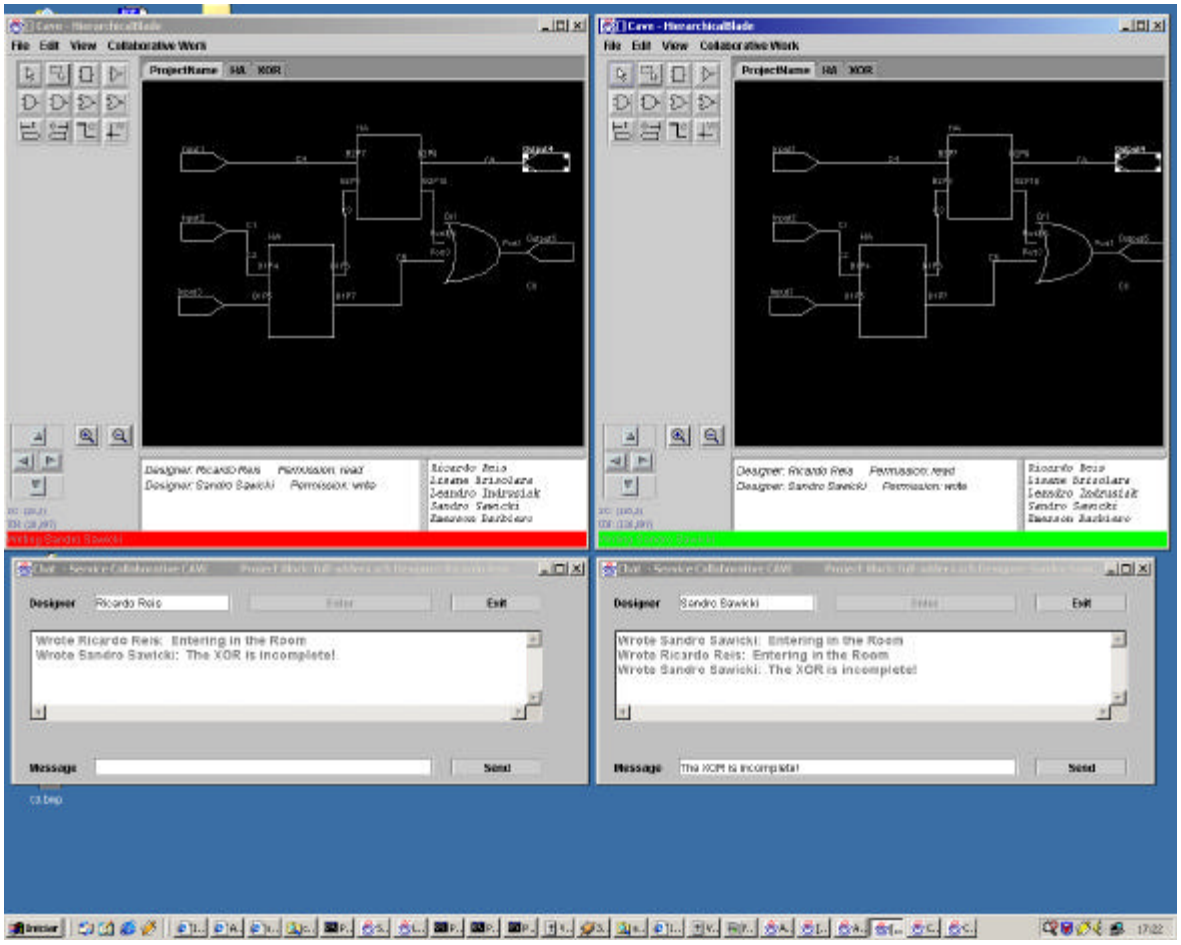


FIGURA 6. 14 - Transferência da permissão de escrita

6.4 Modelos de Visualização

A ferramenta Blade trabalha com a separação dos dados de projeto e dados de visualização. Essa característica possibilitou a criação de dois modos, visualmente acoplado e visualmente desacoplado [IND 2001]. No modo visualmente acoplado, qualquer modificação, tanto nos dados de projeto, quanto nos dados de visualização, propagam notificações a todos os participantes. Em outras palavras, todos os participantes tem a mesma visualização do projeto. Já no modo visualmente desacoplado, somente os dados de projeto são armazenados no repositório, a visualização é particular de cada projetista e fica armazena localmente nas ferramentas. Com isso, o mesmo projeto pode ter diferentes visualização.

Neste estudo de caso, foi implementado somente o modo visualmente acoplado, mas testes estão sendo realizados para validar a cooperação através do modo visualmente desacoplado.

6.5 Resumo do Capítulo

Este capítulo mostrou a cooperação do modelo proposto, integrado em uma ferramenta que trabalha com componentes gráficos, representado nesse caso pelo *Blade*, um editor de esquemáticos hierárquicos. Foi apresentada também a integração do módulo de cooperação com a ferramenta *Blade*, detalhando-se quais as alterações que devem ser realizadas para que a ferramenta utilize esse serviço.

Nesse particular, a atuação do *Collaborative Service* pode ser descrita como genérica, pois independe do modelo de dados imposto por cada uma das ferramentas, em especial a ferramenta *Blade*. A cooperação através dos níveis hierárquicos mostra que o módulo pode se adaptar a qualquer caso, pois a hierarquia da ferramenta *Blade* não impede a sua utilização, visto que todo o estado da hierarquia é armazenado no repositório de dados e tratado como um objeto.

O capítulo destacou também alguns pontos importantes do modelo, tais como a interface gráfica para o tratamento da percepção dos projetistas, sua comunicação entre os blocos de projeto, quais os projetistas que participam do projeto. Esses mecanismos evidenciam uma fácil interação entre o grupo de projetistas envolvidos no projeto.

O próximo capítulo mostra a mesma interação de projetistas em um bloco de projeto, mas através da ferramenta Homero, um editor de descrições textuais. Esse caso descreve o mesmo tratamento de armazenamento, notificação e integração utilizado neste estudo de caso.

7 Estudo de Caso 2: Homero, Um Editor de Descrições Textuais voltado a programação VHDL, Linguagem C e Verilog

7.1 Introdução

Este capítulo mostra o *Collaborative Service* trabalhando em conjunto com a ferramenta Homero, um editor de descrições textuais, utilizado para descrever códigos em VHDL, Verilog e Linguagem C, proposto por Hernandez e Sawicki [HER 2001]. Homero, assim como *Blade,* é uma das ferramentas que compõem o *Framework Cave*.

Este capítulo demonstra a integração do *Collaborative Service* com a ferramenta Homero, descrevendo a cooperação de projetistas em um bloco de projeto. Além disso, este segundo estudo de caso traz a atuação do módulo de cooperação sob uma ferramenta textual, comprovando sua flexibilidade no tratamento, tanto em ferramentas que processam a entrada de componentes gráficos (esquemáticos, *layout*), quanto em ferramentas que descrevem seus projetos através de descrições textuais (VHDL, Verilog).

7.1.1 Características da Ferramenta

A ferramenta Homero é um editor de descrições textuais implementado em Java. Seu projeto engloba analisadores sintáticos das linguagens VHDL, Verilog e C, permitindo que o projetista escreva seu código fonte e localize os erros antes de repassar o código para outras ferramentas.

Homero possibilita a visualização de cores diferentes para as palavras reservadas e estruturas sintáticas, além de funcionalidades padrão encontradas em editores normais, como copiar, colar, salvar, abrir, etc. Essa ferramenta, assim como *Blade*, segue o mesmo modelo de programação utilizado no *Framework Cave*. Com isso, as alterações no código, bem como a inserção de novas ferramentas, facilitam o trabalho de seus desenvolvedores.

7.2 Integração da ferramenta Homero com o *Collaborative Service*

Como visto na seção 6.2, a estrutura do Ambiente Cave padroniza a criação de algumas classes com o intuito de explorar a legibilidade e, conseqüentemente, a facilidade de reuso desse modelo. Assim, para integrar o módulo de cooperação com a ferramenta, é necessário modificar a classe responsável pela manipulação de eventos, representada na ferramenta Homero pela classe *HomeroHandler*.

A modificação é realizada da mesma forma que na ferramenta *Blade*: deve-se implementar a interface *DocumentListener* na classe *HomeroHandler* e adicionar os três métodos estáticos *insertUpdate(), changeUpdate(), removeUpdate()*. Abaixo está ilustrada a modificação na classe *HomeroHandler*.

```
import cave.collaborative.connect.*;
import cave.collaborative.chat.*;
import cave.collaborative.service.*;

public class HomeroHandler … implements … DocumentListener
{
…
   public void insertUpdate(DocumentEvent e) {}
   public void changedUpdate(DocumentEvent e){}
   public void removeUpdate(DocumentEvent e) {}
…
}
```

O método *insertUpdate()* é o método responsável por receber todas as notificações do grupo de projetistas envolvidos no projeto. Os métodos *changeUpdate() e removeUpdate()* devem ser somente declarados, pois compõem a interface *DocumentListener*.

A alteração no método *insertUpdate()* consiste em chamar novos métodos que tratam da interação da ferramenta com o módulo de cooperação. Esses métodos pertencem à interface de comunicação do *Collaborative Service* representada pela classe *CollaborativeExtendService*. A função de cada um dos métodos foi detalhada na sessão 6.2.

```
public void insertUpdate( DocumentEvent e )
{
…
   collaboration.getNotificationProject()…
…
   collaboration.getProject()…
…
   collaboration.getNotificationParticipant()…
…
   collaboration.getParticipant()…
…
   collaboration.getWritePermission()…
…
}
```

Assim como na ferramenta *Blade*, o menu de cooperação é inserido através da linha de comando: *CollaborativeExtendMenu.**createMenu**(menubar,this)*. Esse menu disponibiliza todas a opções do módulo de cooperação tais como criar novos projetos, abrir projetos já existentes, atualizar projetos, requisitar permissão, executar c*hat* de

texto, chamar o *help on-line*, etc. Cada uma dessas opções foram apresentadas com mais detalhes na sessão 6.2.

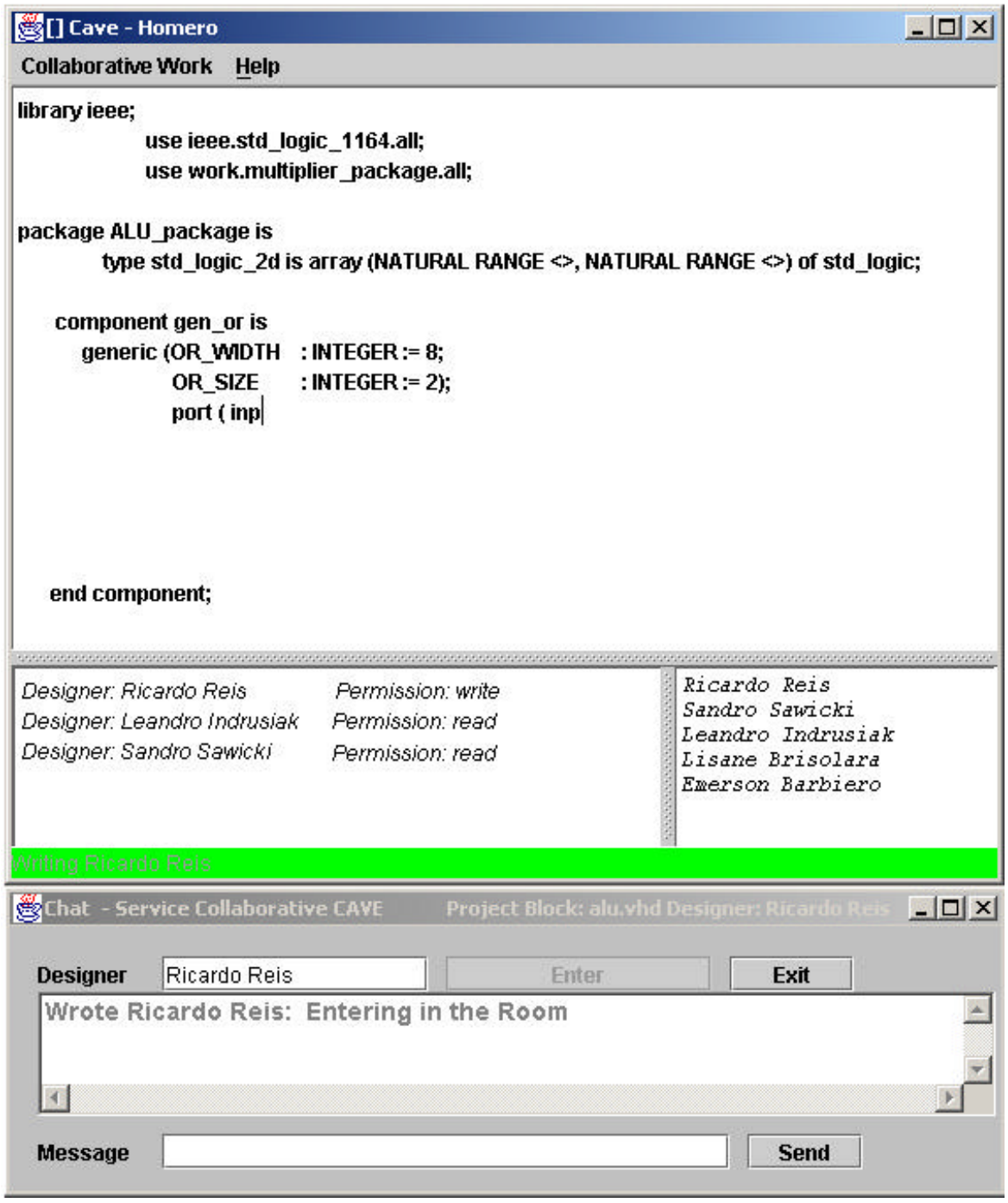


FIGURA 7. 1 – Ferramenta Homero após a inserção dos componentes de cooperação

A figura 7.1 mostra o módulo de cooperação integrado à ferramenta Homero. Os pontos de cooperação a destacar são: (1) o menu de cooperação, chamado de

Collaborative Work, encarregado de disponibilizar as opções de interação da ferramenta com os projetos e projetistas; (2) os componentes que possibilitam a percepção do grupo de participantes *on-line* e suas permissões dentro do projeto; (3) as cores na barra de *status* que representam as permissões de escrita/leitura, bem como o nome do projetista escritor; (4) o componente que mostra o grupo de projetistas envolvidos no bloco de projeto; (5) e o *chat* de texto, utilizado para a comunicação entre o grupo de projetistas.

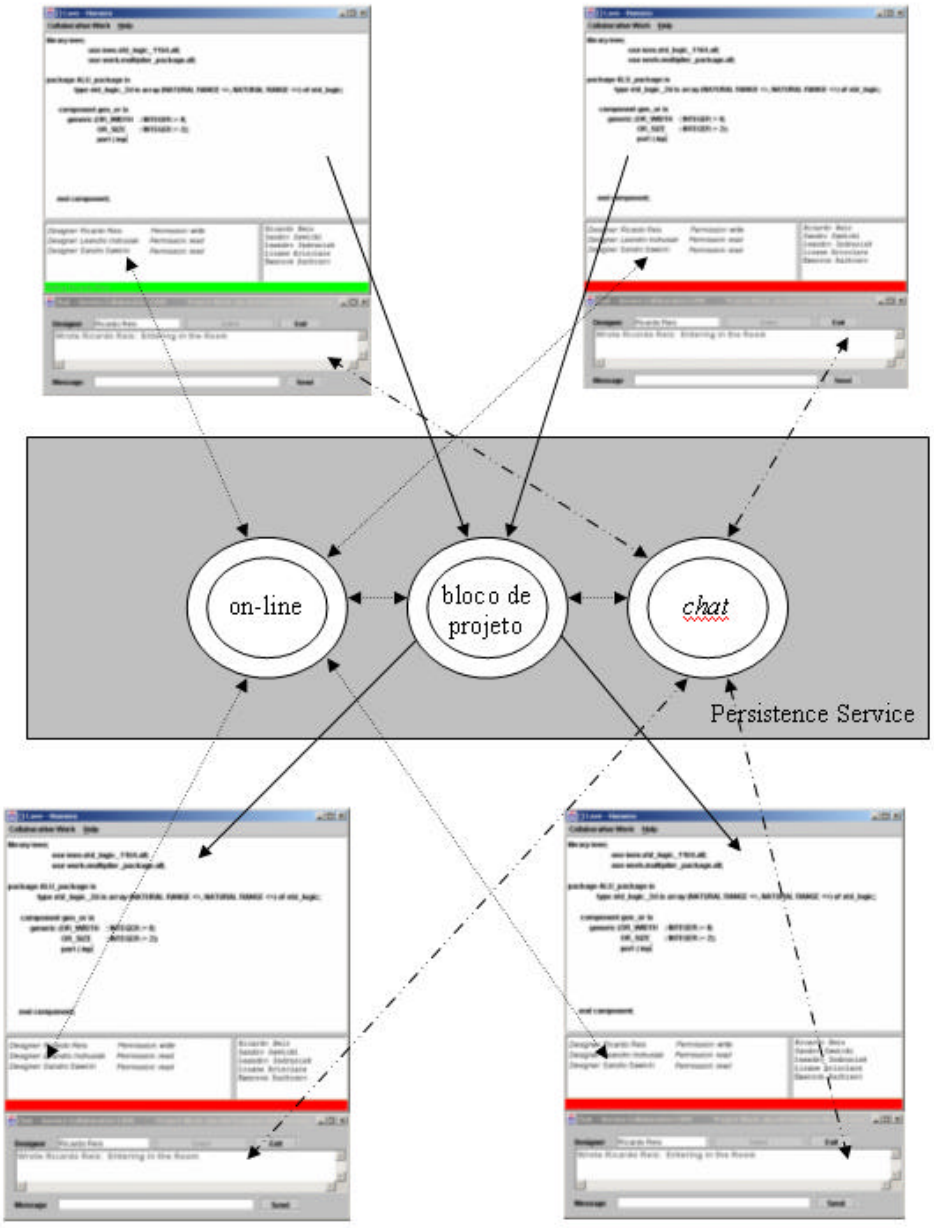


FIGURA 7. 2 – Objetos envolvidos na cooperação

A figura 7.2 mostra a estrutura de objetos armazenada no repositório de dados *(Persistence Service)*. Descrito na sessão 5.5.7, cada bloco de projeto, interage com

dois outros objetos (percepção e *chat*). Nesse exemplo quatro instâncias da ferramenta Homero estão interagindo sob o mesmo bloco de projeto e, conseqüentemente, trocam informações com os demais objetos relacionados com esse bloco.

A execução do exemplo anterior pode ser visualizada na figura 7.3. Percebe-se nessa figura que todas as instâncias da ferramenta visualizam o mesmo bloco de projeto. Esse exemplo foi executado em apenas uma máquina para fins de ilustração, mas pode ser executado em diferentes plataformas de *hardware/software.*

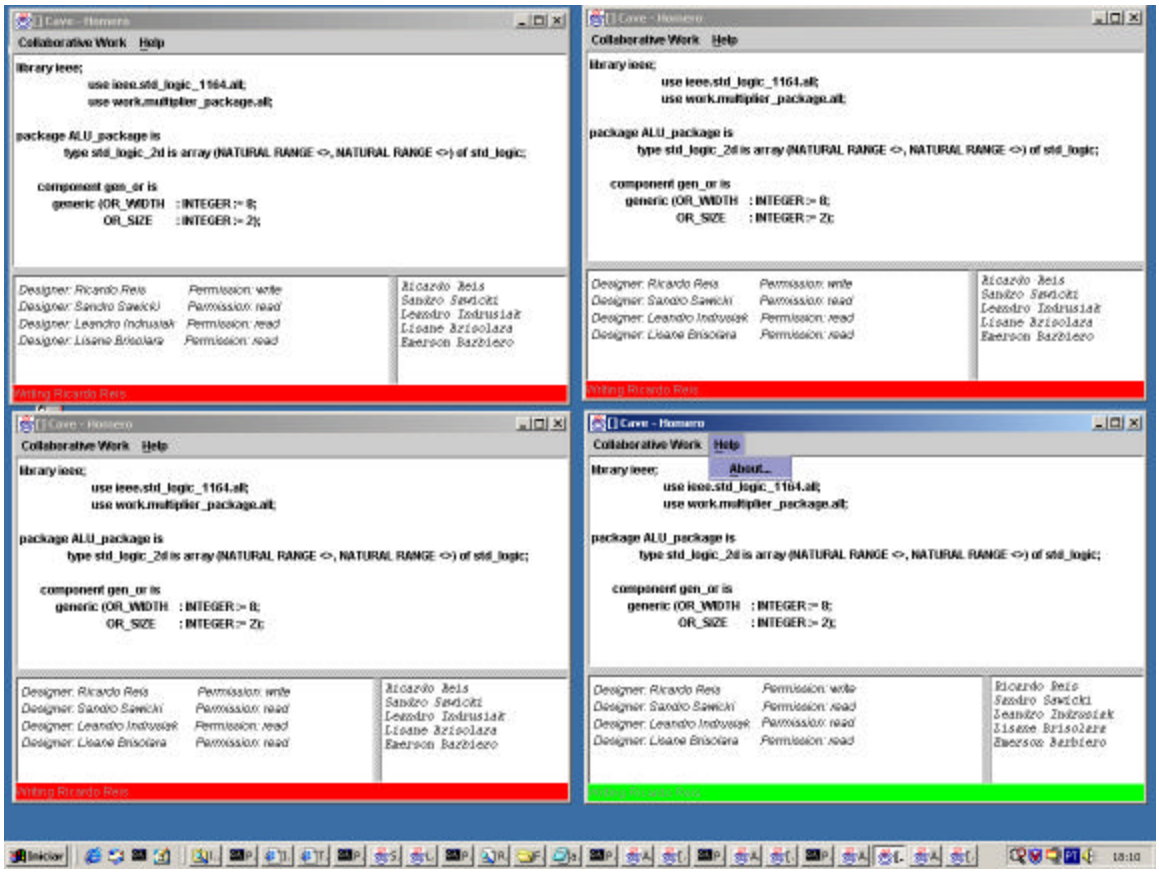


FIGURA 7. 3 – Interação de quatro instâncias da ferramenta Homero sob um bloco VHDL

A utilização do serviço de cooperação não altera a funcionalidade da ferramenta. Com isso, recursos executados em máquinas locais, tais como salvar localmente, abrir arquivo local, compilar podem ser executados normalmente, mesmo conectados ao serviço de cooperação.

A especificação da abertura e criação de novos projetos foi descrita com mais detalhes na seção 6.2, figuras 6.3 e 6.4 respectivamente. Essas figuras ilustraram a interface gráfica padrão utilizada pelos projetistas para se trabalhar com o módulo de

cooperação, representando a inserção de blocos de projeto, de projetistas participantes e de senhas de acesso ao projeto.

A figura 7.4 representa o início do projeto de uma ALU, descrita em linguagem VHDL. Nessa ilustração, é possível perceber a representação inicial do código, visualizada da mesma forma nas duas instâncias da ferramenta.

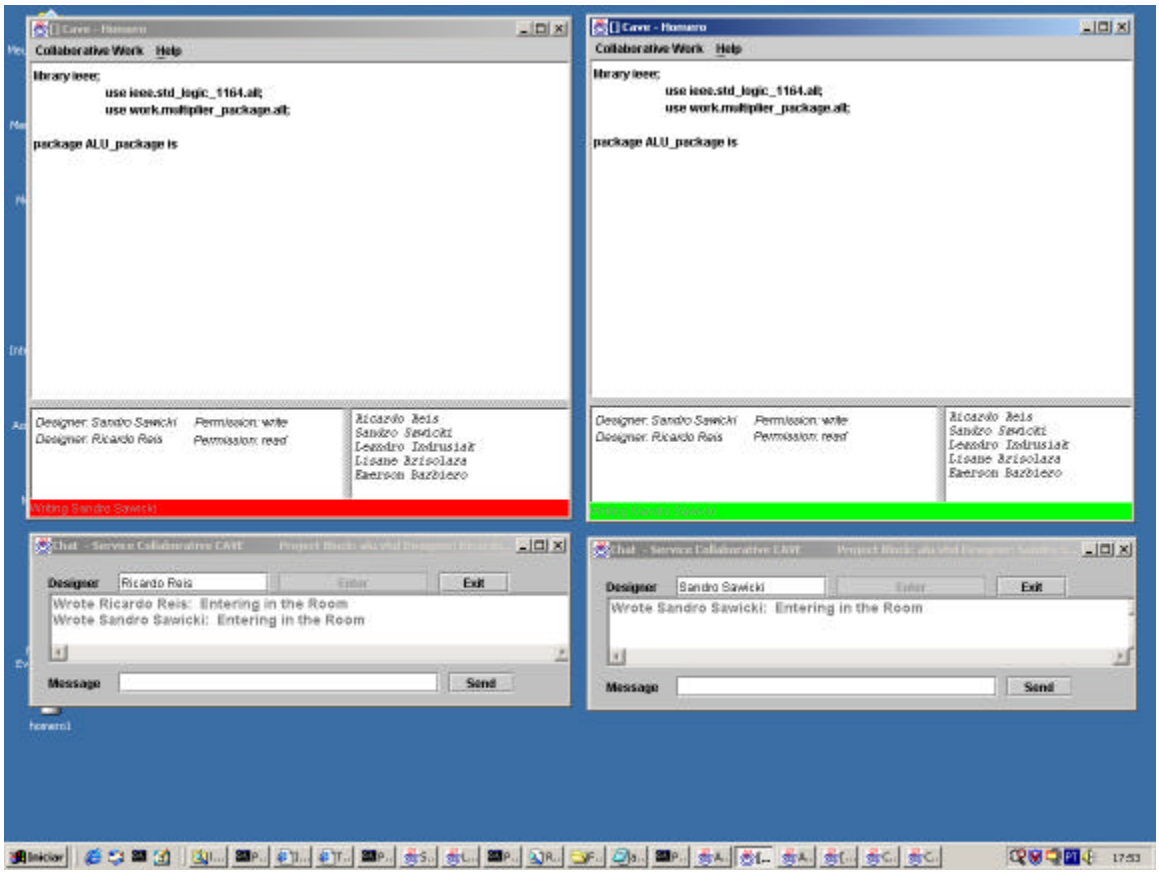


FIGURA 7. 4 – Interação sob um bloco VHDL

A utilização da ferramenta Homero como objeto de estudo de caso mostra a flexibilidade e a fácil adaptabilidade do módulo de cooperação. A execução do módulo de cooperação independe do formato de saída da ferramenta, isso pode ser comprovado através dos dois estudos de caso representados pelas ferramentas *Blade* (esquemáticos) e Homero (VHDL).

Todos os recursos referentes à cooperação são transferidos para as ferramentas que utilizam o *Collaborative Service*, são elas a percepção, o *chat* de texto, o menu de cooperação, entre outros, todos adaptados na ferramenta e executados através da rede.

A seguir, a figura 7.5 traz o projeto da ALU descrito anteriormente, no qual modificações foram realizadas pelo projetista escritor e repassado a projetista ouvinte.

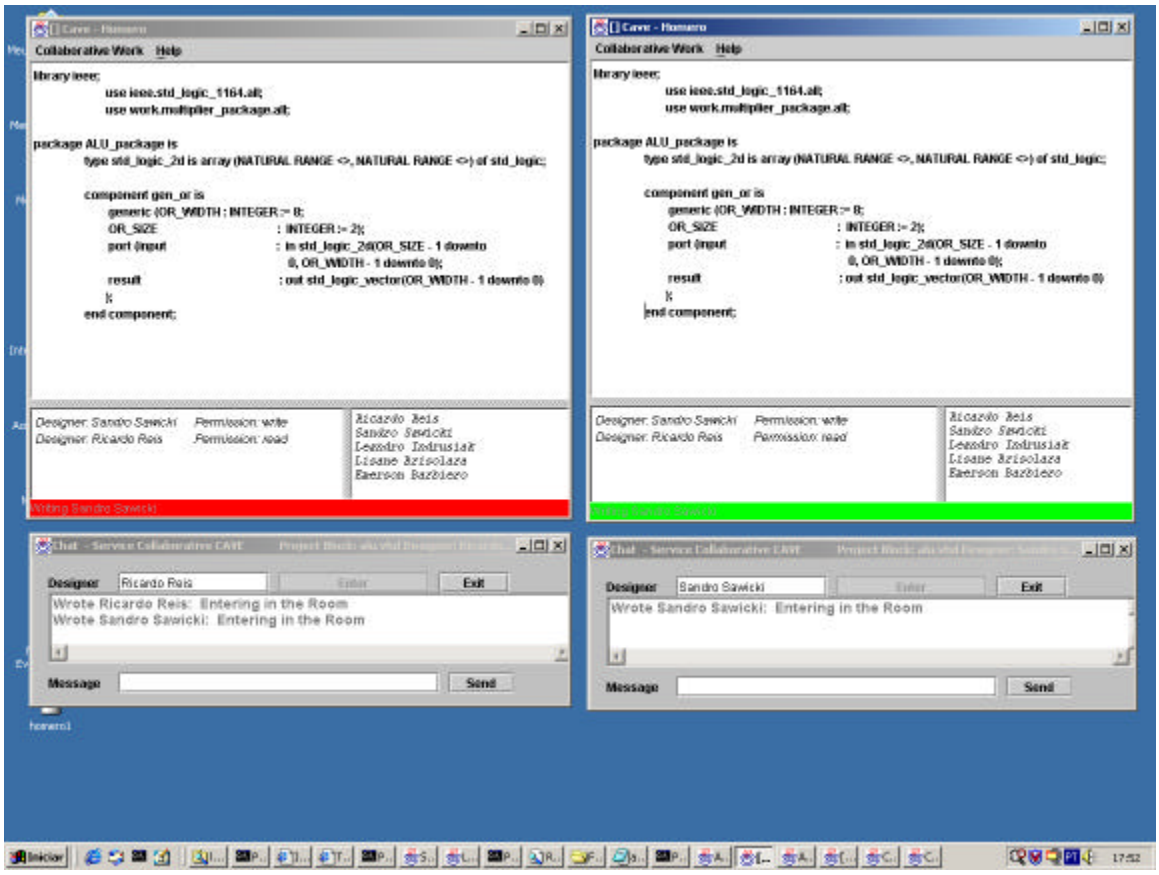


FIGURA 7. 5 – Projeto de uma ALU em VHDL

7.3 Resumo do capítulo

Esse capítulo mostrou a integração da ferramenta Homero, um editor de descrições textuais com o *Collabortive Service*. Esse estudo de caso pretendeu demonstrar a flexibilidade e a adaptabilidade do módulo de cooperação interagindo e cooperando sob ferramentas com interface e saída textual. A utilização da ferramenta Homero como segundo estudo de caso comprova que o modelo de dados empregado na implementação do *Collaborative Service* pode interagir em qualquer formato de saída imposto pelas duas ferramentas do Ambiente Cave (Homero - textuais ou *Blade* - gráficas). Esse aspecto qualifica o *Collaborative Service* como um poderoso recurso do Ambiente Cave para a realização de projetos cooperativos.

Conclusão

Este trabalho apresentou uma proposta de suporte a cooperação que foi inserido no Ambiente Cave. Esse suporte foi especificado e implementado como um serviço e pode ser requisitado pelas ferramentas do ambiente, permitindo o compartilhamento de dados de projeto e a troca de experiências entre os projetistas.

Neste contexto, apresentou-se uma revisão bibliográfica envolvendo o Ambiente Cave, o qual foi o foco central para a criação do modelo cooperativo. Percebeu-se que a evolução desse ambiente impulsionou o desenvolvimento do modelo de cooperação proposto nesse trabalho.

Outros aspectos relativos à especificação de um módulo que suporte o trabalho em grupo foram levantados, tais como tecnologias de suporte e funcionalidades que servem de apoio às atividades em grupo, como a comunicação, negociação, percepção, coordenação e compartilhamento. Nesse sentido, a pesquisa relativa à essência do trabalho em grupo foi explorada. Com isso, o *Collaborative Service* surge através do estudo de conceitos inseridos dentro de seu domínio de aplicação.

É possível destacar alguns aspectos relevantes desse modelo, como a possibilidade de interação entre dois ou mais projetistas (mesmo distantes geograficamente) em um único bloco de projeto; possibilidade de execução desse modelo em diferentes plataformas de *hardware/software*; capacidade de suportar ferramentas de CAD, atuais e futuras, do Ambiente Cave; disponibilidade de um mecanismo de comunicação entre os projetistas envolvidos no mesmo projeto; acesso remoto a dados de projeto, fácil adaptação nas ferramentas, mecanismo de percepção de projetistas; entre outros.

Para que fosse possível validar essas características, necessitou-se especificar e implementar a arquitetura *Service Space*, detalhada anteriormente neste texto. Essa arquitetura é composta por serviços que são requisitados pelas ferramentas do Ambiente Cave, entre eles: serviço de transações, serviço de persistência, serviço de cooperação. A autenticação do usuário que requisita uma sessão cooperativa foi implementada junto ao serviço de cooperação. Buscou-se formar uma arquitetura baseada em serviços, focando unicamente na modularidade do modelo. Em outras palavras, os serviços foram implementados de forma independente (alguns adaptados), mas com possibilidade de interação entre eles.

Como base para a implementação desse modelo, foram utilizadas as tecnologias *Java, Jini e Javaspaces*. Java, é uma linguagem que oferece além da possibilidade da criação de programas orientados a objeto, uma característica chave para o desenvolvimento desse modelo, a portabilidade. Essa característica tornou possível a execução do *Collaborative Service* em diferentes plataformas de *hardware/software*. Já a tecnologia *Jini*, além de disponibilizar vários recursos, permite a reutilização do seu serviço de transações *(Transaction Service)* e persistência *(Javaspaces)*. Esses serviços

foram adaptados para trabalharem em conjunto com o *Collaborative Service* e, atualmente, compõem o *Service Space*.

Muitas características citadas neste trabalho ainda podem ser exploradas a fim de que aumentem cada vez mais a eficiência do apoio ao projetista de circuitos integrados, no contexto do trabalho em grupo. Muito trabalho ainda precisa ser pesquisado e executado, por isso deve-se considerar o presente texto como ponto de partida.

7.4 Contribuições do Trabalho

A implementação do modelo de cooperação incorporou ao Cave um novo recurso no projeto de circuitos integrados. Mas, para que esse módulo fosse validado, houve a necessidade de se implementar uma arquitetura de suporte, chamada *Service Space*. Com isso, esse modelo produziu um conjunto de opções que engloba os objetivos estabelecidos neste trabalho. Tais características são descritas a seguir:

?? **Compartilhamento de dados de projeto:** através desse mecanismo o projetista pode armazenar seus projetos em um repositório único, com possibilidade de disponibilizá-lo a mais de um projetista. Permite também, a recuperação remota dos dados de projeto, bastando apenas requisitar o serviço de cooperação através de uma das ferramentas do Ambiente Cave.

?? **Acesso restrito ao projeto:** cada projeto utiliza uma senha de acesso aos dados. Com isso, somente o grupo (caso envolva mais pessoas) envolvido no projeto tem condições de manipular esses dados. Um ponto importante é que com a senha do projeto, o projetista pode navegar por todos os blocos envolvidos.

?? **Mecanismo de comunicação:** a comunicação entre os projetistas é representada através de um *chat* de texto. Esse mecanismo pode ser requisitado logo após a comunicação com o módulo de cooperação. Com isso, todos os blocos podem utilizar esse recurso com a finalidade de expressar as idéias do projeto. Pesquisas estão sendo realizadas pelo GME (Grupo de Microeletrônica da UFRGS) a fim de incorporar áudio ao modelo textual do *chat*.

?? **Coordenação de projeto:** para cada projeto existe um coordenador (chefe) encarregado de gerenciar as atividades do grupo. Sua função principal está na criação de novos projetos, inclusão ou exclusão de participantes. Assim como os demais participantes, o projetista coordenador tem transito livre por todos os blocos do projeto.

?? **Modo de percepção:** é um ponto de extrema importância dentro de um ambiente cooperativo. Essa característica permite que todos os projetistas envolvidos em um bloco de projeto consigam perceber quem está trabalhando no projeto num

determinado momento. Nesse modelo, essa opção identifica também as permissões de escrita ou leitura adquiridas pelo grupo de projetistas.

?? **Suporte genérico:** essa característica faz com que o módulo de cooperação suporte as ferramentas atuais, e futuras, do ambiente Cave. Permite a cooperação em nível de descrições gráficas e textuais. É facilmente adaptável a qualquer ferramenta CAD que segue o padrão de desenvolvimento do *framework* Cave.

?? **Independência de plataforma:** sendo baseado na linguagem Java, toda a arquitetura pode rodar, ou ser acessada em diferentes plataformas de *hardware/software*. Nesse sentido, a portabilidade, que é um dos aspectos principais do Ambiente Cave, continua atuante, mesmo com a inserção desse novo módulo.

?? **Metodologia de cooperação:** um dos aspectos importantes em uma arquitetura cooperativa é a definição de sua metodologia. O *Collaborative Service* tem essa definição clara através da extensão da metodologia *Pair-Programming* (máquinas locais), para *Paar-Programming* (máquinas remotas). Essa definição facilita o projetista a decidir qual metodologia se adapta melhor ao seu problema.

?? **Integração com hiperdocumentos de suporte:** o *framework* Cave, assim como o módulo de cooperação podem ser apoiados por hiperdocumentos como *help on-line*, tutoriais, cursos, etc. Assim, o usuário pode alternar de forma simples entre o ambiente de projeto cooperativo e ambientes de instrução. Com isso, além de toda a documentação armazenada no servidor de ambiente, pode-se ligar informações adicionais sobre cada tópico abordado através de *links* de hipertexto.

?? **Distribuição de recursos transparentes ao usuário:** a interface criada pela estrutura do Ambiente Cave e do módulo de cooperação permite ocultar a localização geográfica dos recursos do ambiente distribuído entre as máquinas da rede. Além desse aspecto, o usuário do módulo de cooperação não precisa se preocupar em como os serviços tratam suas interações. Isso é realizado de forma transparente, sendo necessário somente a comunicação com o serviço de cooperação.

7.5 Trabalhos Futuros

Ao término desse trabalho percebeu-se que a arquitetura que agrega o módulo de cooperação poderia ser estendida, visando o acréscimo de novos serviços, descritos a seguir.

?? recursos de modelagem de fluxo de projeto devem ser estudados e implementados, baseando-se na evolução desse ambiente. Nesse caso, o fluxo de projeto pode ser modelado como um novo serviço, com a função de ligar e gerenciar uma etapa (ferramenta) às etapas subseqüentes e precedentes.

?? como o sistema é essencialmente distribuído, com ferramentas e dados trafegando entre servidor do ambiente e máquinas cliente, o gerenciamento de versões é um

ponto crítico. Nesse caso, pode-se criar um módulo capaz de gerenciar as versões dos dados de projeto. No atual modelo, os dados não são versionados, porém, podem ser armazenados localmente.

?? novas implementações desse modelo devem ser estudadas a fim de verificar uma possível integração com banco de dados. Assim, o espaço compartilhado de objetos pode ser o local ao qual dá-se a cooperação. Ao final da sessão cooperativa, os dados de projeto podem ser armazenados em uma estrutura de banco de dados.

?? a tomada de decisões em um ambiente de cooperação é um aspecto de extrema importância. Nesse sentido, mecanismos de geração de idéias; organização de idéias e votação são necessários para que um projeto adquira maturidade suficiente na organização de suas decisões.

?? o modelo proposto pode incorporar mais de uma metodologia de cooperação. Com isso, o projetista pode escolher qual a metodologia que se adapta ao seu objetivo. Nesse sentido, novas metodologias de cooperação devem ser estudadas e incorporadas ao *Collaborative Service.*

?? pesquisas estão sendo realizadas pelo GME (Grupo de Microeletrônica da UFRGS) com o objetivo de incorporar ao *chat* de texto um recurso de áudio. Os resultados atualmente correspondem ao armazenamento local. Porém, estudos estão sendo desenvolvidos a fim de possibilitar o acesso remoto a esse áudio através da atual interface do *chat*.

A figura 8.1 (a) a seguir, ilustra o modelo do *framework* Cave original, instanciando ferramentas e componentes de projeto utilizados por projetistas de CIs. A figura 8.1 (b), mostra a contribuição desse trabalho com a implementação do *service space* através das tecnologias *Java, Jini e Javaspaces*, vislumbrando a alteração do modelo existente através da criação de novos serviços. Esses serviços correspondem ao tratamento de versões, tomada de decisões, fluxo de projetos, além de uma base de dados relacionada com o serviço de persistência, a fim de armazenar o fechamento das sessões cooperativas.

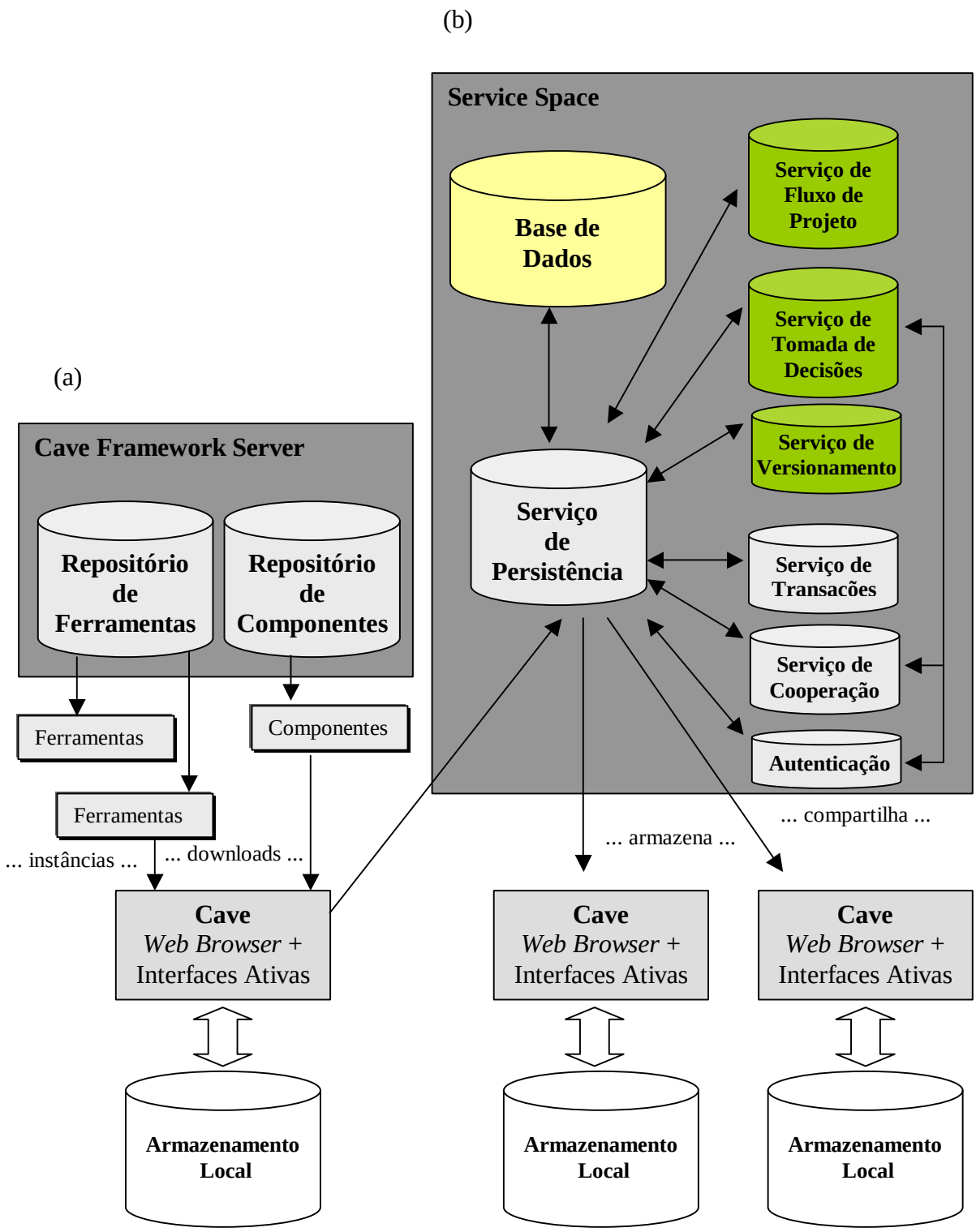


Figura 8. 1 – Abrangência e trabalhos futuros

Anexo 1: Tutorial de Inicialização das Tecnologias *Jini/JavaSpaces*

Introdução

Esse tutorial é baseado em experiências práticas na utilização da tecnologia *Jini* e de alguns de seus serviços, tais como: Javaspaces e Transações. Ele está disponível via WWW em http://www.inf.ufrgs.br/~sawicki/Inicializando-Jini-JS.html. Abaixo são demonstradas algumas características principais que possibilitam a execução dessas tecnologias.

Instalação dos Pacotes Jini

Todos os pacotes citados abaixo podem ser encontrados na página da *Sun Microsystems*, no endereço http://java.sun.com. Os exemplos desse apêndice, partem do princípio que o JDK *(Java Development Kit)* já está instalado na máquina. Nesse tutorial é utilizado como exemplo o JDK 1.3. Para começar a trabalhar com *Javaspaces*, ou qualquer outro serviço, é preciso em primeiro lugar instalar o pacote que contém a tecnologia *Jini*. Atualmente, a tecnologia *Jini* dispõe da versão 1.1.2, porém nos exemplos a seguir utiliza-se a versão 1.1.

Os arquivos JINI-1.1-G-CS.zip e JINITCK-1.1A-G-CS.zip compõem o pacote de classes. Com isso é necessário que se extraia o conteúdo para um diretório de sua escolha (preferencialmente dentro da pasta do JDK. Isso minimiza os possíveis problemas com o PATH dos arquivos).

A interface gráfica do Jini

Para inicializar a tecnologia Jini, existem duas maneiras: (a) uma através de linha de comando; (b) através de sua interface gráfica. É recomendado a inicialização através da interface gráfica, que é bem mais prática e simples. A inicialização da interface gráfica é descrita na linha de comando abaixo:

```
java -cp C:\<JDK_DIR>\jini1_1\lib\jini-ext.jar;C:\<JDK_DIR>\jini1_1\lib\jini-
examples.jar com.sun.jini.example.launcher.StartService
```

OBSERVAÇÃO: Substitua <JDK_DIR> pelo diretório onde Jini está instalado. Copie essa linha de comando para dentro de um arquivo .bat e o coloque dentro da pasta /bin do JDK. Esse procedimento é recomendado, pois diminui possíveis problemas referentes a *paths*. Cada vez que a interface gráfica é executada, o Jini cria a pasta *log* no diretório onde está localizado o arquivo *bat*. Esta pasta sempre deve ser apagada antes de cada nova execução.

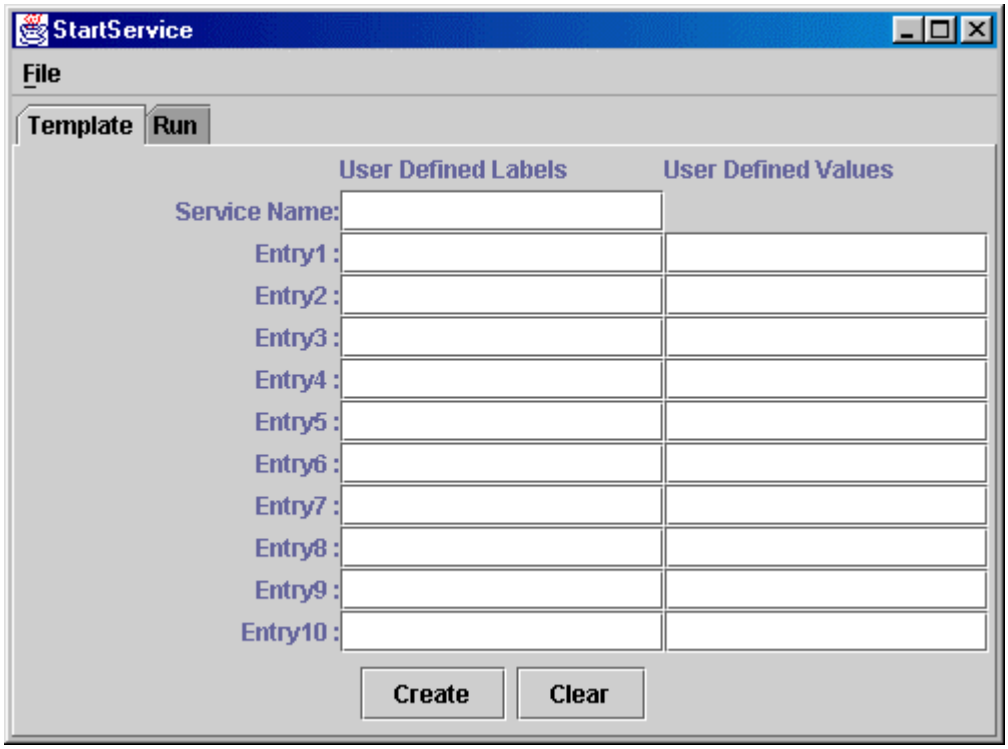


FIGURA 1 - Interface gráfica *Jini*

Configuração e inicialização dos serviços Jini

Após inicializada a interface gráfica da tecnologia *Jini*, deve-se iniciar a configuração. Para isso, é necessário carregar o arquivo de propriedades. Ele está localizado em C:\<JDK_DIR>\jini1_1\example\launcher. Este arquivo pode ser aberto, e seus parâmetros modificados na própria interface gráfica. Mas recomenda-se alterá-lo em um editor de texto comum (*notepad* do *windows*, por exemplo), pois muitas vezes, após a alteração dos parâmetros, foram utilizadas as opções **save** e **save as** e as alterações feitas no arquivo foram completamente perdidas.

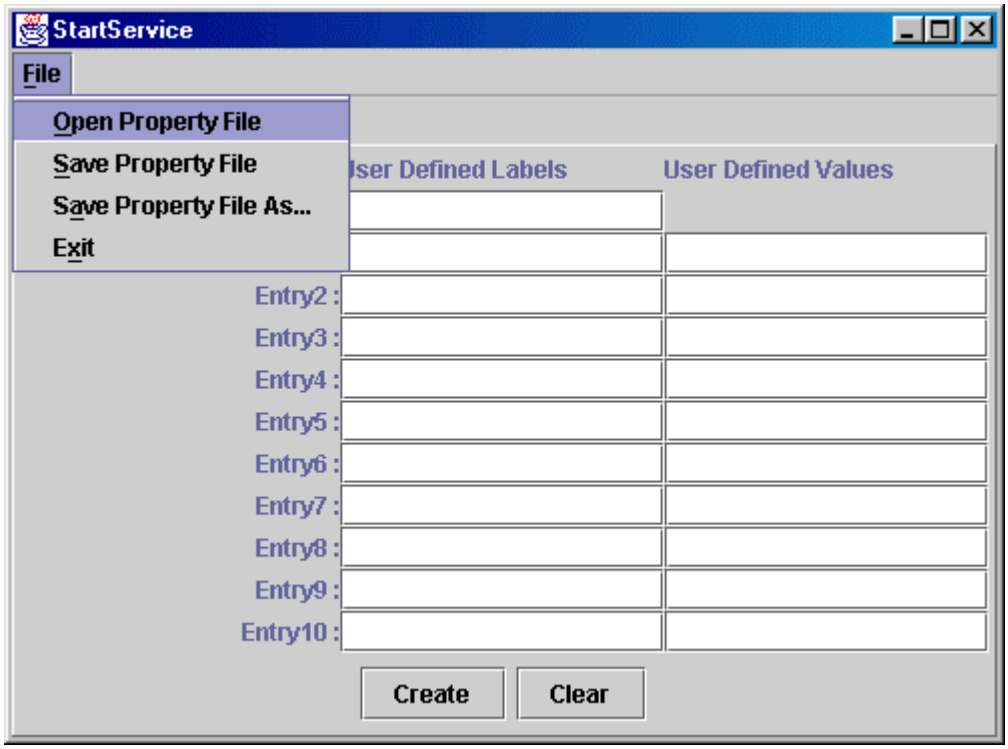


FIGURA 2 - Menu de acesso ao arquivo de configuração

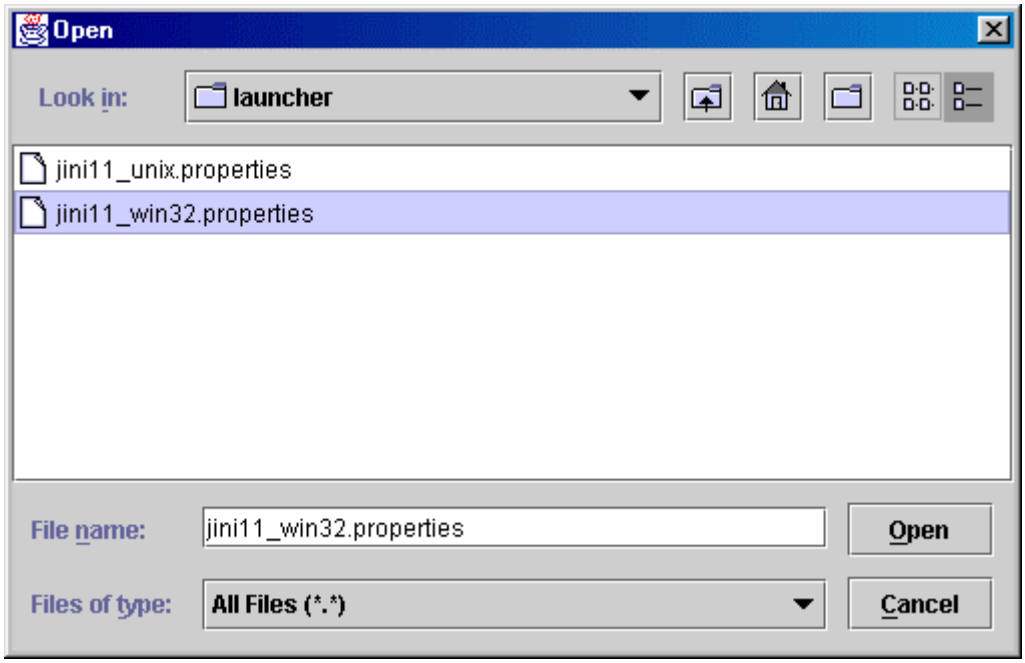


FIGURA 3 - Selecionando o arquivo de configuração para plataforma *windows*

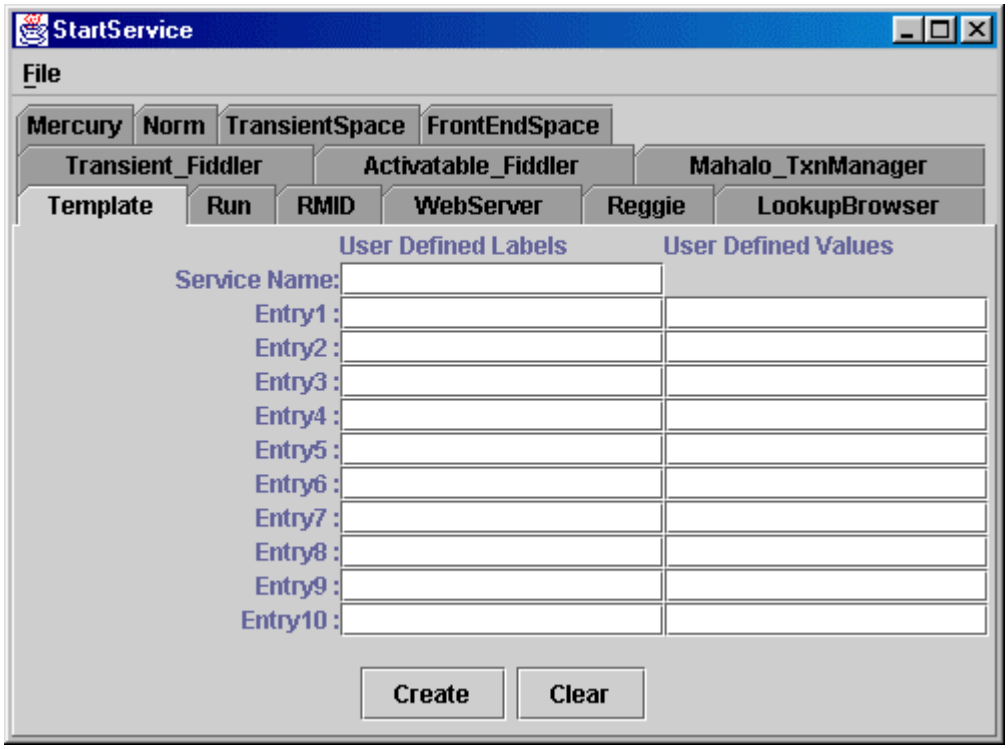


FIGURA 4 - Interface gráfica após abertura do arquivo de configuração

Conforme mostra a figura abaixo, devem ser alterados os seguintes parâmetros:

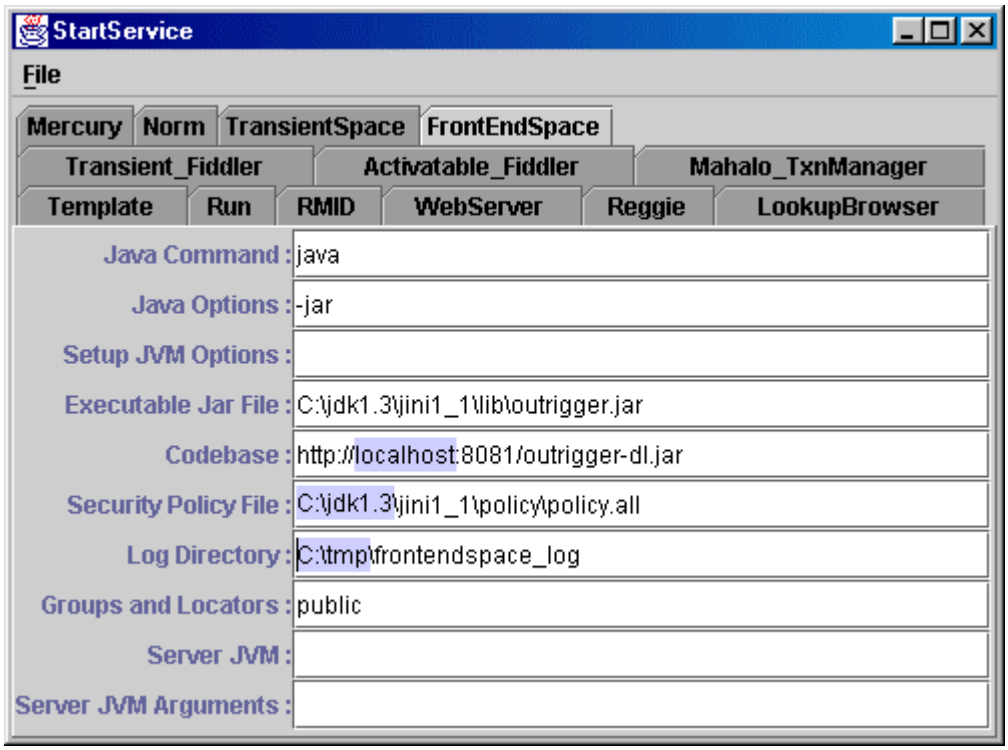


FIGURA 5 - Parâmetros que devem ser modificados

?? Aqueles que indicam o nome da máquina <*hostname*> (opcionalmente pode ser usado o nº IP da máquina);

?? Os que apontam para o diretório onde o *Jini* está instalado <c:\files\jini1_1>;

?? Os que apontam para diretórios onde serão criados os arquivos de *log* <c:\tmp>.

Cada vez que forem inicializados os serviços do *Jini*, serão criados diretórios com os *logs* e estes devem ser obrigatoriamente apagados antes da próxima inicialização. Por *default*, estes arquivos serão criados em c:\tmp.

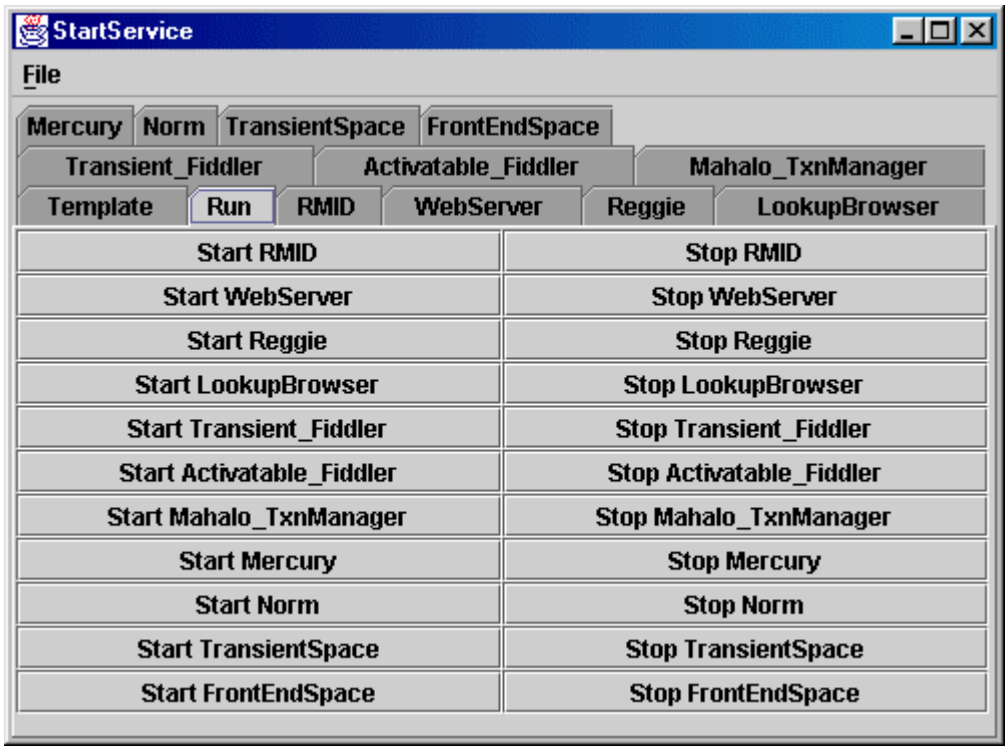


FIGURA 6 – Inicialização dos serviços

Após todos as modificações estarem concluídas, podemos iniciar os serviços Jini. Esses serviços devem ser iniciados na ordem da disposição dos botões da interface (de baixo para cima), e deve-se aguardar o término da execução de cada um deles. Dependendo da capacidade da máquina, a inicialização de alguns serviços é bastante lenta. Neste caso, iniciam-se os seguintes serviços: RMID, *WebServer*, *LookupBrowser*, *FrontEndSpace* (Persistente) / *TransientSpace*.

Ao iniciar o serviço *LookupBrowser*, uma janela será aberta. É nesta janela que serão feitas as últimas configurações para que seja possível rodar um programa utilizando *Javaspaces*.

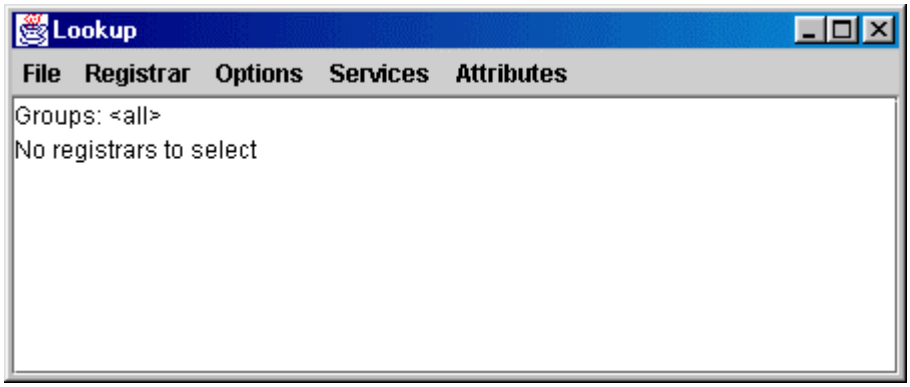


FIGURA 7 - Lookup Browser

Em primeiro lugar deve ser localizado o *host* que dispõe de serviços Jini. Por *default* o *Jini* inicializa os grupos como públicos. Para localizar um *host* público devemos utilizar a opção **File** >> **Find Public**. Dependendo da capacidade da máquina, pode demorar alguns segundos para encontrar o serviço.

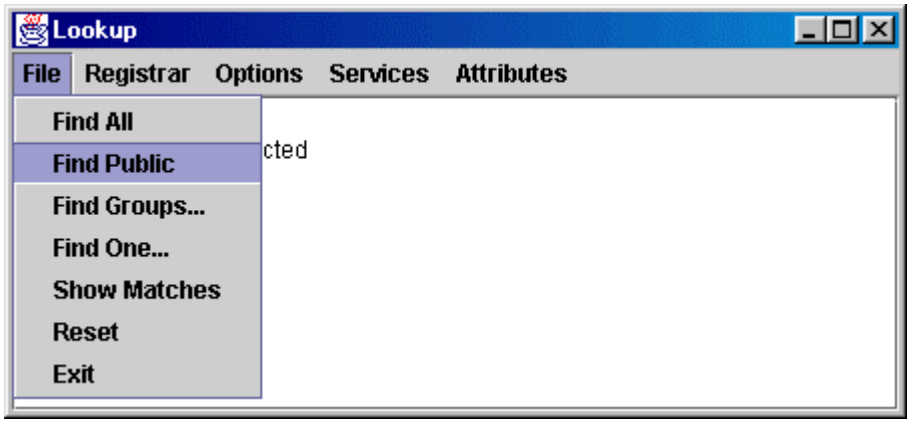


FIGURA 7 - *Lookup Browser*

Na Figura 8 podemos ver a mensagem que aparece na tela quando um *host*/grupo é localizado.

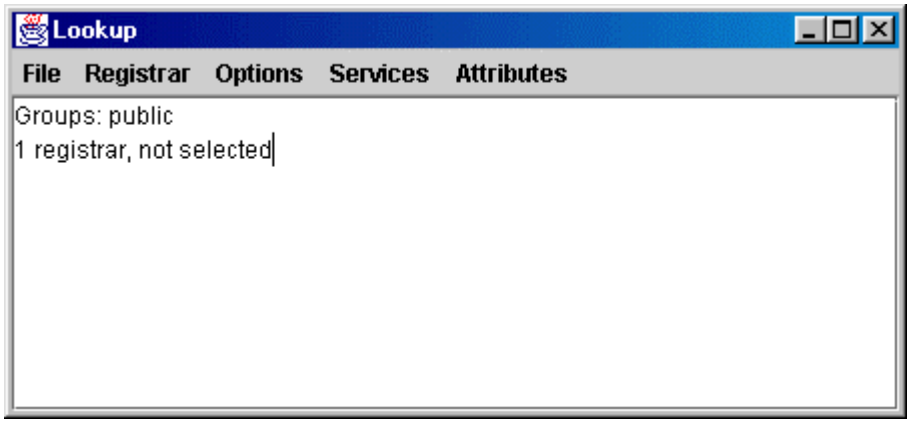


FIGURA 8 - *Lookup Browser* Indicando que um *host* foi localizado

O próximo passo é registrar o *host*. Use a opção **Registrar** >> **<NOME_HOST>**. Isso pode ser visto na Figura 9.

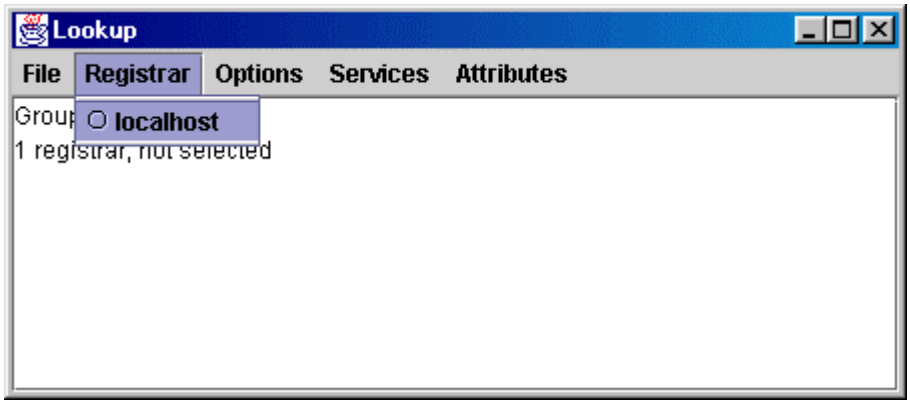


FIGURA 9 – Registrando *host* no *Lookup Browser*

Finalmente, define-se o atributo nome do serviço *Javaspaces*. Usando a opção **Attributes** >> **Name** e selecionando o nome *"Javaspce"*

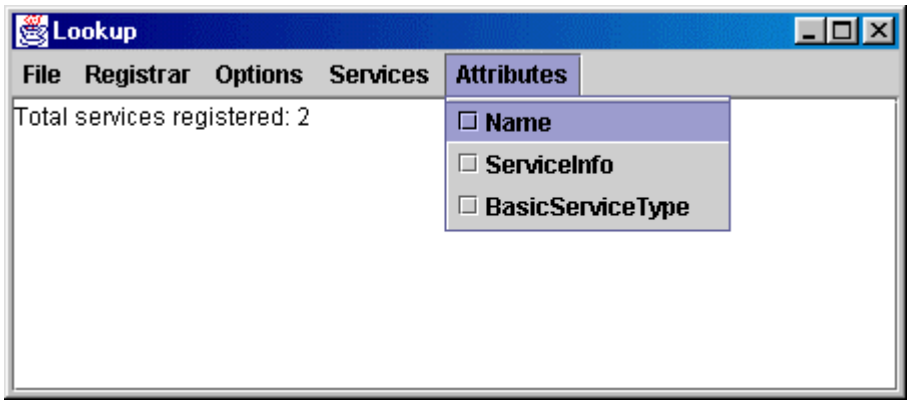


FIGURA 10 - Definindo um atributo

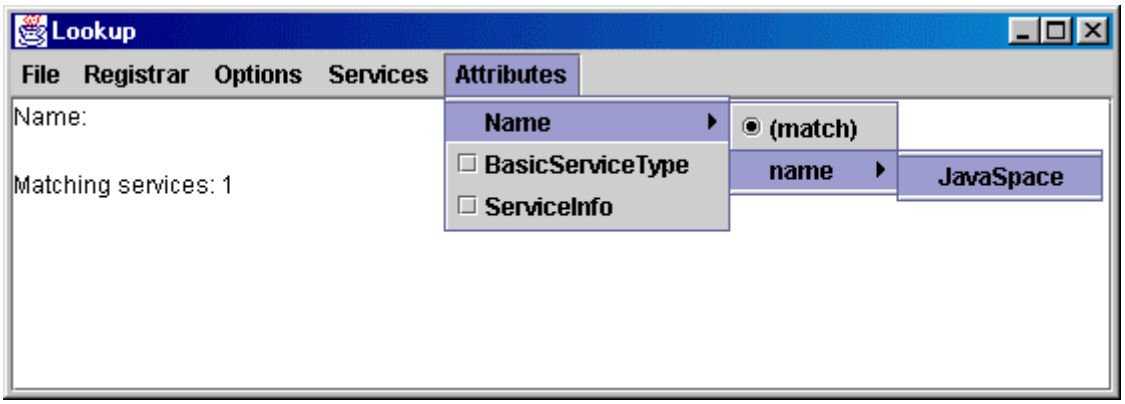


FIGURA 11 – Acrescentando o nome ao atributo

Nesse ponto,o processo está finalizado, o serviço está configurado e rodando corretamente, conforme pode ser visto na figura 12.

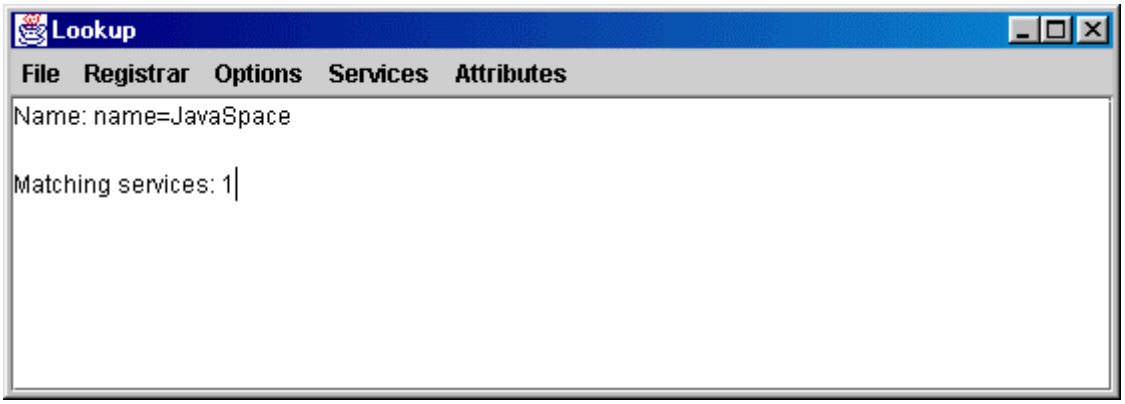


FIGURA 12 - Serviço *Javaspaces* ativado

Dicas para compilar, executar e definir a política de segurança da aplicação

?? **Path:** Deve-se indicar o caminho das bibliotecas *Jini* e JDK.

?? **Compilação:** Acrescentar as seguintes linhas de código:

javac -classpath
c:\<JDK_DIR>\bin;c:\<JDK_DIR>\jini1_1\lib;c:\<JDK_DIR>\jini1_1\lib\mahalo.jar;c:\<
JDK_DIR>\jini1_1\lib\space-examples.jar;
c:\<JDK_DIR>\jini1_1\lib\jini-core.jar;c:\<JDK_DIR>\jini1_1\lib\jini-
ext.jar;c:\<DIR_PROG_JAVA>;c:\<DIR_PROG_JAVA> %1.java

?? **Política de Segurança:** Na execução da sua aplicação Java, deve existir uma referência a um arquivo indicando a política de segurança. Recomendamos que esse arquivo seja colocado no mesmo diretório da sua aplicação. Para fins de testes, utilizamos o arquivo *policy*, que neste caso libera a concessão de uso do recurso para todos que solicitarem.

?? **Execução:** Para a execução da aplicação é necessário utilizar o seguinte caminho:

java -classpath
c:\<JDK_DIR>\bin;c:\<JDK_DIR>\jini1_1\lib;c:\<JDK_DIR>\jini1_1\lib\space-
examples.jar;c:\<JDK_DIR>\jini1_1\lib\jini-core.jar;

c:\<JDK_DIR>\jini1_1\lib\jini-ext.jar;c:\<DIR_PROG_JAVA>\ -
Djava.security.policy=c:\<DIR_PROG_JAVA>\policy %1

O serviço *Javaspaces* já está rodando e por *default* o seu name=JavaSpace, precisa-se, agora, definir o nome da aplicação que seja igual ao nome do *Javaspaces* servidor. Para isso utiliza-se o código fornecido junto com o livro *JavaSpaces Principles,*

Patterns, and Practice, de Eric Freeman, Susanne Hupfer e Ken Arnold (Addison-Wesley, 1999). O download dos exemplos do livro pode ser feito em The Jini™ Technology Series. Ele fornece dois arquivos para facilitar o acesso ao *Javaspaces*: O *SpaceAccessor.java* e o *TransactionManagerAccessor.java*

Para definir o nome do *Javaspaces* no *LookupBrowser*, como foi mostrado anteriormente, deve-se especificá-lo para a aplicação Java. Existem duas formas de fazer isso:

?? Editar o arquivo *SpaceAccessor.java* e alterar a linha 38: return getSpace ("**<NOME_ JAVASPACE>**");

?? Na chamada do método *SpaceAccessor.getSpace* passar como parâmetro o nome do *Javaspaces*.

HelloWorld - Exemplo de programação com JavaSpaces

Veja a seguir um exemplo de uma aplicação Java bastante simples que utiliza o serviço Javaspaces.

Código fonte comentado

Message.class

```
import net.jini.core.entry.Entry;  // Declaração da classe Message, que obrigatoriamente deve implementar Entry
public class Message implements Entry {
   public String content;  // Declaração da string content
   public Message() {  // Declaração do método Message()
   }
}
```

HelloWorld.class
São importadas as classes:

```
import jsbook.util.SpaceAccessor;  // SpaceAccessor (Para localizar o JavaSpace)
import net.jini.core.lease.Lease;  // Lease (Para indicar o tempo de espera)
import net.jini.space.JavaSpace;   // JavaSpace (Para utilizar os objetos do JavaSpace)
public class HelloWorld {  // Declaração da classe HelloWorld
   public static void main(String[] args) {

      try {      // Tratamento de Exceções

         Message msg = new Message();   // Instanciação de uma entry msg
         msg.content = "Hello World";    // content da entry msg é setado com "Hello World"
```

```
        JavaSpace space = SpaceAccessor.getSpace();
        space.write(msg, null, Lease.FOREVER);
        Message template = new Message();   // Para ler a entry é usado um template

        Message result =(Message)space.read
       (template, null, Long.MAX_VALUE);

        System.out.println(result.content);  // Resultado é impresso na tela

      } catch (Exception e) {        // Tratamento de Exceções
        e.printStackTrace();
      }
   }
}
```

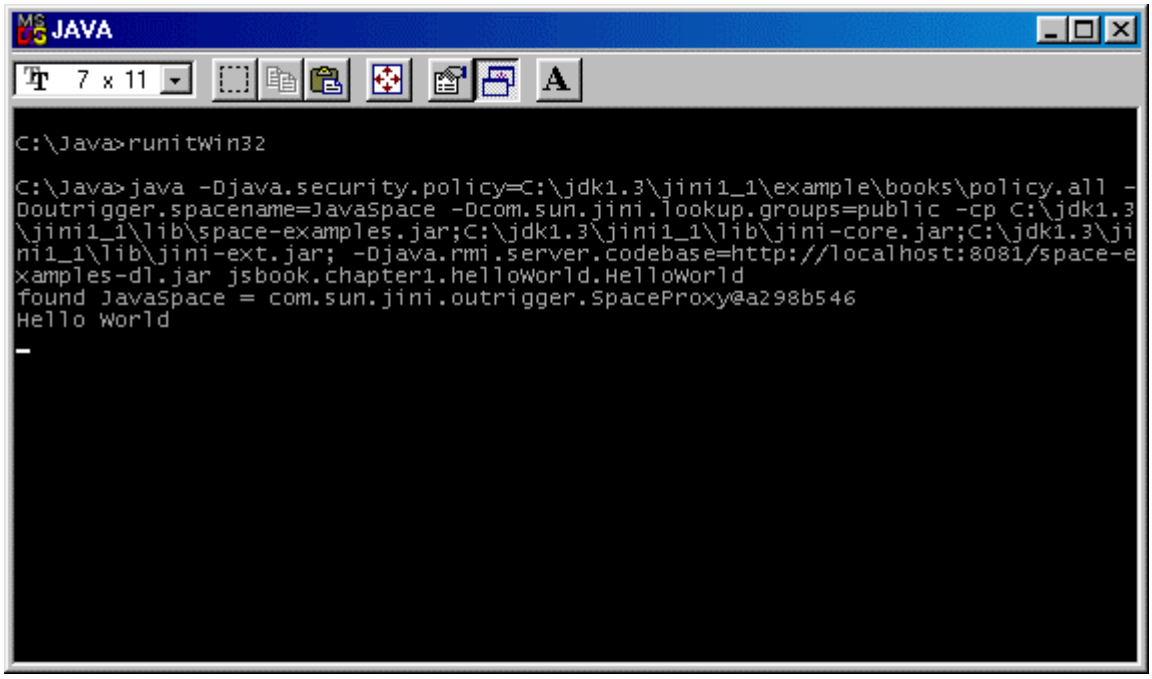


FIGURA 13 - Resultado da execução da aplicação *HelloWorld*

Bibliografia

[BAL 93] BALASUBRAMANIAM, V. **State of the Art Review on Hipermedia Issues And Application.** Newark, NJ: Rutgers University, 1993.

[BAR 92] BARNES, T.J. et al. **Electronic CAD Frameworks.**[S.1.]: Kluwer Academic Publisher, 1992. 196p.

[BAR 94] BARROS, Ligia A. **Suporte a Ambientes Distribuídos para Aprendizagem Cooperativa**. 1994. Tese (Doutorado em Ciência da Computação) - COPPE/UFRJ, Rio de Janeiro.

[BEN 98] BENTLEY, Richard; HORSTMANN, Thilo; SIKKEL, Klass; TREVOR, JOHATHAN. **Supporting Cooperative Information Sharing with the World Wide Web**: The BSCW Shared Workspace System. 1998. Disponível em: <http://www.w3.org/Conferences/WWW4/>. Acesso em: 20 set. 2002.

[BOO 2000] BOOCH, Grady. **UML:** guia do usuário. Rio de Janeiro: Campus, 2000. 472 p.

[BOO 2001] BOOCH, Grady. **Object-Oriented Analysis and Design with Applications**. Redwood City: Addison Wesley Longman, 2001.

[BOR 95] BORGES, Marcos Roberto da Silva; CAVALCANTI, Maria Claudia Reis; CAMPOS, Maria Luiza Machado; LIMA, José Valdeni de. Suporte por Computador ao Trabalho Cooperativo. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 1995. **Anais...** Porto Alegre: SBC: Instituto de Informática, UFRGS, 1995.

[BRG 2001] BRGLEZ, F.; LAVANA H. A Universal Client for Distributed Networked Design and Computing. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings**... New York: ACM, 2001.

[BRI 2001] BRISOLARA, Lisane. B; INDRUSIAK, Leandro S. ; REIS, Ricardo A. L. Developing an Hierarchial Schematic Editor to WWW. In: UFRGS MICROELETRONICS SEMINAR, SIM, 16., 2001, Santa Maria. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2001. p. 49-52, 2001.

[BRI 2002] BRISOLARA, Lisane B.; INDRUSIAK, Leandro S.; REIS, Ricardo A. L. Modelagem Orientada a Objetos de Primitivas de Projeto de Sistemas Eletrônicos voltada para Colaboração. In: WORKSHOP IBERCHIP, 8., Guadalajara. **[Anales]**. Guadalajara: Cinvestav, 2002.

[BRI 2002a] BRISOLARA, Lisane. B. , INDRUSIAK, Leandro S., REIS, Ricardo A. L. Blade: A Hierarchical Diagram Editor Target to Collaboration. In: SOUTH MICROELETRONICS SEMINAR, SIM, 17., 2002, Gramado, **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2002. p. 29-32.

[CAY 2001] CAYRES, Paulo Henrique. **Um Estudo Sobre Editores Colaborativos para Modelagem Distribuída de Sistemas**. 2001. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS.

[COC 2000] COCKBURN, Alistair; WILLIAMS, Laurie. **The Costs and Benefits of Pair Programming. Extreme Programming and Flexible Processes in Software Engineering XP**. 2000. Disponível em: <http://collaboration. csc.ncsu.edu/laurie/pubs.htm>. Acesso em: 14 ago. 2001.

[COD 70] CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Communication of the ACM**, New York, v. 13, n. 6, p. 377-387, 1970.

[DEC 95] DECOUCHANT, Dominique; SALCEDO, Manuel Romero; QUINT, Vicent. **Structured Cooperative Authoring on the World Wide Web**. 1995. Disponível em: <http://www.w3.org/pub/Conferences/WWW/ Papers/91/>. Acesso em: 12 set. 2002.

[DIE 96] DIETRICH, Elton; LIMA, José Valdeni de. **Projeto de um Sistema de Suporte a Autoria Cooperativa de Hiperdocumentos**. 1996. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[DEI 2001] DEITEL, M. H.; DEITEL, P. J. **Java, Como Programar**. Porto Alegre: Bookman, 2001.

[DIL 94] DILLENGOURG, P. **The Evolution of Research on Collaborative Learning**. 1994. Disponível em: <http://tecfa.unice.ch/ tecfa/research/lhm/ ESF-Chap5.text>. Acesso em: 03 ago. 2000.

[EDW 99] EDWARDS, W. Keith. **Core Jini**. Upper Saddle River: Prentice Hall Ptr, 1999. 772 p.

[FAR 98] FARLEY, Jim. **JAVA™ Distributed Computing**. Sebastopol: O'Reilly, 1998.

[FER 98] FERNANDES, Jorge Henrique Cabral. CIBERESPAÇO: Modelos, Tecnologias, Aplicações Perspectivas. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 19., 1998, Belo Horizonte.

Anais... Belo Horizonte: DCC – UFMG, 1998.

[FRE 99] FREEMAN, E.; HUPFER, S.; ARNOLD, K. **JavaSpaces:** Principles, Patterns, and Practice. Reading: Addison Wesley, 1999.

[FRI 97] FRITZKE, Udo Jr.; FARINES, Jean-Marie. Modelagem e Implementação de Aplicações de Computação Cooperativa em Ambientes Distribuídos. In: SEMINÁRIO INTEGRADO DE HARDWARE E SOFTWARE, SEMISH'97, 24., 1997, Brasília. **Anais...** Brasília: UnB, 1997.

[FUR 98] FURLAN, José Davi. **Modelagem de Objetos através da UML:** The Unified Modeling Language. São Paulo: Makron Books, 1998.

[GAM 2000] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de Projeto:** Soluções Reutilizáveis de Software Orientado a Objetos. Porto Alegre: Bookman, 2000.

[GEL 85] GELERNTER, D. Generative Communication in Linda, **ACM Transactions on Programming Languages and Systems**, New York, v. 7, n. 1, p. 80-112, 1985.

[GRA 99] GRALEWSKI, Daniel D.; WINCKLER, Marco A. A.; INDRUSIAK, Leandro S.; REIS, Ricardo A. L. **JALE – JAVA Layout Editor**, In: UFRGS MICROELETRONICS SEMINAR, SIM, 14., 1999, Pelotas. **Proceedings...** Porto Alegre: Instituto de Informática, 1999. p. 51-53.

[GRU 94] GRUDIN, J. Computer-Supported Cooperative Work: History and Focus. **Computer,** Los Alamitos, v.27, n.5, p. 19-26, May 1994.

[HER 2000] HERNANDEZ, Emerson, B.; SAWICKI, Sandro; INDRUSIAK, Leandro, S.; REIS, Ricardo. WWW an Environment to IC Project: The Tool JASE (Java Schematic Editor). In: UFRGS MICROELETRONICS SEMINAR, SIM, 15., 2000, Torres. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2000. p. 35-38.

[HER 2001] HERNANDEZ, Emerson, B.; SAWICKI, Sandro; INDRUSIAK S. Leandro; **Homero:** Um Editor VHDL Cooperativo via Web. In: WORKSHOP IBERCHIP, 7., 2001, Montevideo. **[Memorias]**. Montevideo: Universidad de la República, 2001.

[HER 2001a] HERNANDEZ, Émerson B.; REIS, Ricardo. **Homero :** Editor de Descrições Textuais para Trabalho Colaborativo. Trabalho de Diplomação. Instituto de Informática, UFRGS. Porto Alegre, 2001.

[HUP 2001] HUPFER, Susanne. **The Nuts and Bolts of Compiling and Running**

JavaSpaces Programs. 2001. Disponível em: <http://developer. java.sun. com/developer/technicalArticles/ Programming/javaspaces/>. Acesso em: 15 maio 2001.

[IET 98] INTERNET ENGINEERING TASK FORCE. Disponível em: <http://www.ietf.org>. 1998. Acesso em: 26 mar. 2001.

[IND 97] INDRUSIAK, Leandro. S.; REIS, Ricardo. A. L. A WWW approach for EDA Tool Integration. In: SYMPOSIUM ON INTEGRATED CIRCUIT AND SYSTEMS DESIGN, SBCCI, 10., 1997, Gramado. **Proceedings**... Porto Alegre: Instituto de Informática da UFRGS, 1997.

[IND 98] INDRUSIAK, Leandro. S.; REIS, Ricardo A. L. **Ambiente de Apoio ao Projeto de Circuitos Integrados Utilizando World Wide Web**. 1998. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[IND 98b] INDRUSIAK, Leandro S.; REIS, Ricardo A. L. A Case Study for a WWW Based Cad Framework. In: BRAZILIAN SYMPOSIUM ON INTEGRATED CIRCUIT DESIGN, SBCCI, 11., 1998, Armação de Búzios. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p. 116-119.

[IND 98c] INDRUSIAK, Leandro S.; REIS, Ricardo A. L. A Case Study for the Cave Project. In: UFRGS MICROELETRONICS SEMINAR, SIM, 13., 1998, Porto Alegre. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1998. p. 39-42.

[IND 99] INDRUSIAK, Leandro S.; REIS, Ricardo A. L. 3d Integrated Circuit Layout Visualization Using VRML. In: INTERNATIONAL CONFERENCE ON WEB-BASED MODELING AND SIMULATION, WEBSIM, 1999, San Francisco, CA, USA, 1999. **Proceedings....** San Diego: Society for Computer Simulation International, 1999. p.177 – 181.

[IND 99a] INDRUSIAK, Leandro S.; REIS. Ricardo, A. L. Project Management and Design Methodology Support for the Cave Project : a Hyperdocument-Centric Approach. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 12., 1999, Natal. **Proceedings**... Los Alamitos : IEEE Computer Society, 1999. p. 188-191.

[IND 2001] INDRUSIAK, Leandro. S.; BECKER, J.; GLESNER, M.; REIS, Ricardo A. L. Distributed Collaborative Design over Cave2 Framework. In: INTERNATIONAL CONFERENCE ON VERY LARGE INTEGRATED VLSI, 11., Montpellier, 2001. **Proceedings...** Montpellier: LIRMM, 2001.

[IND 2002] INDRUSIAK, Leandro. S.; GLESNER, M.; REIS, Ricardo A. L. Comparative Analysis and Application of Data Repository Infrastructure for Collaboration-enabled Distributed Design Environments. In: DESIGN AUTOMATION AND TEST IN EUROPE, DATE, 39., 2002, Paris. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002.

[IND 2002a] INDRUSIAK, Leandro S.; REIS, Ricardo A. L. **A Review on the Framework Technology Supporting Collaborative Design of Integrated Systems.** 2002. Exame de Qualificação (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[IOC 91] IOCHPE, Cirano; LIVI, Maria A. C. Suporte à Cooperação em Ambiente de Projeto Assistido por Computador. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 6., 1991, Manaus. **Anais...** Manaus:[s.n.], 1991. p. 101-116.

[JAC 2001] JACOBS, Stephan; GEBHARDT, Michael; KETHERS, Stefanie; RZASA, Wojtek. **Filing HTML Forms Simultaneously:** CoWeb – Architecture and Funcionality. 2001. Disponível em: <http://decweb.ethz.ch/WWW5/www220/ paper.htm>. Acesso em: 09 set. 2001.

[JAV 2000] JAVA.SUN.COM. Java Tecnology. 2000. Disponível em: <http://java.sun.com>. Acesso em: 10 nov. 2000.

[JOH 92] JOHNSON. R. Documenting Frameworks Using Patterns. In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, LANGUAGES AND APPLICATIONS, 8., 1992, Washington DC. **Tutorial Notes...** [S.l:s.n.], 1992.

[KÄF 90] KÄFER, W. **A Framework for Version-Based Cooperation Control**. 1990. Disponível em: <http://wwwdbis.informatik.uni-kl.de/pubs/papers/ Kae91. DASFAA.html>. Acesso em: 19 out. 2001.

[KIR 2001] KIROWSKI, D. R.; POTKONJAK M.; DRINIC M. Hypermedia Aided Design. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings...** New York: ACM, 2001.

[KON 99] KONDURI, Gangadhar; CHANDRAKASAN, Anantha. A Framework for Collaborative and Distributed Web-based Design. In: DESIGN AUTOMATION CONFERENCE, DAC, 36., 1999, New Orleans, Louisiana. **Proceedings...** New York: ACM, 1999.

[KOR 91] KORTH, Henry F.; SILBERSCHATZ, Abraham. **Database System Concepts**. São Paulo: McGraw Hill, 1991.

[LAR 2000] LARMAN, Craig. **Utilizando UML e Padrões**: Uma Introdução à Análise e ao Projeto Orientados a Objetos. Porto Alegre: Bookman, 2000. 492 p.

[LAV 97] LAVANA H., KHETAWAT A.; BRGLEZ F.; KOZMINSKI K. **Executable workshops:** A Paradigm for Collaborative Design on the Internet. In: DESIGN AUTOMATION CONFERENCE, DAC, 34., 1997, Anaheim, CA, USA. **Proceedings...** New York: ACM, 1997.

[LAV 2000] LAVANA H.; KHETAWAT A.; BRGLEZ F.; KOZMINSKI K. **CollabWiseTk**: A Toolkit for Rendering Stan-alone Aplications Collaborative. In: DESIGN AUTOMATION CONFERENCE, DAC, 37., 2000, Los Angeles, CA, USA. **Proceedings...** New York: ACM, 2000.

[LEM 98] LEMAY, Laura; CADENHEAD, Roger. **Sam´s Teach Yourself Java 1.2 in 21 Days**. Indianapolis: Sams Publishing, 1998.

[MEY 97] MEYER, Bertrand. **Object-Oriented Software Construction**. New York: Prentice Hall, Inc, 1997.

[MOR 2000] MORGAN, Steven. Jini to the Rescue. **Spectrum IEEE**, New York, v.37, n.4. p. 44-49, April 2000.

[MOR 2000a] MORGAN, Michael. **Java 2 para Programadores Profissionais**. Rio de Janeiro: Ciência Moderna, 2000.

[MUE 2000] MUELLER, G.; BRAEUTIGAM, F. **Tutorial**: Getting Started with ozone. The Ozone database Project. 2000. Disponível em: <http://www.ozone-db.org>. Acesso em: 16 set. 2001.

[NOD 90a] NODINE, M. H. **Cooperative Transaction Model Hierarchies**. 1990. Disponível em: < http://citeseer.nj.nec.com/context/62681/0>. Acesso em: 10 abr. 2001.

[NOD 90b] NODINE, M. H. **Synchronization and Recovery in Cooperative Transactions**. 1990. Disponível em: <http://www.informatik.uni-trier.de/ ~ley/db/journals/vldb/NodineZ92.html>. Acesso em: 11 abr. 2001.

[OAK 97] OAKS, Scott; WONG, Henry. **Java Threads**. Sebastopol: O'Really & Associates, 1997.

[OST 2001] OST, Luciano C.; MAINARDI, M.; INDRUSIAK, Leandro S.; REIS, Ricardo A. L. Jale3D - Platform-independent IC/MEMS Layout Edition Tool. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 14., 2001, Pirenopolis. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2001.

[OTS 97] OTSUKA, J. L. Proposta de um Sistema de Aprendizagem Colaborativa Baseado no WWW. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 8., 1997, São José dos Campus, SP. **Anais...** São José dos Campus: SBC, 1997.

[PIN 99] PINHEIRO, Manuele Kirsch. **Edição Cooperativa de Hiperdocumentos na WWW**. 1999. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[PIN 99a] PINHEIRO, Manuele Kirsch**.** Mecanismo de Monitoramento e Contextualização em Ambientes Cooperativos. In: SIMPOSIO NACIONAL DE INFORMATICA, 4., Santa Maria, 1999. **Anais...** Santa Maria: Multipress, 1999. p. 46-47.

[PRE 92] PRESSMAN, Roger S.; **Software Engineering:** A Practitioner´s Approach. Auckland: McGraw-Hill, 1992.

[REI 98] REIS, Rodrigo. Q.; LIMA, C.; NUNES, Daltro. **Uma Proposta de Suporte ao Desenvolvimento Cooperativo de Software no Ambiente PROSOFT.** Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, 1998.

[QUI 90] QUINTANA, E. B.; ANNARUMMA, G. O.; NETO, P. M. Design version management in the GARDEN framework. In: DESIGN AUTOMATION CONFERENCE, DAC, 28., 1990, Orlando, Flórida, USA. **Proceedings...** New York: ACM, 1991.

[REI 2000] REIS, Carla Alessandra Lima. REIS, R. Q; NUNES, Daltro J. Evolução do ambiente PROSOFT para Apoiar Aspectos de Distribuição, Cooperação e Automação do Processo de Desenvolvimento de Software. In: JORNADAS IBEROAMERICANAS DE INGENIERIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 3., 2000, Cancún, México. **Proceedings...** Cancún:[s.n.], 2000. p. 192-203.

[REI 2000a] REIS, Ricardo Augusto da Luz; REIS, André Inácio. **Ferramentas de CAD**. In: Concepção de Circuitos Integrados. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzato, 2000.

[RUM 94] RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro: Campus, 1994.

[KEN 99] KEN, Arnold; BRYAN, Sullivan; SCHIFLER, Robert W.; WALDO, Jim; WOLLRATH, Ann. **The Jini Specification**. Reading: Addison-Wesley, 1999. 385 p.

[SAW 2001] SAWICKI, Sandro; HERNANDEZ, E. B.; INDRUSIAK, Leandro. S.; REIS, Ricardo. A. L. Cooperative Project Using the Cave Framework. In: MICROELETRONICS SEMINAR, SIM, 16., 2001, Santa Maria. **Proceedings...** Porto Alegre: Instituto de Informática, UFRGS, 2001. p. 53-56.

[SAW 2001a] SAWICKI, Sandro; HENANDEZ, E. B.; INDRUSIAK, Leandro. S.; REIS, Ricardo. A. L. Collaborative Project Using the Cave Framework. In: USER FORUM SBMICRO, Sforum, 1., 2001, Pirenópolis, Goiás **Proceedings...** Porto Alegre: Instituto de Informática, UFRGS, 2001.

[SAW 2002] SAWICKI, Sandro; INDRUSIAK, L. S.; REIS, R. A. L. Projeto Cooperativo no Ambiente Cave. In: WORKSHOP IBERCHIP, 8., 2002, Guadalajara, México. **Proceedings...** Guadalajara:[s.n.], 2002.

[SAW 2002a] SAWICKI, Sandro; BRISOLARA, Lisane; INDRUSIAK, L. S.; REIS, R. A. L. Collaborative Design using a Shared Object Spaces Infrastructure. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 15., Porto Alegre. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002.

[SAW 2002b] SAWICKI, Sandro; BRISOLARA, Lisane; INDRUSIAK, L. S.; REIS, R. A. L. Collaborative Design based on Shared Object Spaces. In: SOUTH MICROELECTRONICS SEMINAR, SIM, 17., 2002, Gramado. **Proceedings...** Porto Alegre: Instituto de Informática, UFRGS, 2002.

[STE 87] STEFIK, M; et al. **WYSIWIS revised:** Early Experiences With Multiuser Interfaces. 1987. Disponível em: < http://bmrc.berkeley.edu/courseware/ cscw/fall97/notes/groupware-design.html >. Acesso em: 07 jan. 2002.

[SOU 97] SOUZA, C. R. B.; WAINWE, J.; RUBIRA, C. M. F. Um modelo de anotações para o desenvolvimento cooperativo de software. In: WORKSHOP ON HYPERMEDIA E MULTIMEDIA APPLICATIONS, 3., 1997, São Carlos. **Anais...** São Carlos:[s.n.], 1997. p. 143-154.

[SOU 96] SOUZA, Adriana Silveira de; GOLENDZINER, Lia Goldstein; **Um Estudo Sobre Trabalho Cooperativo Suportado por Computador (CSCW)**. 1996. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[SOU 98] SOUZA, Adriana Silveira de. **Uma Proposta de Mecanismo de Interface Cooperativa para Suporte a Identificação de Conflitos de Linguagem em Ambiente de Desenvolvimento de Software**. 1998. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[SUN 99a] SUN MICROSYSTEMS. **Why Jini Technology Now?** 1999. Disponível em: <http://www.sun.com/jini/ whitepapers/>. Acesso em: 23 set. 2000.

[SUN 99b] SUN MICROSYSTEMS. **Jini Architectural Overview**. 1999. Disponível em: <http://www.sun.com/jini/ whitepapers/>. Acesso em: 24 set. 2000.

[TAN 92] TANENBAUM, Andrew. **Sistemas Operacionais Modernos**. Rio de Janeiro: LTC, 1999. 493 p.

[TAN 96] TANENBAUM, Andrew S. **Computer Networks.** Upper Saddle River: Prentice-Hall, 1996.

[TEX 97] TEXEL, Putnam P. **Use Cases Combined With BOOCH/OMT/UML**: Process and Products. Upper Saddle River: Prentice Hall, c1997. 465 p.

[UNA 91] UNAMAKER, J.F. Electronic Meeting Systems to Support Group Work. **Communications of the ACM**, New York, v.34, n.7, p. 40-61, July 1991.

[VIE 89] VIEGAS, A.H. de Lima; MARTINS, R. S.; CARNEIRO, L.M.F. GARDEN: an Integrated and Evolving Environment for ULSI/VLSI CAD Applications. **IBM Systems Journal**, New York, v.28, n.4, 1989.

[WAG 91] WAGNER, Flávio R.; VIEGAS, Arnaldo H. L.; GOLENDZINER, Lia G.; IOCHPE, Cirano. STAR: Um Ambiente para Integração de Ferramentas de Projeto de Sistemas Digitais. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA. SBMicro, 6., 1991, Belo Horizonte. **Anais...** Belo Horizonte:UFMG, 1991. p. 176-185.

[WAG 94] WAGNER, F. R. **Ambientes de Projeto de Sistemas Eletrônicos.** 1994. Curso ministrado na 9. Escola de Computação. Recife: UFPE, 1994. 156p.

[WIL 2000] WILLIAMS, L. KESSLER, R. R. All I Really Need to Know about Pair Programming I Learned in Kindergarden. **Communications of the ACM**, New York, v.43, n.5, p 108-114, 2000.

[WIL 2000a] WILLIAN, L.; KESSLER, R. R.; CUNNINGHAM, W.; JEFFRIES, Ron. **Strengthening the Case for Pair Programming**. 2000. Disponível em: < http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF>. Acesso em: 20 nov. 2000.

[WIL 2000b] WILLIAN, L.; KESSLER, R. R. **Experimenting with Industry's Pair-Programming Model in the Computer Science Classroom**. 2000. Disponível em: <http://collaboration.csc.ncsu.edu/laurie/Papers/csed.pdf>. Acesso em: 30 nov. 2000.

[WIL 99] WILLIANS, L. **Pair Programming Questionnaire**. 1999. Disponível em:

<http://limes.cs.utah.edu/questionnaire/questionnaire.htm>. Acesso em: 08 jan. 2002.

[WIL 94] WILLIANS, Neil; BLAIR, Gordan S.; COULSON, Geoff; DAVIES, Nigel; RODDEN, Tom. **The Impact of Distributed Multimedia systems on CSCW**. 1994. Disponível em: <http://www.sics.se/ice/people/tom.html >. Acesso em: 07 jan. 2003.

[YOR 90] YOURDON, Edward. **Modern Structured Analysis**. Upper Saddle River: Prentice Hall, 1990.