Universidade Federal do Ceará
Centro de Tecnologia
Departamento de Engenharia de Teleinformática
Programa de Pós Graduação em Engenharia de Teleinformática

ADContexto - Uma Arquitetura para Ambientes Virtuais de Colaboração Orientada a Contexto

Paulo Roberto Furtado Serra

Orientador
Prof. Dr. José Neuman de Souza
Co-orientador
Prof. Dr. Antônio de Barros Serra

Fortaleza – Ceará
Setembro 2008

Universidade Federal do Ceará
Centro de Tecnologia
Departamento de Engenharia de Teleinformática
Programa de Pós Graduação em Engenharia de Teleinformática

ADContexto - Uma Arquitetura para Ambientes Virtuais de Colaboração Orientada a Contexto

Autor
Paulo Roberto Furtado Serra

Orientador
Prof. Dr. José Neuman de Souza
Co-orientador
Prof. Dr. Antônio de Barros Serra

Dissertação de Mestrado apresentada à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de ***Mestre em Engenharia de Teleinformática****.*

Fortaleza – Ceará
Setembro 2008

Paulo Roberto Furtado Serra

ADContexto – Uma Arquitetura para Ambientes Virtuais de Colaboração Orientada a Contexto

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Teleinformática e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Paulo Roberto Furtado Serra

Banca Examinadora:

Prof. José Neuman de Souza, Dr.
Universidade Federal do Ceará – UFC

Prof. Antônio de Barros Serra, Dr.
Centro Federal de Educação Tecnológica do Ceará – CEFET/CE

Prof. Giovanni Cordeiro Barroso, Dr.
Universidade Federal do Ceará – UFC

Prof. José Marques Soares, Dr.
Centro Federal de Educação Tecnológica do Ceará – CEFET/CE

Profa. Judith Kelner, Dra.
Universidade Federal de Pernambuco – UFPE

Resumo

Os Ambientes Virtuais de Aprendizagem têm evoluído a cada dia. Tais ambientes possuem diversos serviços que buscam atender às necessidades dos seus usuários. Várias pesquisas e estudos vêm sendo realizadas para aprimorar os recursos fornecidos por tais ambientes. No entanto, existem alguns aspectos considerados importantes no presente trabalho que merecem atenção especial. Dentre eles, destacam-se: utilizar múltiplas interfaces, permitindo que usuários acessem o ambiente através de interfaces gráficas diferentes; oferecer serviços orientados a contexto, possibilitando a escolha dos serviços que estarão disponíveis a um grupo de participantes; configurar os serviços de forma dinâmica, permitindo que usuários escolham as interfaces acionadas pelos serviços e prover independência de infra-estrutura, o que possibilita ao ambiente escolher sua plataforma de comunicação (P2P ou Cliente/Servidor). Esse trabalho tem como objetivo a concepção, a especificação e a avaliação de uma arquitetura, denominada ADContexto (Arquitetura Dinâmica Orientada a Contexto) que, tomando como base alguns paradigmas da engenharia de software, notadamente a orientação a aspectos e a programação reflexiva, atenda aos aspectos mencionados. Visando avaliar a solução proposta através desta arquitetura, foi também implementado um protótipo de um ambiente de colaboração virtual, denominado AVContexto (Ambiente Virtual de Colaboração Orientado a Contexto). Através da implementação do AVContexto foi possível mostrar que é viável agregar novas características aos ambientes colaborativos de forma a tornar estas ferramentas mais flexíveis e acessíveis aos seus usuários.

Abstract

The Virtual Learning Environments has been envolved every day. Such environments have many services that meet the needs of their users. Several surveys and studies have been undertaken to improve the resources provided by these environments. However, there are some issues considered important by this work that deserve special attention. Among the main aspects is the use of multiple interfaces, allowing users to access the environment through different graphical interfaces; provision of the context-oriented services, allowing the choice of services that will be available to a group of participants; set up services such dynamic allowing users to choose the interfaces driven by the services; and provide independence of infrastructure, which allows the environment choose the communication´s plataform that will be used, P2P or Client/Server. This work aims to specify and to design a architecture, called ADContexto (Context-Oriented Dynamic Architecture) that, building upon some software engineering paradigms, more specifically the Aspect-Oriented Programming and Reflection-Oriented Programming, addressing the aspects mentioned. In order to evaluate the proposed solution through this architecture was also implemented a prototype of a collaborative virtual environment, called AVContexto (Virtual Environment for Context-Oriented Collaboration) using the proposed architecture. Through the implementation of AVContexto was possible to show that it is feasible to add new features in collaborative environments to make these tools more flexible and accessible to its users.

Dedico este trabalho a Deus e a minha família.

Sumário

Lista de Figuras viii

Lista de Tabelas ix

Lista de Siglas x

1 Introdução 1
- 1.1 Objetivos Específicos . . . 2
- 1.2 Estrutura do Trabalho . . . 3

2 Ambientes Virtuais Colaborativos 5
- 2.1 Ensino a Distância e Projetos Colaborativos . . . 6
- 2.2 Aplicações Colaborativas . . . 7
  - 2.2.1 AulaNet . . . 7
  - 2.2.2 Teleduc . . . 8
  - 2.2.3 Moodle . . . 11
  - 2.2.4 COGEST . . . 12
- 2.3 Aspectos importantes em Ambientes Virtuais de Colaboração . . . 14
  - 2.3.1 Múltiplas Interfaces Gráficas . . . 14
  - 2.3.2 Serviços Orientados a Contexto . . . 15
  - 2.3.3 Serviços Dinâmicos . . . 16
  - 2.3.4 Infra-estrutura de comunicação . . . 17
- 2.4 Trabalho Relacionado . . . 19
  - 2.4.1 Arquitetura do Ambiente AulaNet . . . 19
- 2.5 Considerações finais . . . 20

3 Técnicas de Engenharia de Software adotadas no ADContexto 22
- 3.1 Programação Orientada a Aspectos - POA . . . 22
  - 3.1.1 Separação de Interesses ou Responsabilidades . . . 22
  - 3.1.2 Definição de Programação Orientada a Aspectos . . . 25
  - 3.1.3 Composição de um sistema orientado a aspectos . . . 26
  - 3.1.4 Combinação de Aspectos . . . 27
  - 3.1.5 Conceitos fundamentais da Orientação a Aspectos . . . 28

3.2 Reflexão Computacional 31
3.2.1 Reflexão Estrutural 32
3.2.2 Reflexão Comportamental 33
3.3 Reflexão na Linguagem Java 33
3.4 Considerações Finais 35

4 ADContexto - Uma Arquitetura Dinâmica Orientada a Contexto 36
4.1 Arquitetura do ADContexto 36
4.2 Camada de Aplicação 37
4.2.1 Interface Gráfica de Aplicação 37
4.2.2 Adaptador de Interface gráfica de Aplicação 39
4.3 Camada de Controle 40
4.3.1 Interface de Inicialização 41
4.3.2 Núcleo da Arquitetura 41
4.3.3 Componente de Configuração 42
4.3.4 Persistência 42
4.3.5 Serviços de Contexto 43
4.4 Camada de Comunicação 44
4.4.1 Adaptador de Comunicação 44
4.4.2 Framework de Comunicação Externo 44
4.4.3 Considerações Finais 44

5 Um Ambiente Virtual Orientado a Contexto 46
5.1 Tecnologias utilizadas 46
5.1.1 Java Reflection 46
5.1.2 AspectJ 47
5.1.3 CSFrame 48
5.1.4 JXTAula 50
5.2 AVContexto - Um Ambiente Multi-interface, com Serviços Dinâmicos e Independente de Infraestrutura. 51
5.2.1 Interface de Inicialização do ambiente 51
5.3 Escolhendo o Framework de Comunicação 51
5.4 Escolhendo a Interface Visual 53
5.5 Serviços Orientados a Contexto 53
5.6 Resultados Obtidos 55
5.7 Considerações finais 56

6 Conclusão 57
6.1 Contribuições e Resultados 58
6.2 Limitações e trabalhos futuros 59
6.3 Considerações finais 59

Apêndice A Classes do ADContexto 61
A.1 Camada de Aplicação 61
A.1.1 Classe `IAdaptadorInterface.java` 61
A.2 Camada de Controle 62

A.2.1 Núcleo 62
A.2.2 Persistência 63
A.2.3 Configuração 64
A.2.4 Serviços de Contexto 66
A.3 Camada de Comunicação 68

Apêndice B Instalação e configuração do AspectJ **70**
B.1 Instalação do AspectJ 70
B.2 Configurando o AspectJ no Windows 72
B.3 Configurando o AspectJ no Linux 73
B.4 Testando a Instalação 73

Apêndice C Framework Cliente/Servidor para Comunicação **75**
C.1 Diagrama de Classes 75
C.2 Classes do Núcleo 75
C.2.1 CSFrameServer 75
C.2.2 CSFrameClient 78
C.3 Classe de Interface 83
C.3.1 ICSFrame.java 83
C.4 Classes de Negócio 83
C.4.1 Participante.java 83
C.4.2 Sala.java 84
C.5 Mensagens 84
C.5.1 TipoMensagem.java 84
C.5.2 MsgAbstrata.java 85
C.5.3 MsgClienteConectado.java 85
C.5.4 MsgCriarSala.java 86
C.5.5 MsgEntrarSala.java 86
C.5.6 MsgEnviarAmbiente.java 87
C.5.7 MsgIdCliente.java 88
C.5.8 MsgNovoParticipante.java 88

Apêndice D Código do Protótipo **90**
D.1 Interface de Inicialização do ADContexto 90
D.1.1 Arquivo `ConfiguracaoAVContexto.xml` 90
D.2 Interface Gráfica de Serviços 91
D.2.1 Código do Aspecto da Interface Gráfica de Serviços 91
D.3 Classe PersistenciaObjeto.java 96
D.4 Infra-estrutura de comunicação 96
D.4.1 Classe `InfraestruturaPeerToPeer.java` 96
D.4.2 Classe `InfraestruturaCSFrame.java` 97

Referências Bibliográficas **103**

Lista de Figuras

2.1 Serviços fornecidos pela versão 2.1 do AulaNet 8
2.2 Serviço de Forum de Discussão do Teleduc 10
2.3 Serviço de Correio do Teleduc 10
2.4 Moodle 11
2.5 Interface do COGEST 13
2.6 Serviços Orientados a Contexto - Sala de Álgebra 16
2.7 Serviços Orientados a Contexto - Sala de Matemática 16
2.8 Serviço Dinâmico - Sala de Álgebra 17
2.9 Serviços Dinâmico - Sala de Matemática 17
2.10 Arquitetura Cliente/Servidor 18
2.11 Arquitetura P2P 18
2.12 Modelos de Rede 19
2.13 Arquitetura do Aulanet 20

3.1 Exemplo de código emaranhado 23
3.2 Exemplo de código centralizado 23
3.3 Composição de um sistema orientado a aspectos (WINCK; JUNIOR, 2006) 26
3.4 Combinação aspectual (WINCK; JUNIOR, 2006) 28

4.1 Arquitetura do ADContexto 37
4.2 Diagrama de Classes do ADContexto 38
4.3 Exemplo de Interface Gráfica de Aplicação 38
4.4 Adaptador de Interface usando Aspectos 39
4.5 Adaptador de Interface usando Polimorfismo 39
4.6 Interface de Inicialização 41

5.1 Estrutura de um aspecto (WINCK; JUNIOR, 2006) 48
5.2 Arquitetura do CSFrame 49
5.3 Tipo de Inicialização 1 52
5.4 Tipo de Inicialização 2 52
5.5 Tipo de Inicialização 3 52
5.6 Tipo de Inicialização 4 52

5.7 Interface Visual 1 53
5.8 Interface Visual 2 53
5.9 Exemplo de tela para criar um grupo e configurar os serviços 54

B.1 Download do AspectJ 70
B.2 Tela inicial de instalação do AspectJ 71
B.3 Informar diretório da JRE 71
B.4 Escolha do Diretório de Instalação do AspectJ 72
B.5 Finalização da Instalação do AspectJ 72

D.1 Interface de Inicialização do protótipo 91
D.2 Interface Gráfica de Serviços do protótipo 92

Lista de Tabelas

2.1 Alguns Serviços do Teleduc 9
2.2 Algumas atividades disponibilizadas no Moodle 12
2.3 Comparação entre os Ambientes de Colaboração 20

4.1 Aspectos x Polimorfismo para Adaptadores de Interface 40
4.2 Informações fornecidas na Interface de Inicialização 41
4.3 Atributos da Classe Grupo 43
4.4 Alguns métodos da interface IAdaptadorInfraestrutura 44

5.1 Principais classes do Framework CSFrame 50
5.2 Serviços fornecidos pela Interface Visual 1 54

B.1 AloMundo.java 74
B.2 TesteAloMundo.java 74

Lista de Siglas

ADContexto	Arquitetura Dinâmica Orientada a Contexto
AVContexto	Ambiente Virtual de Colaboração Orientado a Contexto
AVC	Ambiente Virtual Colaborativo
AVA	Ambiente Virtual de Aprendizagem
P2P	Peer-to-Peer
POA	Programação Orientada a Aspectos
COGEST	Comunicação Gestual com Humanóides Virtuais
EAD	Ensino a Distância
AVA	Ambiente Virtual de Aprendizagem
C/S	cliente Servidor
SCORM	Sharable Content Object Reference Model
VNC	Virtual Network Computing
POO	Programação Orientada a Objetos
MOP	Meta-Object Protocol
API	Aplication Program Interface
JDK	Java Development Kit
FAQ	Frequently Asked Questions
GCF	Groupware Component Framework
NIED	Núcleo de Informática Aplicada a Educação
CSFrame	Client-Server Framework

Capítulo 1

Introdução

A busca contínua por mecanismos que facilitem o ensino colaborativo tem motivado diversos trabalhos de pesquisa na área de Educação a Distância (EaD). Ambientes Virtuais de Aprendizagem (AVA) têm mostrado novas alternativas para os tradicionais mecanismos de ensino através do uso de serviços de colaboração que permitem a construção coletiva do conhecimento. Ferramentas como Teleduc (AMORIM et al., 2005), AulaNet (LUCENA, 2008) e Moodle (COLE; FOSTER, 2007) têm sido utilizadas como auxílio na busca do aumento da capacidade de aprendizado de seus alunos.

A quantidade de serviços fornecidos por essas aplicações é grande, segundo (RIVERA et al., 2006). Devido a essa quantidade de serviços disponíveis nos AVAs e o número de pessoas que podem acessá-los, alguns ambientes permitem a criação de áreas virtuais para facilitar a colaboração de participantes com interesses em comum. No caso específico do Moodle (citado anteriormente), ele permite a criação de diversos tipos de cursos que possibilitam aos participantes interagirem através de salas de bate-papo, listas de discussão, etc.

Os benefícios que essas aplicações têm trazido para a comunidade acadêmica são inquestionáveis. No entanto, existem alguns aspectos considerados importantes e desejáveis nestes tipos de ambientes que ainda não são devidamente contemplados. Um deles, por exemplo, é a possibilidade de acessar o ambiente através de interfaces visuais distintas. Essa capacidade em uma aplicação colaborativa permite a criação de interfaces que atendam a especificidades do indivíduo. Interfaces projetadas para pessoas com necessidades especiais, por exemplo, poderiam ser agregadas facilmente

aos ambientes, permitindo que o trabalho colaborativo se tornasse mais democrático.

Outro aspecto importante nos AVAs é a possibilidade de selecionar os serviços voltados para um determinado contexto[1]. As ferramentas citadas anteriormente fornecem esse tipo de funcionalidade. No entanto, no âmbito desse trabalho, considera-se importante a possibilidade de selecionar interfaces específicas para os serviços. Essa característica permite que exista, além da colaboração na produção de conteúdo pedagógico, uma colaboração na construção do próprio ambiente, o que eleva o grau de interação entre os participantes de um contexto. Em (CAMPOS; TEIXEIRA, 2006), é dito que a autoria colaborativa constitui uma dinâmica na qual os envolvidos podem assumir livremente diversas funções em busca da ampliação do conhecimento, tanto individual quanto coletivo, a partir de suas próprias experiências, possibilidades e percepções. Assim, além de exercer um papel de criador de subsídios, são estimulados a analisar a importância dos recursos que estão utilizando, bem como explorar diferentes maneiras de aplicação desses meios dentro da perspectiva que melhor contemple o objetivo que pretendem alcançar.

Um outro aspecto considerado importante é a infra-estrutura de comunicação utilizada pelos ambientes. Geralmente, essa camada é construída sobre uma plataforma Cliente/Servidor (C/S), *Peer-to-Peer* (P2P) ou híbrida. Essa decisão de arquitetura ainda é motivo de discussão entre especialistas (RIZZETTI et al., 2006), já que muitos ainda confiam em um modelo centralizado como o C/S e outros preferem as vantagens prometidas por uma plataforma mais distribuída, como a P2P (SAMPAIO et al., 2008). Desse modo, esse ponto motiva uma reflexão sobre uma arquitetura voltada para ambientes virtuais de aprendizagem que possa ter um certo grau de independência em relação ao tipo de infra-estrutura a ser utilizada.

Dessa forma, este trabalho propõe a concepção, a especificação e a avaliação de uma arquitetura flexível e dinâmica, denominada ADContexto, voltada para aplicações virtuais colaborativas que, à luz de novos paradigmas da engenharia de software, atenda aos aspectos considerados importantes mencionados anteriormente (interfaces múltiplas; serviços dinâmicos e orientados a contexto; e independência de infra-estrutura de comunicação).

1.1 Objetivos Específicos

Como objetivos específicos deste trabalho, podem ser enumerados:

[1] Como contexto pode-se entender a divisão de um ambiente em Salas ou Grupos.

- Avaliar como os elementos relativos a Interface, Serviços e Infra-estrutura são tratados dentro dos ambientes virtuais colaborativos e propor contribuições no sentido de aperfeiçoar o processo de colaboração dentro dos mesmos;
- Pesquisar novos paradigmas da engenharia de software e refletir como os mesmos podem auxiliar no aperfeiçoamento dos ambientes virtuais de colaboração;
- Conceber, com base no estudo dos ambientes virtuais de colaboração e de técnicas da engenharia de software, uma solução (uma Arquitetura Dinâmica Orientada a Contexto) que permita o aprimoramento do processo de colaboração;
- Avaliar a solução (arquitetura) proposta, validando os conceitos através de implementações que ilustrem a aplicabilidade dos mesmos.

1.2 Estrutura do Trabalho

No Capítulo 2, é discutida a importância das aplicações colaborativas, sejam elas voltadas para o Ensino a Distância ou não. Algumas dessas aplicações são apresentadas e é avalidado como as mesmas lidam com os elementos relacionados: a interface visual, a contextualização dos serviços e a infra-estrutura de comunicação. Em seguida, são apresentados aspectos considerados importantes, na composição deste trabalho, que devem receber atenção especial no momento da concepção e construção destes tipos de ambientes. O Capítulo finaliza mostrando como tais aspectos são tratados em uma arquitetura específica, a do ambiente Aulanet.

No Capítulo 3, são detalhadas as técnicas de engenharia de software que serviram de base para a concepção da arquitetura ADContexto e a motivação para se utilizar cada uma delas. Será mostrada a importância do uso da Programação Orientada a Aspectos (WINCK; JUNIOR, 2006) e da Programação Reflexiva (WIKI, 2008) na construção de soluções arquiteturais mais dinâmicas. Desse modo, este Capítulo se destina a justificar o uso dessas técnicas para atingir os objetivos propostos no trabalho.

No Capítulo 4, é descrita a arquitetura ADContexto, detalhando todas as camadas que a compõem: Camada de Aplicação, de Controle e de Comunicação. Cada uma dessas camadas possui um conjunto de componentes responsáveis por

algum comportamento dentro da Arquitetura. A relação entre cada um dos componentes será ilustrada e justificada levando em consideração os aspectos considerados importantes para a concepção da solução e como as técnicas descritas no Capítulo 3 foram aplicadas.

A avaliação dos conceitos propostos na arquitetura será mostrada no Capítulo 5 através da análise de um protótipo denominado AVContexto (Ambiente Virtual de Colaboração Orientado a Contexto) que foi desenvolvido utilizando a arquitetura ADContexto. As tecnologias utilizadas em sua implementação e as principais telas e funções da aplicação serão brevemente descritas. Além disso, será explicado como o protótipo atende aos aspectos discutidos no tópico 2.3: Múltiplas interfaces Gráficas, Serviços Dinâmicos, Serviços Orientados a Contexto e Independência de Infra-estrutura de Comunicação. Os resultados obtidos com a implementação também serão apresentados.

Por fim, no Capítulo 6, serão feitas as considerações finais sobre este trabalho, assim como as conclusões obtidas com os testes realizados no protótipo. O Capítulo finaliza com a descrição de trabalhos futuros que podem ser realizados para amadurecer a arquitetura proposta.

Capítulo 2

Ambientes Virtuais Colaborativos

É incontestável o avanço obtido pela comunicação nos últimos dez anos após a explosão da internet. A cada dia surgem novas ferramentas e serviços que se destinam a resolver problemas ligados à comunicação ou ainda melhorar a colaboração entre pessoas. Essa busca incessante por novas formas de colaboração gerou como conseqüência uma vasta quantidade de trabalhos de pesquisa destinados a resolver problemas específicos de interação entre usuários.

Segundo (GOMES; RIVERA-HOYOS; COURTIAT, 2005), trabalhos colaborativos são atividades conduzidas por grupos de indivíduos apresentando diferentes requisitos e tarefas a serem executadas. Aplicações de colaboração passaram a ser um mecanismo interessante para aprimorar o conhecimento dos envolvidos, bem como permitir a construção de projetos de software em cooperação com outras pessoas.

Além disso, vários serviços têm sido agregados a estes tipos de aplicação. Alguns são citados em (RIVERA et al., 2006):

- **Chat:** Ferramenta de conversação em tempo real através do computador;
- **Quadro Branco:** Superfície virtual que pode ser utilizada nos ambientes virtuais de colaboração para que os participantes possam expressar visualmente suas idéias para os demais. Assemelha-se a um quadro branco do mundo real.
- **Mural:** Ferramenta que permite aos usuários de um ambiente colaborativo publicar avisos e comunicados para outros participantes. Geralmente é

utilizado para divulgar horários de cursos, data de eventos, propagandas de interesse público, etc.

- **Fórum:** Serviço bastante utilizado na internet para promover discussões sobre assuntos diversos. Muitas vezes utilizado para divulgar dúvidas ou questionamentos sobre determinados temas;

Em muitos casos, a colaboração entre usuários é realizada através dos Ambientes Virtuais de Aprendizagem, que são sistemas computacionais disponíveis na internet, destinados ao suporte de atividades mediadas pelas tecnologias de informação e comunicação. Permitem integrar múltiplas mídias, linguagens e recursos, apresentar informações de maneira organizada, desenvolver interações entre pessoas e objetos de conhecimento, elaborar e socializar produções tendo em vista atingir determinados objetivos (VALENTE; PRADO; ALMEIDA, 2003).

Alguns autores, como (LIMA, 2007) classificam as aplicações colaborativas em dois níveis: Ensino a Distância e Projetos Colaborativos, que serão mostrados no próximo tópico.

2.1 Ensino a Distância e Projetos Colaborativos

Como dito anteriormente, os ambientes de colaboração vêm sendo cada vez mais difundidos e adotados para fins diferentes, que podem envolver o ensido/aprendizado e a construção de projetos em colaboração. Seguindo essa linha, (LIMA, 2007) cita dois grupos de aplicações que mais se beneficiam com os serviços fornecidos para aplicações virtuais colaborativas:

- Aplicações de Ensino à Distância
- Projetos colaborativos

Ferramentas de Ensino à Distância, ou simplesmente ferramentas de EAD, são adotadas com o objetivo de facilitar o processo de ensino-aprendizagem e estimular a colaboração e interação entre os participantes de um curso baseado na Web (FUKS et al., 2005). No ensino à distância, todos os participantes estão em uma conferência única, e na maior parte do tempo o professor interage com o sistema e os alunos assistem passivamente (LIMA, 2007).

Apesar das aplicações de EAD serem as mais beneficiadas com os serviços de colaboração, existem outros tipos de aplicações, como por exemplo, os projetos colaborativos, que também possuem características semelhantes. Segundo (LIMA, 2007), em projetos colaborativos, diversos indivíduos podem colaborar para atingir um objetivo em comum, assumindo uma posição mais ativa em relação ao que é produzido.

O estudo feito sobre estes dois tipos de aplicações colaborativas mostrou alguns elementos que estão sempre presentes e que merecem uma atenção especial: a interface com o usuário, os serviços oferecidos e a infra-estrutura de comunicação. A maneira como algumas aplicações colaborativas lidam com estes elementos e como eles podem ser aprimorados visando o aperfeiçoamento do processo de colaboração é um aspecto tratado mais adiante.

2.2 Aplicações Colaborativas

Para ilustrar melhor como as aplicações colaborativas tratam os elementos (interface, serviços e infra-estrutura), serão apresentadas aplicações que têm sido bem aceitas pela comunidade acadêmica.

2.2.1 AulaNet

O AulaNet é um ambiente baseado numa abordagem *groupware*[1] para o ensino-aprendizagem na Web que vem sendo desenvolvido desde junho de 1997 pelo Laboratório de Engenharia de Software (LES) da Universidade Católica do Rio de Janeiro (PUC-Rio). Os serviços disponíveis no AulaNet, que podem ser usados na composição de um curso a distância, estão baseados no modelo de colaboração "3C" que, segundo (FUKS et al., 2005), é baseado na idéia de que para colaborar, um grupo tem que exercer três atividades principais: comunicação, coordenação e cooperação. Essa idéia é utilizada por (GEROSA et al., 2004) na concepção do AulaNet. Na figura 2.1, são ilustrados os principais serviços fornecidos pela ferramenta, bem como a divisão deles em relação ao modelo de colaboração "3C".

Os serviços do AulaNet podem ser vistos como componentes que são inseridos ou removidos conforme a necessidade do grupo. Um arcabouço de componentes denominado *Groupware Component Framework* fornece um conjunto de interfaces

[1]Sistema baseado em computador que auxilia grupos de pessoas envolvidas em tarefas comuns (ou objetivos) e que provê interface para um ambiente compartilhado

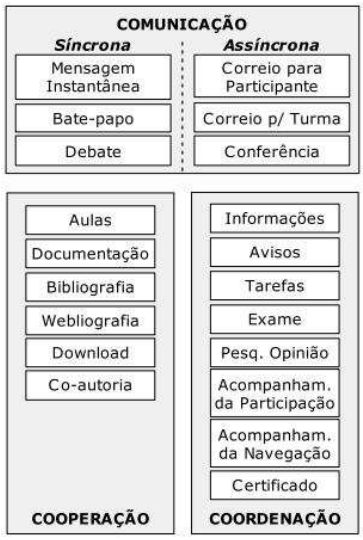


Figura 2.1: Serviços fornecidos pela versão 2.1 do AulaNet

que devem ser implementadas pelos componentes para que estes possam ser inseridos no framework e interagir com outros componentes.

Na arquitetura do AulaNet, a maior parte do processamento acontece no servidor, deixando para o cliente a tarefa de exibir a interface com o usuário. Esta estratégia é especialmente útil em um ambiente de aprendizagem, visto que seus usuários não são, necessariamente, de áreas relacionadas com a tecnologia. No entanto, alguns serviços, como o Debate, necessitam de alguns componentes funcionando no cliente. Nestes casos, a arquitetura provê mecanismos de comunicação cliente-servidor através de *applets* rodando na máquina do usuário (GEROSA et al., 2004).

2.2.2 Teleduc

O TelEduc é um ambiente de ensino a distância gratuito através do qual se pode criar, participar e gerir cursos pela Internet. Ele está sendo desenvolvido conjuntamente pelo Núcleo de Informática Aplicada a Educação (NIED) e pelo Instituto de Computação da Universidade Estadual de Campinas. O ambiente possui um esquema de autenticação de acesso aos cursos e vários recursos (ou ferramentas) (AMORIM et al., 2005). Os principais serviços da ferramenta podem ser conferidos na Tabela 2.1.

Os recursos do ambiente estão distribuídos de acordo com o perfil de seus usuários: alunos e formadores (ou professores). O conjunto total de funcionalidades oferecidas pelo Teleduc pode ser reunido em três grandes grupos: *ferramentas de*

Tabela 2.1: Alguns Serviços do Teleduc

Serviço	Descrição
Fóruns de Discussão	Tópicos que estão em discussão de maneira hierárquica (figura 2.2)
Bate-Papo	Conversa em tempo-real entre os alunos do curso e os formadores
Correio	Sistema de correio eletrônico interno ao ambiente
Dinâmica do Curso	Informações sobre a metodologia e a organização do curso
Agenda	Programação de um determinado período do curso (diária,semanal, etc.)
Avaliações	Avaliações em andamento no curso Coordenação
Atividades	Atividades a serem realizadas durante o curso
Exercícios	Exercícios com questões dissertativas, de múltipla-escolha, de associar colunas e de verdadeiro ou falso
Grupos	Gerenciamento de subgrupos para o desenvolvimento de tarefas
Perfil	Espaço para que cada participante do curso se apresente aos demais
Mural	Espaço reservado para que todos os participantes disponibilizem informações relevantes
Acessos	Acompanhamento da freqüência de acesso dos participantes ao curso e às ferramentas
Intermap	Visualização da interação dos participantes do curso nas ferramentas de comunicação
Material de Apoio	Informações úteis relacionadas à temática do curso Cooperação
Leituras	Artigos relacionados à temática do curso
Perguntas Freqüentes	Perguntas realizadas com maior freqüência durante o curso e suas respectivas respostas
Parada Obrigatória	Conteúdos que visam desencadear reflexões e discussões entre os participantes ao longo do curso
Diário de Bordo	Espaço para que cada participante registre suas experiências ao longo do curso, eventualmente de forma compartilhada
Portfólio	Armazenamento de textos, arquivos e endereços da Internet, de maneira privada ou pública

coordenação, *ferramentas de comunicação* e *ferramentas de administração*.

Como ferramentas de coordenação entende-se todas as ferramentas que, de alguma forma, organizam e subsidiam as ações de um curso. Nesse conjunto tem-se a ferramenta Agenda, que é responsável por controlar a programação de um determinado período do curso; a ferramenta Histórico, que armazena de forma seqüencial todas as agendas de um curso; a ferramenta Dinâmica de Curso, onde o formador descreve o formato do curso, tempo de duração, objetivos, o que é esperado dos alunos e avaliação.

No conjunto de ferramentas de comunicação têm-se o Correio Eletrônico (Figura 2.3), o Bate-Papo e os Fóruns de Discussão (Figura 2.2). Todas são internas ao ambiente, ou seja, para se ter acesso às mensagens do correio é preciso estar conectado ao TelEduc. Os formadores têm total liberdade de criar e eliminar grupos de discussão de acordo com tópicos que julguem relevantes a serem discutidos por meio deste tipo de ferramenta. As sessões de bate-papo são registradas e qualquer

Figura 2.2: Serviço de Forum de Discussão do Teleduc

participante do curso pode ter acesso a esses registros para posterior análise dos assuntos e tópicos discutidos. Além destas ferramentas, existe o Mural, que exibe aos participantes do ambiente diversas informações e comunicados como: avisos de eventos, links interessantes e datas importantes.

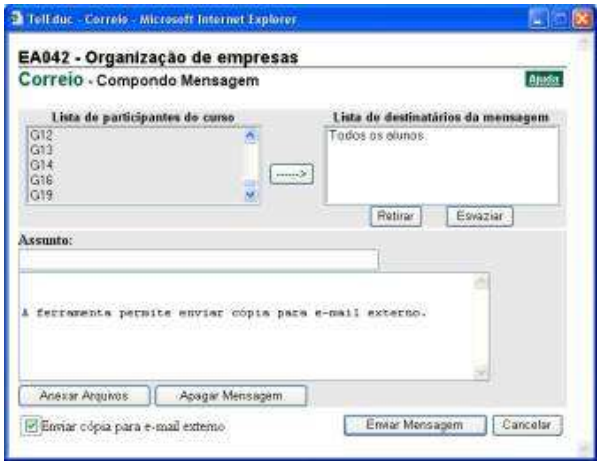


Figura 2.3: Serviço de Correio do Teleduc

Cada curso no TelEduc pode utilizar um subconjunto qualquer das ferramentas oferecidas pelo ambiente. Assim, pode acontecer de em um determinado momento do curso algumas ferramentas não estarem visíveis e, portanto, não disponíveis aos alunos. Oferecer ou não uma ferramenta em diferentes momentos do curso faz parte da metodologia adotada pelo formador. As ferramentas podem ser disponibilizadas e retiradas a qualquer momento, dependendo tão somente da dinâmica escolhida

pelo formador (ROCHA, 2002).

2.2.3 Moodle

Concebido por Martin Dougiamas, o Moodle é um software criado em 2001 para gestão da aprendizagem e de trabalho colaborativo, permitindo a criação de cursos online, páginas de disciplinas e de grupos de trabalho. Está em desenvolvimento constante, tendo como filosofia uma abordagem *social construcionista*[2] da educação.

Alguns protótipos do Moodle foram desenvolvidos e descartados, antes que fosse lançada a versão 1.0 no dia 20 de agosto de 2002. Esta versão estava dirigida a turmas de nível universitário e visava analisar a natureza da colaboração entre tais grupos. Atualmente, o Moodle possui mais de 400.000 usuários cadastrados e tradução para mais de 75 línguas em 195 países.

Figura 2.4: Moodle

Para que um usuário possa gerir uma página no Moodle, ele dever ter o privilégio de Professor, Criador ou Administrador. O Professor pode editar páginas que lhe foram atribuídas e, dependendo das permissões concedidas pelo administrador de sistema, nomear outros professores para a sua página. O Criador acumula as funções do professor, podendo criar páginas, enquanto que o Administrador tem acesso a todo o site, podendo inclusive apagar páginas.

As atividades são um dos pontos fortes do Moodle enquanto ferramenta de aprendizagem. Ele possui serviços de comunicação e discussão (Fóruns,

[2]Grupos sociais que cooperam uns com os outros, criando, de forma colaborativa, uma pequena cultura de objetos compartilhados, com significados compartilhados. Quando alguém é introduzido dentro de uma cultura como esta, está aprendendo constantemente sobre como ser uma parte dessa cultura

Chats, Diálogos), ferramentas de avaliação e construção coletiva (Teste, Trabalhos, Workshops, Wikis, Glossários) e outras, como ferramentas de pesquisa. Através do uso de *SCORM*[3] (SCORM, 2008), é possível importar para o Moodle conteúdos de *e-learning* já produzidos. Também é possível adicionar módulos a ferramenta bastando, para isso, que o módulo respeite algumas regras e passos pré-definidos.

Tabela 2.2: Algumas atividades disponibilizadas no Moodle

Serviço	Descrição
Fóruns	Os fóruns do Moodle podem ser estruturados de diversas formas (discussão geral, uma única discussão, sem respostas, etc.) e podem permitir classificação de cada mensagem. As mensagens podem também incluir anexos.
Chats	O Chat permite uma comunicação síncrona, em tempo real, entre professores e alunos. Pode ser útil como espaço de esclarecimento de dúvidas, mas pode ter outros usos. A sessão de chat pode ser agendada, com repetição.
Diálogos	O diálogo torna possível um método simples de comunicação entre dois participantes da disciplina. O professor pode abrir um diálogo com um aluno, um aluno pode abrir um diálogo com o professor, e ainda podem existir diálogos entre dois alunos.
Testes	Os testes podem ter diferentes formatos de resposta (V ou F, escolha múltipla, valores, resposta curta, etc.) e é possível, entre outras coisas, escolher aleatoriamente perguntas, corrigir automaticamente respostas e exportar os dados para Excel. O criador tem apenas de construir a base de dados de perguntas e respostas. É ainda possível importar questões de arquivos txt seguindo algumas regras.
Trabalhos	Os Trabalhos permitem ao professor classificar e comentar na página materiais submetidos pelos alunos, ou atividades 'offline' como por exemplo apresentações. As notas são do conhecimento do próprio aluno e o professor pode exportar para Excel os resultados.
Wikis	O Wiki, para quem não conhece a Wikipedia, torna possível a construção de um texto (com elementos multimídia) com vários participantes, onde cada um dá a sua contribuição e/ou revê o texto. É sempre possível acessar às várias versões do documento e verificar diferenças entre versões.
Glossários	O glossário permite aos participantes da disciplina criar dicionários de termos relacionados com a disciplina, bases de dados documentais ou de arquivos, galerias de imagens ou mesmo links que podem ser facilmente pesquisados.
Lições	A lição tenta associar a uma lógica de delivery uma componente interactiva e de avaliação. Consiste num número de páginas ou slides, que podem ter questões intercaladas com classificação e em que o prosseguimento do aluno depende das suas respostas.
Livros	Os livros permitem construir sequências de páginas muito simples. É possível organizá-las em capítulos e sub-capítulos ou importar arquivos html colocados na área de diretórios da sua página. Caso as referências dentro destes html (imagens, outras páginas, vídeo, áudio) sejam relativas, o livro apresentará todo esse conteúdo.

2.2.4 COGEST

O COGEST (Comunicação Gestual com Humanóides Virtuais) é um projeto desenvolvido na Universidade Federal do Ceará que tem como principal objetivo

[3]SCORM (*Sharable Content Object Reference Model*) é uma coleção de padrões e especificações para *e-learning* baseado na web. Através da sua utilização, é possível criar conteúdos de aprendizagem padronizados.

fazer uso de uma interface gráfica 3D (Figura 2.5) para aumentar a comunicação e o grau de percepção entre os participantes do ambiente. Isso é possível devido a comunicação gestual dos avatares[4] de cada participante. É através dos elementos perceptivos que os membros de um grupo conseguem identificar inconsistências em seu raciocínio e interagir de maneira a trocar informações e referências para auxiliar na resolução de problemas. Assim, uma maneira bastante intuitiva de perceber quem está fazendo o quê é através da observação dos gestos dos avatares dos participantes (SOARES, 2004).

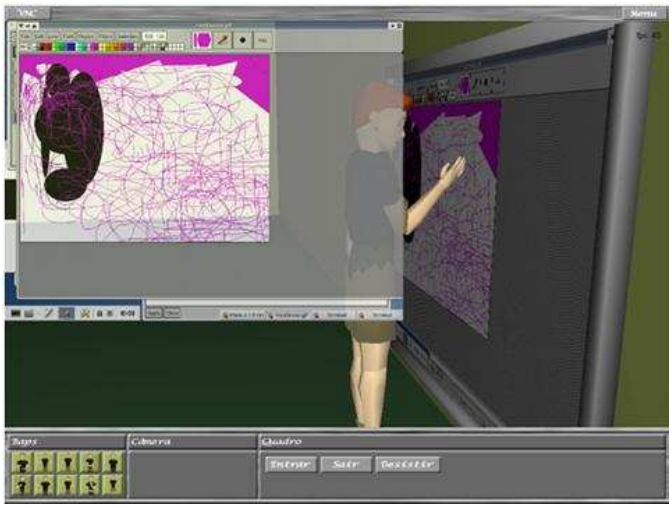


Figura 2.5: Interface do COGEST

A arquitetura do ambiente foi inicialmente desenvolvida de maneira centralizada através de dois servidores essenciais: Servidor de Eventos e Servidor VNC. O *Servidor de Eventos* é o responsável por manter o estado atual do ambiente, além de ser encarregado pela difusão de eventos entre os clientes. Sempre que um cliente deseja se comunicar, este envia uma mensagem ao servidor de eventos, para que a mensagem seja encaminhada ao destino. Por exemplo, no caso de um novo usuário ingressar no ambiente, seu avatar é adicionado ao mundo tri-dimensional e uma mensagem é enviada do servidor de eventos para os demais usuários, informando que o novo usuário entrou no ambiente. Já o *Servidor VNC* mantém a aplicação que é compartilhada no quadro virtual (Figura 2.5). O cliente que está interagindo com o quadro virtual envia os eventos para o servidor VNC, para que estes sejam realizados na aplicação compartilhada.

[4]Em informática, avatar é a representação gráfica de um utilizador em realidade virtual. De acordo com a tecnologia, pode variar desde um sofisticado modelo 3D até uma simples imagem.

Alguns trabalhos têm sido realizados com o intuito de flexibilizar a arquitetura do COGEST, tornando-a mais distribuída. Um desses trabalhos pode ser visto em (SAMPAIO et al., 2008), onde é proposta uma nova arquitetura ao ambiente fazendo uso da tecnologia JXTA (GRADECKI, 2002) para fornecer mecanismos P2P à ferramenta. Será mostrado no Capítulo 3 que o JXTAula (SAMPAIO et al., 2008), utilizado como ferramenta para modificar a arquitetura centralizada do COGEST, também poderá ser utilizado como framework de infraestrutura de comunicação da arquitetura ADContexto.

No próximo tópico, utilizando como base um conjunto de características desejáveis em um ambiente colaborativo, serão mostradas algumas premissas que foram consideradas importantes na concepção de um ambiente virtual de colaboração.

2.3 Aspectos importantes em Ambientes Virtuais de Colaboração

No tópico 2.2, foram citadas algumas aplicações colaborativas. Entretanto, existem diversas outras ferramentas em desenvolvimento que caminham neste mesmo sentido. Apesar de existir uma tendência para esse número de aplicações se tornar maior, devido a motivos como ensino, aprendizado e colaboração à distância (GEROSA et al., 2004), alguns aspectos ainda merecem atenção no que se refere, principalmente, à dinâmica dessas aplicações e como elas podem se tornar mais democráticas.

Após uma análise das características das aplicações discutidas no tópico anterior, concluiu-se, através do uso dessas ferramentas, que alguns aspectos relativos à interface, aos serviços e à infra-estrutura de comunicação poderiam ser tratados de forma diferente, visando enriquecer o processo de colaboração entre os usuários dos ambientes. Nos próximos subtópicos serão descritos os aspectos considerados importantes no contexto do presente trabalho.

2.3.1 Múltiplas Interfaces Gráficas

Geralmente, ambientes virtuais de colaboração são acessados por diversos usuários ao mesmo tempo. No entanto, muitos desses ambientes não se preocupam com algumas particularidades de cada participante. Com o uso de uma única interface gráfica de acesso ao ambiente, os participantes ficam presos a um conjunto

de requisitos para conseguirem interagir com os demais usuários. Se o ambiente tem uma interface gráfica muito rica, como um mundo virtual 3D, o participante é, muitas vezes, obrigado a ter uma boa placa de vídeo, por exemplo.

Outro aspecto interessante quando se fala em interfaces múltiplas se refere ao fato de que alguns usuários podem precisar de interfaces especiais para acessar o ambiente. Basta imaginar como seria a interação, em um ambiente virtual de colaboração, entre uma pessoa sem necessidades especiais e uma outra pessoa com alguma deficiência física que exija uma interface mais específica.

Sendo assim, um aspecto considerado importante na concepção do ADContexto, é a possibilidade de prover um mecanismo que permita que interfaces visuais diferentes sejam integradas em um mesmo ambiente de colaboração, o que possibilita, por exemplo, que interfaces projetadas para pessoas com necessidades especiais possam se comunicar com interfaces desenvolvidas para participantes que não as necessitam.

2.3.2 Serviços Orientados a Contexto

A definição de contexto segundo (DEY; ABOWD, 1999) é:

> *"Contexto é qualquer informação utilizada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação. Uma aplicação é sensível ao contexto se utiliza informações de contexto para fornecer serviço ou informação relevante para o usuário."*

Trazendo esta definição para o âmbito dos ambientes virtuais de aprendizagem, pode-se dizer que os avanços recentes da Computação têm possibilitado a construção de aplicações sensíveis ao contexto capazes de utilizar dinamicamente informações que provém do ambiente ou do usuário (COUTAZ et al., 2005). Dessa forma, além de ter conhecimento sobre o estado do ambiente em um dado momento, tais aplicações possuem a habilidade de interpretar e usar o contexto como base para um comportamento adaptativo (DAMASCENO et al., 2006).

Em outras palavras, o contexto pode ser, por exemplo, um grupo de usuários ou uma sala virtual. Desse modo, o trabalho aqui apresentado considera importante o fato de que uma aplicação colaborativa, dividida em grupos ou salas virtuais, permita que serviços diferentes sejam fornecidos por contextos diferentes.

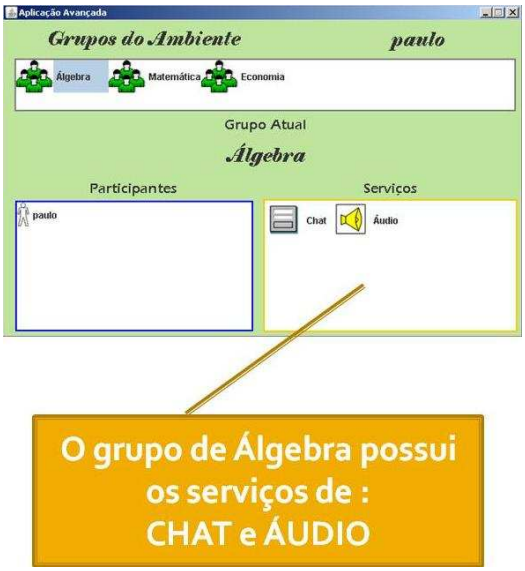


Figura 2.6: Serviços Orientados a Contexto - Sala de Álgebra

Figura 2.7: Serviços Orientados a Contexto - Sala de Matemática

Um exemplo pode ser visto nas Figuras 2.6 e 2.7. Estas Figuras foram retiradas de uma das interfaces visuais utilizadas no protótipo elaborado neste trabalho e serão detalhadas mais adiante (Capítulo 5). O que pretende-se ilustrar aqui é a possibilidade de um participante (neste caso o Paulo) entrar em uma sala de "Álgebra", (Figura 2.6) que disponibiliza os serviços de "Chat" e "Áudio" e num segundo momento, este mesmo participante estar em uma sala de "Matemática", (Figura 2.7) que possui os serviços de "Chat", "Vídeo", "Cálculo" e "Áudio". Nesse caso, cada sala passa a ser um contexto dentro do ambiente.

2.3.3 Serviços Dinâmicos

Em aplicações colaborativas que permitam a divisão virtual do ambiente, seja em grupos ou salas, é comum existirem tipos de serviços semelhantes para contextos diferentes. Por exemplo, se um ambiente virtual possui um contexto representando um grupo de estudantes de Física e um outro contexto representando um grupo de estudantes de Economia, ambos podem possuir um serviço de *chat*. Contudo, a maioria das aplicações deixa os serviços presos a uma determinada interface gráfica, não permitindo que os participantes do contexto possam definir que interface visual será acionada quando aquele serviço específico for solicitado.

Um outro aspecto considerado importante no trabalho proposto refere-se à possibilidade de se escolher dinamicamente a interface específica para um serviço

qualquer dentro de um contexto. Por exemplo, dentro de uma mesma aplicação colaborativa um contexto pode fornecer uma interface simples para o serviço de *chat*, enquanto que uma interface mais rica pode ser fornecida por outro contexto para o mesmo serviço. Isso é importante caso um determinado serviço tenha que fornecer alguma funcionalidade adicional para um contexto específico, como pode ser visto nas Figuras 2.8 e 2.9, onde duas salas diferentes (Álgebra e Matemática) possuem um serviço de Chat, mas com interfaces bem distintas.

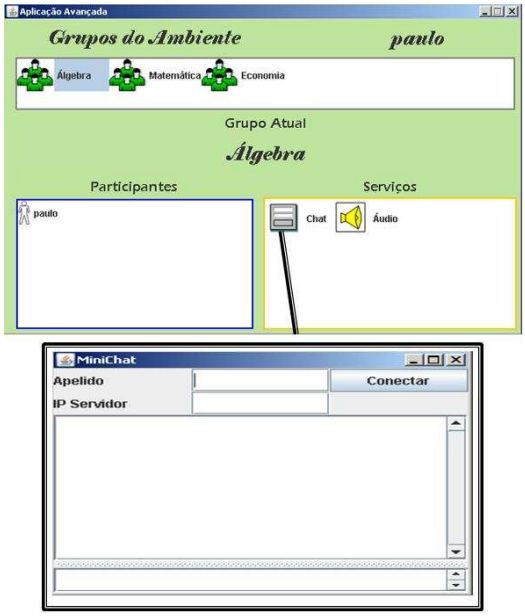


Figura 2.8: Serviço Dinâmico - Sala de Álgebra

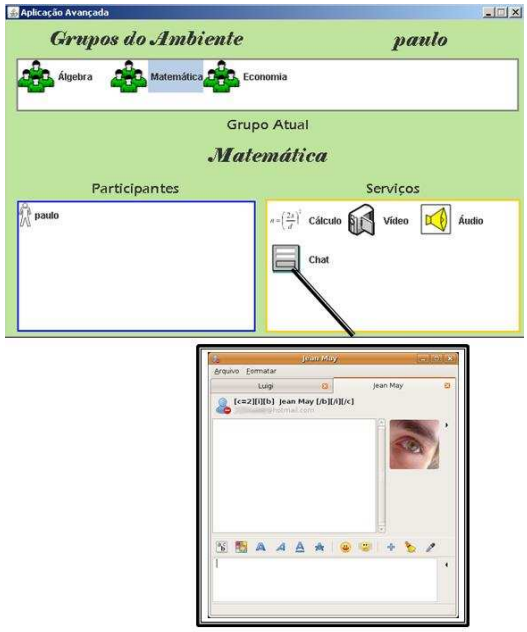


Figura 2.9: Serviços Dinâmico - Sala de Matemática

Assim sendo, a arquitetura proposta neste trabalho, que será apresentada mais adiante no Capítulo 4, propõe um mecanismo capaz de permitir que usuários possam escolher que interface visual será apresentada para um serviço qualquer em um determinado contexto. Dentro da arquitetura, o único elemento semelhante entre os contextos é a camada de infra-estrutura de comunicação, que é responsável pela dinâmica do ambiente, ou seja, quem sai ou entra de um grupo e quem está participando de um dado contexto.

2.3.4 Infra-estrutura de comunicação

Certamente, uma das decisões mais importantes no momento de conceber uma aplicação colaborativa, é saber se a comunicação vai ser provida por uma plataforma *peer-to-peer* (P2P, Figura 2.11) ou Cliente-Servidor (C/S, Figura 2.10). Não existe

ainda um consenso sobre qual a melhor alternativa, pois sabe-se que em alguns casos uma abordagem C/S é aconselhável e, em outros, o modelo P2P é mais adequado. O fato é que, atualmente, o modelo cliente-servidor é bem mais usado em aplicações corporativas pela sua facilidade de gerenciamento. No entanto, a dinâmica e o poder - seja ele computacional ou de armazenamento - que abordagens P2P trazem para aplicações distribuídas, estão ajudando a iniciar pesquisas nesta direção, principalmente depois do surgimento de ferramentas como: Napster (FANNING, 1999), SETI@home (ANDERSON et al., 2002) e Emule (BREITKREUZ, 2002). Por essa razão, várias discussões têm sido levantadas em congressos e eventos da área acadêmica para identificar os reais benefícios de cada uma dessas abordagens.

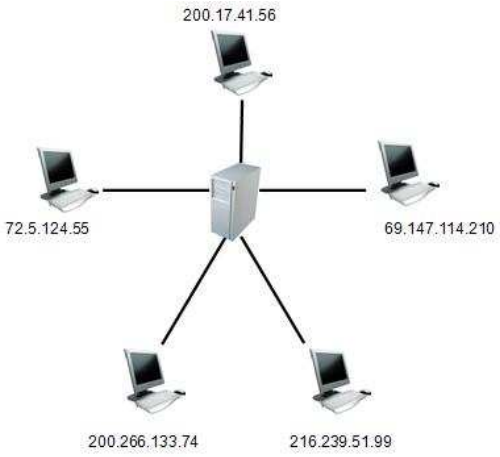


Figura 2.10: Arquitetura Cliente/Servidor

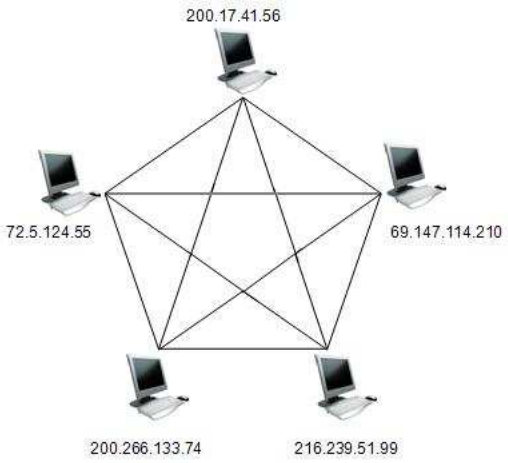
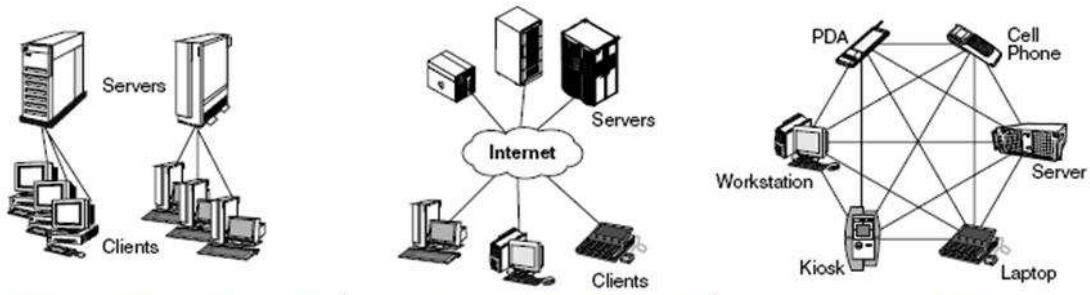


Figura 2.11: Arquitetura P2P

Segundo (RIZZETTI et al., 2006), a descentralização existente nas redes P2P tem como principal vantagem o aproveitamento de uma pequena quantidade de recursos ociosos em cada máquina, podendo fornecer grandes capacidades computacionais ao conjunto como um todo. Além disso, (WILSON, 2002) afirma que, ao contrário da arquitetura cliente/servidor, redes P2P não dependem única e exclusivamente de um servidor centralizado para prover acesso a serviços. Ou seja, uma grande vantagem no uso de modelos P2P diz respeito à possibilidade da distribuição de responsabilidades em prover serviços para os *peers*[5] da rede, eliminando o processamento excessivo em um único ponto, enquanto que no caso de uma arquitetura cliente/servidor é o próprio servidor que processa a maior parte das requisições.

[5]Peer é a definição dada a qualquer equipamento final da rede P2P, pode ser um computador, um PDA ou qualquer outro equipamento pertencente à rede P2P.

Porém, o modelo não hierárquico, característico de sistemas P2P, traz também algumas complicações que não existem no modelo hierárquico e que devem ser resolvidas. As principais são: descoberta de peers e serviços na rede, o roteamento de mensagens, o gerenciamento de segurança e grupos de usuários (RIZZETTI et al., 2006). Dessa forma, algumas vantagens da plataforma C/S em relação a P2P dizem respeito a facilidade de gerenciamento do ambiente, pois há uma concentração de serviços no lado do servidor, o que facilita o controle. Além disso, dependendo da aplicação, o cliente pode não precisar de tantos recursos computacionais, já que a maior parte do trabalho é feita pelo servidor (WILSON, 2002). Como se não bastasse, o modelo C/S é largamente aceito e está bem adaptado às necessidades do mercado, o que lhe dá uma vantagem considerável, visto que aplicações P2P comerciais ainda não são bem conhecidas. Na Figura 2.12 são ilustrados os principais modelos de redes existentes atualmente.



a. Modelo cliente-servidor, trabalha de forma hierárquica e possui uma interoperabilidade limitada

b. Modelo da arquitetura Web, já possui um certo suporte a heterogeneidade entre clientes e servidores

c. A computação P2P permite comunicação direta entre peers e novos estilos de interação

Figura 2.12: Modelos de Rede

Com o intuito de evitar essa discussão, este trabalho considerou importante a possibilidade de independência de modelo de comunicação, deixando a cargo da aplicação a escolha sobre qual infra-estrutura utilizar.

Na tabela 2.3 é mostrada uma análise de como os ambientes vistos na seção 2.2 lidam com os aspectos descritos na seção 2.3.

2.4 Trabalho Relacionado

2.4.1 Arquitetura do Ambiente AulaNet

O ambiente AulaNet (LUCENA, 2008) descrito na seção 2.2.1 faz uso de técnicas de desenvolvimento orientado a componente, que provêem a flexibilidade necessária

Tabela 2.3: Comparação entre os Ambientes de Colaboração

	Múltiplas Interfaces	**Serviços Orientados a Contexto**	**Serviços Dinâmicos**	**Independência de Infra-estrutura**
AulaNet	Não	Sim	Sim	Não
Teleduc	Não	Sim	Sim	Não
Moodle	Não	Sim	Não	Não
Cogest	Não	Não	Não	Não

em projetos com requisitos instáveis. Essa arquitetura (veja Figura 2.13) é composta por *component frameworks*, que definem os invariantes e os protocolos de conexão e comunicação entre componentes.

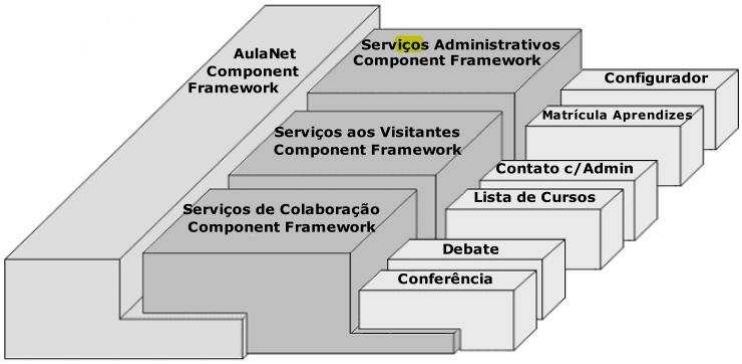


Figura 2.13: Arquitetura do Aulanet

Um dos pontos fortes da arquitetura é permitir que seus usuários possam inserir ou remover componentes de forma a montar uma ferramenta adequada a seus propósitos. Além do mais, podem experimentar diversas configurações e, a partir da realimentação obtida com a utilização da ferramenta, adaptá-la e refiná-la, de forma a acompanhar a evolução da dinâmica do seu curso e do próprio grupo, visto que este também evolui. No entanto, a arquitetura não oferece mecanismos para permitir que interfaces diferentes possam acessar a camada principal da arquitetura. Além disso, o ambiente possui um mecanismo de comunicação fixo.

2.5 Considerações finais

Como visto neste capítulo, existem soluções e ferramentas que já atendem a muitas necessidades no que se refere ao trabalho colaborativo, principalmente em relação à quantidade de serviços fornecidos. Alguns aspectos apresentados na seção

2.3 que ainda não foram devidamente tratados por aplicações colaborativas serão abordados mais adiante com o intuito de aumentar a capacidade de interação entre os mais diversos usuários. Para isso, é necessário identificar as técnicas que podem ser utilizadas para alcançar tais objetivos. No próximo capítulo, serão mostradas algumas técnicas de engenharia de software que foram usadas na concepção da arquitetura proposta neste trabalho.

Capítulo 3

Técnicas de Engenharia de Software adotadas no ADContexto

A Arquitetura proposta neste trabalho, denominada ADContexto, foi concebida com o intuito de oferecer uma alternativa para as aplicações colaborativas no que diz respeito aos aspectos discutidos no tópico Capítulo anterior (Tópico **2.3**): múltiplas interfaces visuais, independência de infra-estrutura de comunicação e o uso de serviços dinâmicos e orientados a contexto. Para atender a estes requisitos, foi necessária uma reflexão aprofundada à luz de novos paradigmas da engenharia de software que provessem uma solução para o problema proposto, principalmente no que diz respeito à Programação Orientada a Aspectos (POA) e a Programação Orientada a Reflexão. Neste capítulo, serão apresentados os principais conceitos relacionados a estas linhas de pesquisa que serviram de base para a concepção da arquitetura proposta nesta dissertação.

3.1 Programação Orientada a Aspectos - POA

3.1.1 Separação de Interesses ou Responsabilidades

Toda a evolução das linguagens de programação girou em torno da criação de meios para facilitar a criação de códigos, principalmente no que se refere à capacidade de dividir o problema. A separação de interesses é a preocupação não apenas de dividir os requisitos de uma solução, mas de agrupá-los em conjuntos de requisitos afins, procurando explicitar a existência de módulos implementáveis (WINCK; JUNIOR, 2006).

O modo de abstração apresentado na Orientação a Objetos é focado na separação de apenas um interesse da solução, habitualmente representado-o de forma bastante natural. Esse interesse normalmente é a própria solução, representando a lógica de negócio. (WINCK; JUNIOR, 2006) afirma que o par classe/objeto consegue encapsular todo o código referente à determinada característica do sistema em uma ou em poucas classes. No entanto, a necessidade de implementar um interesse adicional, como uma auditoria (log), leva à ocorrência de código espalhado, também conhecido no inglês como *scattering code*.

Em uma aplicação computacional vários interesses estão em jogo. Esses interesses são, na verdade, características relevantes da aplicação agrupadas por similaridade. É possível classificar os interesses em dois conjuntos, de acordo com (PIVETA, 2001), o primeiro grupo é formado pelos aspectos relacionados ao domínio da aplicação e denominam-se *interesses funcionais*. O segundo é formado por características de suporte à aplicação, como, por exemplo, o tratamento de exceções, que não fazem parte daquilo que pode ser chamado de lógica de negócio. A esses interesses é dado o nome de *interesses sistêmicos* ou *ortogonais*. Outros autores, como (GRADECKI; LESIECKI, 2003), também chamam estes últimos de Interesses Transversais.

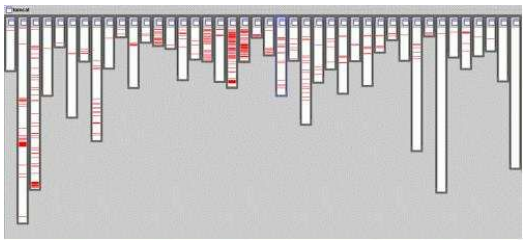


Figura 3.1: Exemplo de código emaranhado

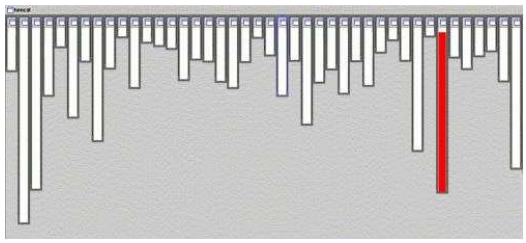


Figura 3.2: Exemplo de código centralizado

Os interesses ortogonais ou transversais, quando implementados pelas técnicas atuais (como Orientação a Objetos), podem provocar a ocorrência de código espalhado e emaranhado no sistema, como pode ser visto na Figura 3.1, onde cada barra vertical representa as classes em um sistema qualquer que implementam os interesses funcionais. Os interesses transversais são representados pelos pequenos riscos horizontais em cada barra. As seguir, serão descritos alguns desses interesses transversais segundo (PIVETA, 2001).

i. **Sincronização de Objetos Concorrentes:** Com a evolução das soluções, pode-se tornar necessário permitir ao programa o uso de múltiplos fluxos de

execução por meio de *threads*[1]. Um exemplo de política de sincronização seria a possibilidade de todos os objetos realizarem operações de leitura em um determinado objeto.

ii. **Tratamento de Exceções:** Algumas vezes, é inevitável efetuar o tratamento de exceção em um código da aplicação. Como exemplo, pode-se citar o uso de pré-condições e pós-condições que devem ser satisfeitas no momento em que uma operação é realizada. Essas condições poderiam ser implementadas separadamente.

iii. **Coordenação de Múltiplos Objetos:** Esta implementação apresenta diversos problemas quanto ao reuso e a extensão. Essa implementação é mais difícil que a primeira já mencionada, tendo em vista a necessidade de sincronizar a integração entre objetos ativos em busca de um objeto global. Os algoritmos usados para coordenação de múltiplos objetos são geralmente inerentes a certos tipos de implementações, não podendo ser reutilizados.

iv. **Persistência:** A implementação da persistência é uma parte relevante de um sistema. Quando implementado, esse interesse localiza-se propagado por boa parte do código, havendo ocorrência em diversas classes. É comum a criação de classes destinadas a essa função, e estas comporão a camada de persistência. Uma abordagem comum é efetuar a recuperação de dados no momento em que o objeto é acessado e atualizar a base de dados no momento da criação ou modificação do estado do objeto.

v. **Auditoria:** Também conhecida como *log*, a auditoria é uma das atividades mais comuns para que possamos entender o comportamento de um sistema. Na sua forma mais simplista, a funcionalidade de auditoria em um sistema consiste em exibir mensagens descrevendo a ação no exato momento em que estava sendo executada. Os mecanismos atualmente utilizados implementam a auditoria juntamente com as regras de negócio do sistema, o que resulta em uma quantidade grande de código espalhado pela aplicação que nada tem a ver com o objetivo principal a que ela se destina.

[1]Thread, ou linha de execução em português, é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente

3.1.2 Definição de Programação Orientada a Aspectos

A Programação Orientada a Aspectos (KICZALES et al., 1997), ou simplesmente POA, foi criada no fim da década de 1990, mais precisamente no ano de 1997, em Palo Alto, nos laboratórios da Xerox, por Gregor Kiczales, John Lamping, Anurag Mandhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier e John Irwin. O objetivo era construir uma abordagem que fosse um conjunto não necessariamente homogêneo, que permitisse à linguagem de programação, composta por linguagem central e várias linguagens específicas de domínio, expressar de forma ideal as características sistêmicas (também chamadas de ortogonais ou transversais) do comportamento do programa (WINCK; JUNIOR, 2006).

Essa nova forma de desenvolvimento foi proposta com o objetivo de facilitar a modularização dos interesses transversais, complementando a Programação Orientada a Objetos (POO). A POA não tem o intuito de ser um novo paradigma de programação, mas uma nova técnica que deve ser utilizada em conjunto com linguagens de programação para construção de sistemas de software com uma arquitetura melhor trabalhada, auxiliando na manutenção dos vários interesses e na compreensão do software.

Como falado anteriormente, os interesses são implementados em blocos de código. Alguns desses interesses podem ser encapsulados de forma clara em uma única unidade de função. Em POO esses interesses são modularizados em objetos, compostos por métodos que contêm a implementação do interesse, e os atributos compostos pelos dados manipulados pelos métodos. Em POA é introduzido um novo mecanismo para abstração e composição, que facilita a modularização dos interesses transversais, o aspecto (*aspect*). Na figura 3.2 é ilustrado como a orientação a aspectos pode isolar os componentes responsáveis pelos interesses sistêmicos do resto do código.

Desta forma, os sistemas de software são decompostos em componentes e aspectos. Assim, os requisitos funcionais normalmente são organizados em componentes através de uma linguagem POO, como Java, e os requisitos não funcionais como aspectos relacionados às propriedades que afetam o comportamento do sistema (KICZALES et al., 1997).

3.1.3 Composição de um sistema orientado a aspectos

Segundo (PIVETA, 2001), uma aplicação orientada a aspectos é composta pelos seguintes componentes:

Linguagem de Componentes: permitir ao programador escrever programas que implementem as funcionalidades básicas do sistema, ao mesmo tempo em que não prevêem nada a respeito do que deve ser implementado na linguagem de aspectos;

Linguagem de Aspectos: deve suportar a implementação das propriedades desejadas de forma clara e concisa, fornecendo construções necessárias para que o programador crie estruturas que descrevam o comportamento dos aspectos e definam em que situações eles ocorrem;

Combinador de Aspectos: A tarefa do combinador de aspectos (*aspect weaver*) é combinar os programas escritos em linguagem de componentes com os escritos em linguagem de aspectos.

A composição de um programa orientado a aspectos pode ser vista na Figura 3.3, que mostra a relação entre os programas escritos em linguagem de componentes, a linguagem de aspectos, os programas escritos em linguagem de aspectos e o combinador aspectual. Como saída, obtém-se, então, o código combinado entre programas escritos em linguagem de componentes e programas escritos em linguagem de aspectos.

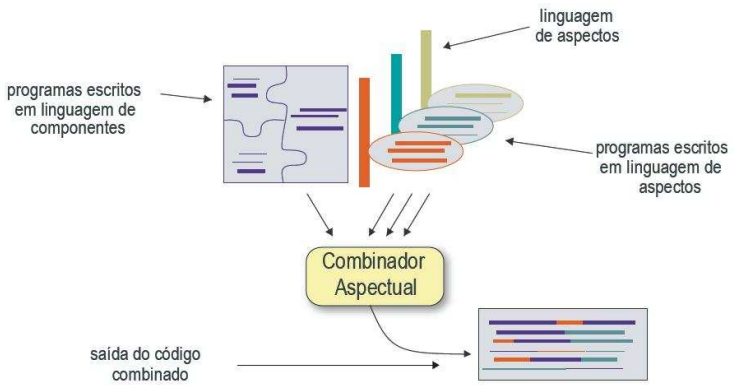


Figura 3.3: Composição de um sistema orientado a aspectos (WINCK; JUNIOR, 2006)

As listagens 3.1 e 3.2 mostram um programa escrito em linguagem de componentes (Java) e um aspecto criado para esse programa usando uma Linguagem

de Aspectos. O objetivo do aspecto da listagem 3.2 é imprimir uma mensagem antes e depois da execução do método `exibeMensagem` da classe ilustrada na listagem 3.1. Mais detalhes sobre esse código pode ser encontrado no apêndice B.

Listagem 3.1: Programa escrito em Linguagem de componente

```
1  package teste.aspect;
2
3  public class AloMundo {
4
5     public void exibeMensagem(){
6        System.out.println("Alô Mundo - Versão AspectJ");
7     }
8
9     public static void main(String[] args) {
10       AloMundo alo = new AloMundo();
11       alo.exibeMensagem();
12    }
13
14 }
```

Listagem 3.2: Programa escrito em Linguagem de Aspectos

```
1  package teste.aspect;
2
3  public aspect TesteAloMundo {
4     pointcut callExibeMensagem(): call(public void AloMundo.*(..));
5
6     before(): callExibeMensagem() {
7        System.out.println("Antes da Execução do método exibeMensagem()");
8     }
9
10    after(): callExibeMensagem() {
11       System.out.println("Depois da Execução do método exibeMensagem()");
12    }
13
14 }
```

3.1.4 Combinação de Aspectos

A Combinação de Aspectos é o processo responsável por combinar os elementos escritos em linguagem de componentes com os elementos escritos em linguagem de aspectos. É um processo que antecede a compilação, gerando um código intermediário na linguagem de componentes capaz de produzir a operação desejada, ou permitir a sua realização durante a execução do programa.

As classes referentes ao código do negócio nos sistemas não sofrem qualquer alteração para suportar a programação orientada a aspectos. Isso é feito no momento

da combinação entre os componentes e os aspectos. Esse processo também pode ser definido como recompilação aspectual, como visto na Figura 3.4.

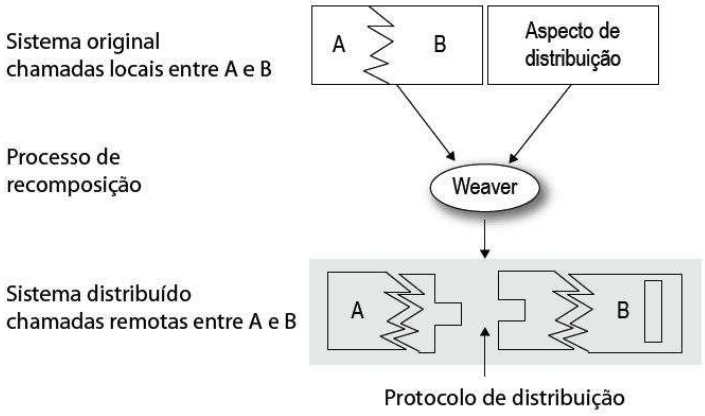


Figura 3.4: Combinação aspectual (WINCK; JUNIOR, 2006)

Segundo (PIVETA, 2001), a combinação aspectual pode ser:

- **Estática:** Um sistema orientado a aspectos, utilizando combinação estática, pode trazer agilidade, já que não há necessidade de que os aspectos existam em tempo de compilação e execução. O uso de uma combinação estática previne que um nível adicional de abstração cause um impacto negativo na performance do sistema.
- **Dinâmica:** Para que a combinação possa ser dinâmica é indispensável que os aspectos existam tanto em tempo de compilação quanto em tempo de execução. Utilizando uma interface que faça uso de recursos da programação reflexiva, o combinador de aspectos tem a possibilidade de adicionar, adaptar e remover aspectos em tempo de execução.

3.1.5 Conceitos fundamentais da Orientação a Aspectos

A orientação a aspectos possui quatro conceitos fundamentais, que serão abordados a seguir:

Pontos de junção (*join points*)

Os pontos de junção são locais bem definidos da execução de um programa, como, por exemplo, uma chamada a um método ou a ocorrência de uma exceção, dentre muitos outros. No exemplo da listagem 3.1, a linha 05 pode ser considerada

um ponto de junção, uma vez que pode ser necessário executar alguma ação antes, durante ou depois da execução do método `exibeMensagem()`.

Muitos outros pontos de junção podem ser identificados dentro de um programa, como: a execução de um método construtor, o disparo de uma exceção, a definição do valor de um atributo, etc. No entanto, o ponto de junção isoladamente não possui grande utilidade, mas quando combinado com os pontos de atuação, percebe-se a real importância do uso de aspectos.

Pontos de atuação (*pointcuts*)

Pontos de atuação são elementos de programa usados para definir um ponto de junção, como uma espécie de regra criada pelo programador para especificar eventos que serão atribuídos aos pontos de junção. Os pontos de atuação têm como objetivo criar regras genéricas para definir os eventos que serão considerados pontos de junção, sem precisar defini-los individualmente, o que tornaria a POA quase sem sentido, segundo (WINCK; JUNIOR, 2006). Para ficar mais clara essa definição de pontos de atuação, observe a listagem 3.2. Essa listagem representa um aspecto criado para o programa ilustrado na listagem 3.1. A linha 04 da listagem 3.2 (mostrada abaixo), cria um ponto de atuação chamado `callExibeMensagem()`.

```
pointcut callExibeMensagem():  call(public void AloMundo.*());
```

A linha acima indica que o ponto de atuação `callExibeMensagem()` será executado sempre que for feita uma chamada (`call`) a qualquer método da classe `AloMundo` que tenha a seguinte assinatura:

```
public void AloMundo.*()
```

Nesse caso, o único método da classe `AloMundo` que possui essa assinatura é o método `public void exibeMensagem`. A questão agora é saber o que vai ser executado pelo ponto de atuação `callExibeMensagem` e em que momento (antes, durante ou depois da execução do método `exibeMensagem()`). Para isso, faz-se uso dos adendos, explicados na próxima seção.

Adendos (*advice*)

Adendos são partes da implementação de um aspecto executados em pontos bem definidos do programa principal (WINCK; JUNIOR, 2006). Os adendos são compostos de duas partes: a primeira delas é o ponto de atuação, que define as regras de captura dos pontos de junção; a segunda é o código que será executado quando ocorrer o ponto de junção definido pela primeira parte.

A listagem 3.2 implementa dois adendos para a chamada aos métodos da classe `AloMundo` que possuem uma assinatura semelhante a: `public void AloMundo.*()`. O primeiro adendo criado (`before`), linhas 06 à 08 da listagem 3.2, exibe a mensagem *"Antes da execução do método exibeMensagem()"* quando a linha 05 da listagem 3.1 é executada. O outro adendo criado (`after`), linhas 10 à 12, exibe a mensagem *"Depois da execução do método exibeMensagem()"* logo após a execução do método `exibeMensagen()` da linha 05 na listagem 3.2.

O adendo é um mecanismo bastante similar a um método, quando comparamos com a programação orientada a objetos, cuja função é declarar o código que deve ser executado a cada ponto de junção em um ponto de atuação.

Aspectos

As propriedades de um sistema que envolvem diversos componentes funcionais não podem ser expressas utilizando a notação e as linguagens atuais de uma maneira bem localizada. Propriedades, tais como sincronização, interação entre componentes, distribuição e persistência, são expressas em fragmentos de código espalhados por diversos componentes do sistema.

Um aspecto é o mecanismo disponibilizado pela programação orientada a aspectos para agrupar fragmentos de código referente aos componentes não-funcionais em uma unidade no sistema. Para desenvolvimento de uma aplicação colaborativa, pode-se imaginar os seguintes interesses a serem implementados:

- Atualização do ambiente virtual
- Persistência dos Dados
- Controle de Acessos
- Log de Eventos

Na programação orientada a aspectos, como dito anteriormente, esses interesses não inerentes ao negócio, denominados interesses sistêmicos, são agrupados em aspectos. Dessa forma, o aspecto não pode existir isoladamente para implementação dos interesses do sistema (tanto funcionais quanto ortogonais). Os objetos continuam existindo e neles são implementados os interesses funcionais. Já nos aspectos são tratados os interesses sistêmicos. Pode-se definir, segundo (WINCK; JUNIOR, 2006), que o par aspecto-objeto é a unidade principal em um programa orientado a aspectos, assim como o objeto é o principal em um programa orientado a objetos.

A listagem abaixo mostra como ficariam as implementações ilustradas nas listagens 3.1 e 3.2 sem o uso de aspectos.

Listagem 3.3: Implementação sem o uso de aspectos

```
1  public class AloMundo {
2     public void exibeMensagem {
3        System.out.println("Alô Mundo - Versão AspectJ");
4     }
5
6     public static void main (String args[])
7        AloMundo alo = new AloMundo();
8        System.out.println("Antes da execução do método exibeMensagem()")
9        alo.exibeMensagem();
10       System.out.println("Depois da execução do método exibeMensagem()")
11    }
12 }
```

3.2 Reflexão Computacional

Outro recurso usado pelo ADContexto é a Reflexão computacional ou Programação Orientada a Reflexão, que é o comportamento apresentado por sistemas reflexivos quando estes realizam operações sobre si mesmos. O princípio de reflexão determina que um sistema deve manter uma representação causalmente conectada de seu próprio comportamento, a qual pode ser examinada e modificada pelo próprio sistema (MAES, 1987). Em outras palavras, a reflexão, no contexto da ciência da computação, é a capacidade de um programa de observar ou até mesmo modificar a sua estrutura ou comportamento. Tipicamente, o termo reflexão refere-se à reflexão dinâmica ou em tempo de execução, embora muitas linguagens suportem reflexão estática ou em tempo de compilação. Segundo (SOUSA; GROTT, 2004), o conceito de reflexão não é novo, porém, o uso de reflexão como um objeto

chave na montagem de uma arquitetura é algo pouco explorado, mas pode trazer grandes benefícios.

Ainda de acordo com (SOUSA; GROTT, 2004), um benefício do uso de reflexão se refere à possibilidade dela ser utilizada para adaptar um determinado sistema dinamicamente a diferentes situações. Considere, por exemplo, uma aplicação que use uma classe X para comunicar-se com algum serviço. Agora, suponha que essa aplicação precise comunicar-se com um serviço diferente, usando a classe Y, que tem nomes de métodos diferentes. Sem reflexão, a aplicação teria de ser modificada e recompilada. Mas, se a reflexão for usada, isso pode ser evitado. A aplicação poderia conhecer os métodos da classe X e essa classe lhe diria que método era usado para que propósito. Assim, quando um novo serviço for usado, via classe Y, a aplicação também procuraria pelos métodos necessários e os utilizaria. Nenhuma modificação no código seria necessária. Nem mesmo o nome da nova classe deveria ser recompilado com a aplicação, uma vez que ele poderia estar armazenado em algum arquivo de configuração, ser verificado e ter a sua classe carregada em tempo de execução. Esse mecanismo é comumente conhecido como *plugin* (WIKI, 2008).

Quando a reflexão é aplicada à programação orientada a objetos, tem-se a abordagem denominada de protocolo de meta-objetos (FABRE; NICOMETTE, 1995), ou MOP (*Meta-Object Protocol*), que estrutura os objetos em dois níveis: nível base (objeto-base) e nível meta (meta-objeto). Cada objeto-base x está associado com um meta-objeto *x1*, que representa aspectos estruturais e computacionais de x, gerenciados através de computações feitas em *x1*. As chamadas aos métodos do objeto-base são desviadas a fim de ativar meta-métodos que permitam a modificação do comportamento do objeto-base, ou a adição de funcionalidades a seus métodos.

Segundo (ZAVADSKI, 2003), a reflexão computacional pode ser classificada quanto aos tipos de características reflexivas oferecidas, assim como quanto ao momento em que tais características podem ser acessadas. A seguir os tipos de reflexão serão mostrados.

3.2.1 Reflexão Estrutural

A Reflexão Estrutural (*structural reflection*) baseia-se na representação ou *reificação*[2] dos elementos que constituem a aplicação, tais como a estrutura de

[2]Processo no qual entidades abstratas são convertidas em entidades concretas, passíveis de manipulação.

classes, tipos de dados e herança, os quais podem ser consultados e utilizados pela própria aplicação (FERBER, 1989). Embora os termos reflexão estrutural e introspecção (*introspection*) sejam freqüentemente utilizados como sinônimos, outros autores como (CHIBA, 2000) os diferenciam, apresentando o conceito de reflexão estrutural como sendo a possibilidade de alterar as estruturas de dados e o próprio código de um programa, possibilitando a modificação de estruturas estáticas quando necessário. Introspecção, portanto, refere-se somente à capacidade de uma aplicação inferir sobre sua estrutura, sem a possibilidade de realizar modificações.

Este tipo de reflexão permite, na verdade, que um programa possa modificar a estrutura das classes/objetos, através do acréscimo de atributos/métodos bem como a modificação da hierarquia de herança.

3.2.2 Reflexão Comportamental

Reflexão Comportamental (*behavioral reflection*) ou Intercessão (*intercession*)(CHIBA, 2000) baseia-se no princípio de interceptação da execução dos programas, ou seja, refere-se ao processo de alterar o comportamento do programa no próprio programa, através do ato de interceder, intermediar, ou interceptar (FABRE; NICOMETTE, 1995) operações de nível base (invocação de método, ou assinalamento e obtenção de valores de atributos), que podem ser reificadas e então manipuladas.

Em outras palavras, a reflexão comportamental possibilita que um programa possa modificar o seu comportamento através da invocação de elementos como objetos, métodos ou atributos de interesse. Esses elementos são, muitas vezes, desconhecidos até o momento em que são acionados. Como será mostrado no próximo capítulo, a arquitetura do ADContexto faz uso da Reflexão Comportamental para instanciar a infra-estrutura de comunicação.

3.3 Reflexão na Linguagem Java

A linguagem JAVA possui algum suporte a reflexão, apresentado como uma API especial que dispõe de acesso a informações de estado da máquina virtual. A sua capacidade, entretanto, se restringe a introspecção, permitindo ao programa, por exemplo, consultar quais os métodos de uma classe e realizar chamadas a eles, sem permitir alterá-los (MUHAMMAD; FERREIRA, 2003). Para o trabalho aqui proposto, esta característica é suficiente para atingir os objetivos do ADContexto.

A API de Reflexão Java reúne um conjunto de classes que possibilitam a representação de elementos da linguagem, como classes e objetos, bem como o acesso a tais elementos via introspecção. Sua introdução ocorreu a partir da versão 1.1 do JDK, para facilitar o desenvolvimento de outras características da plataforma, como o modelo de componentes *JavaBeans* e as características de serialização (SUN, 1998). Esta API implementa um modelo de reflexão em tempo de execução, que disponibiliza somente propriedades de introspecção sobre os elementos estruturais representados. Chiba (CHIBA, 2000) ressalta que esta API não oferece capacidades reflexivas eficazes, por não disponibilizar mecanismos mais eficientes para intervir no comportamento e na estrutura das aplicações.

Segundo (ZAVADSKI, 2003), a API Java para reflexão pode ser usada para as seguintes operações:

i. Construir novas instâncias e novos *arrays* de classes

ii. Acessar e modificar campos de objetos e classes

iii. Realizar invocações de métodos em objetos e classes

iv. Acessar e modificar elementos de *arrays*

Tais operações são realizadas através de classes especificamente projetadas para representar elementos da linguagem. Dentre estas, destacam-se:

i. `java.reflect.Method`: representa um determinado método recuperado a partir de uma classe. Pode ser utilizado para realizar a invocação dinamicamente incluindo a utilização de parâmetros;

ii. `java.reflect.Field`: representa um atributo de uma determinada classe e oferece os meios para obter e atribuir valores;

iii. `java.reflect.Constructor`: similar a `Method`, porém representando um construtor da classe.

O modelo de programação imposto pela API impede que tais classes sejam diretamente criadas pelo programa que deseja utilizar a reflexão. Ao invés disto, as instâncias destas classes são inicializadas a partir de outra classe,

a `java.lang.Class` que atua como um *Factory*[3] para instanciar os demais mecanismos reflexivos, seja a partir de uma classe carregada dinamicamente ou de um objeto previamente alocado. É importante ressaltar que toda classe Java é uma especialização implícita da classe `java.lang.Object`, a qual implementa o método denominado `getClass()`. Isto garante que qualquer classe prevista na especificação da linguagem, bem como as classes definidas pelos programadores, possuam a funcionalidade necessária para recuperar sua representação reflexiva, facilitando assim o processo de introspecção.

3.4 Considerações Finais

O uso de Programação Orientada a Aspectos e Programação Reflexiva foi adotada na especificação de alguns componentes da Arquitetura ADContexto visando alcançar os objetivos descritos no Capítulo 2. No próximo Capítulo, será apresentada a especificação da Arquitetura, seus principais componentes e como eles utilizam as técnicas de engenharia de software já descritas.

[3]Na ciência da computação, é um padrão de projeto de software (*design pattern*, em inglês) que fornece uma interface para criação de famílias de objetos relacionados ou dependentes, sem especificar suas classes concretas.

Capítulo 4

ADContexto - Uma Arquitetura Dinâmica Orientada a Contexto

A Arquitetura ADContexto foi concebida com o intuito de apresentar alternativas no tratamento de alguns aspectos relacionados às aplicações colaborativas, descritos no tópico 2.3. Neste Capítulo será apresentada a concepção da arquitetura proposta. Será ressaltado como a arquitetura atende aos aspectos considerados importantes para as aplicações colaborativas (discutidos no Capítulo 2) assim como a mesma faz uso das técnicas da engenharia de software (discutidas no Capítulo 3), notadamente a Orientação a Aspectos e a Programação Reflexiva, para atingir este objetivo. A próxima seção apresenta em mais detalhes a arquitetura proposta neste trabalho.

4.1 Arquitetura do ADContexto

Como pode ser visto na Figura 4.1, a arquitetura proposta para o ADContexto é dividida em três camadas: Aplicação, Controle e Comunicação. Na Camada de aplicação estão os componentes necessários para o fornecimento da interface visual que dará acesso aos serviços. Na Camada de Controle estão os componentes do núcleo da arquitetura, ou seja, os que integram os artefatos importantes para instanciar e gerir o ambiente. E por último, a Camada de Comunicação ou Infra-estrutura, que é responsável por prover a infra-estrutura de comunicação entre os participantes do ambiente. É nesta última camada onde se encontra o adaptador do framework de comunicação a ser utilizado (P2P ou Cliente/Servidor).

Na Figura 4.1, percebe-se a presença de duas camadas externas: interfaces

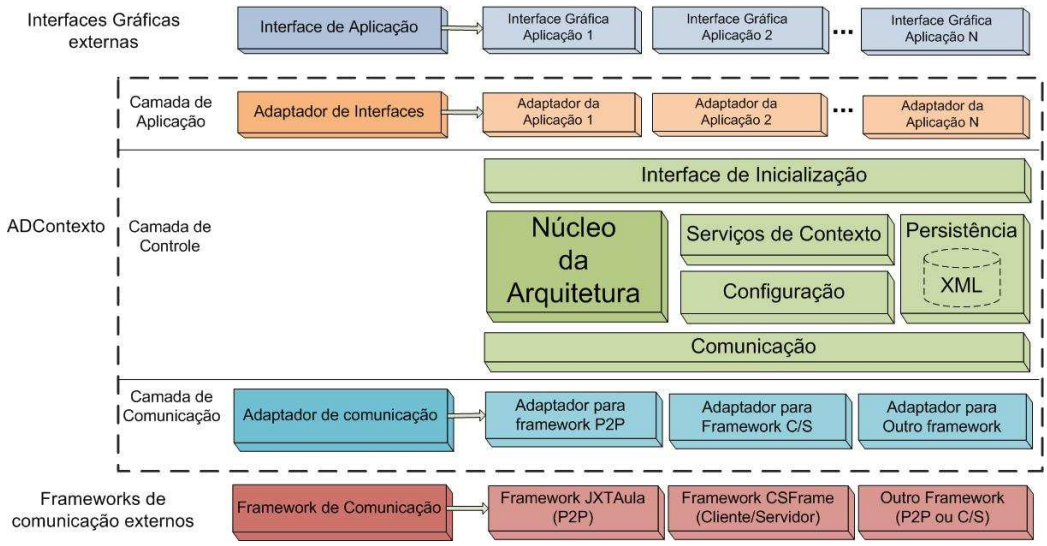


Figura 4.1: Arquitetura do ADContexto

gráficas e frameworks de comunicação. A primeira representa as interfaces visuais que podem ser agregadas pelo ADContexto para a provisão dos serviços fornecidos pelo núcleo da arquitetura. Será visto na seção 4.3.3 que estas interfaces visuais são adicionadas ao ambiente através de um arquivo de configuração. A segunda camada externa, referente à infraestrutura de comunicação, ilustra alguns frameworks que podem ser utilizados para fornecer a dinâmica do ambiente, como a criação, remoção, entrada e saída de grupos. Estes frameworks também são adicionados à arquitetura através do arquivo de configuração especificado pelo ADContexto.

4.2 Camada de Aplicação

A camada de aplicação é composta pelas interfaces visuais que fornecem os serviços para a aplicação que utiliza a arquitetura ADContexto. Os principais elementos dessa camada são: Adaptador de Interface, que é responsável pela comunicação entre a interface visual e o núcleo da arquitetura; e a Interface Gráfica de Serviços, responsável pelo acesso aos serviços de contexto.

4.2.1 Interface Gráfica de Aplicação

O ADContexto tem como um de seus objetivos permitir que interfaces diferentes possam acessar a infra-estrutura do ambiente em paralelo para atender ao requisito de múltiplas interfaces. Tais interfaces foram denominadas de *Interfaces Gráficas de Aplicação* (ou simplesmente Interface de Aplicação), as quais são responsáveis for

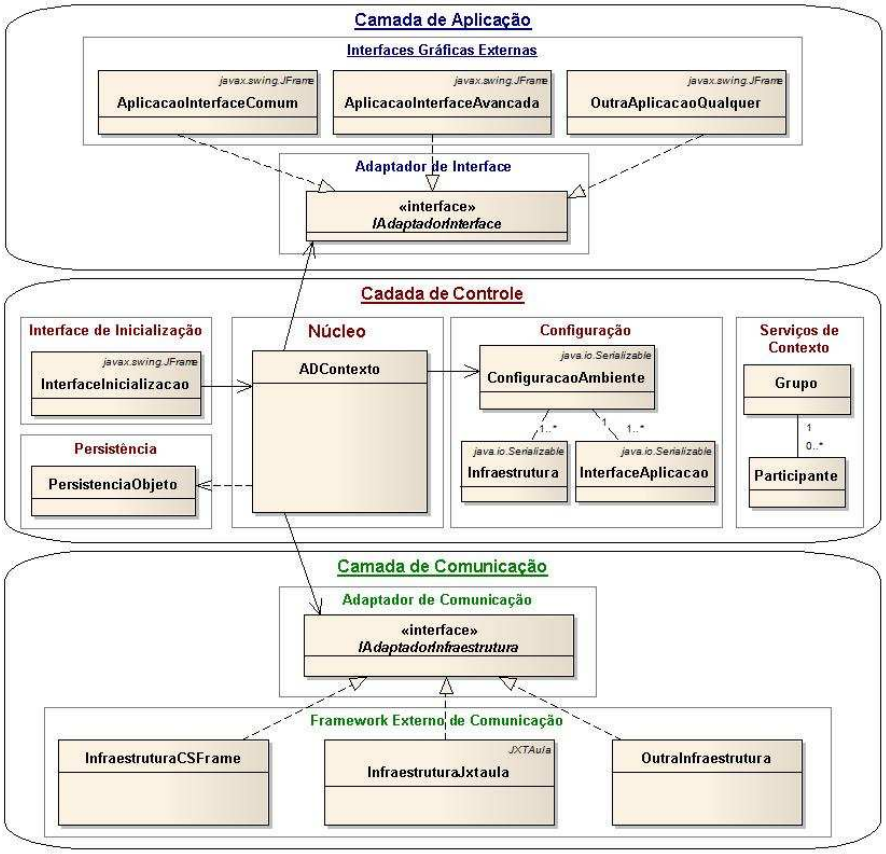


Figura 4.2: Diagrama de Classes do ADContexto

fornecer ao usuário serviços de criação, entrada e saída de grupos, além de permitir o acesso aos serviços específicos de cada grupo. Na figura 4.3 é apresentado um exemplo simplificado desse tipo de interface.

Figura 4.3: Exemplo de Interface Gráfica de Aplicação

Como será mostrado no tópico 4.3.1, a arquitetura proposta neste trabalho permite que sejam agregadas diversas outras interfaces visuais bastando, para isso, a adição de algumas informações em um arquivo de configuração e a implementação do Adaptador de Interface descrito na próxima seção.

4.2.2 Adaptador de Interface gráfica de Aplicação

O principal objetivo do Adaptador de Interface Gráfica é fazer a integração da Interface Visual escolhida com o núcleo da arquitetura. Por exemplo, sempre que ocorrer uma mudança no estado do ambiente - através da criação de um grupo ou da entrada de um novo participante - o adaptador deve comunicar essa mudança à Interface Visual e esta deve refletir visualmente o evento ocorrido.

Para alcançar essa característica, a Arquitetura ADContexto sugere duas formas para a construção dos Adaptadores de Interface Gráfica: Programação Orientada a Aspectos (KICZALES et al., 1997) e Polimorfismo (Orientação a Objetos).

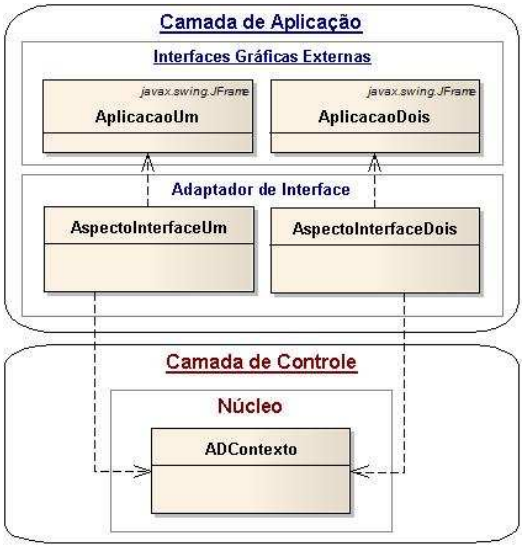


Figura 4.4: Adaptador de Interface usando Aspectos

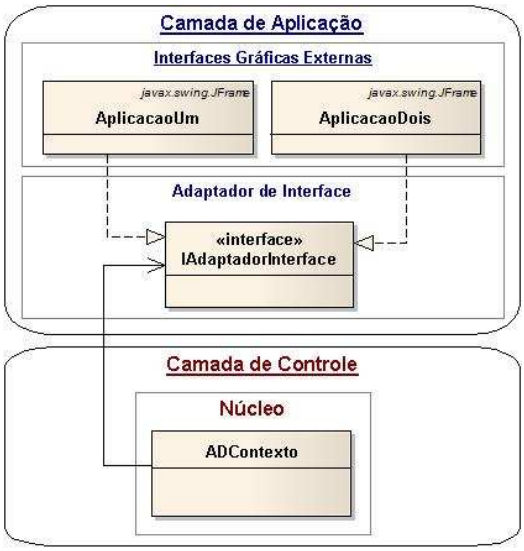


Figura 4.5: Adaptador de Interface usando Polimorfismo

Utilizar POA para a criação de Adaptadores de Aplicação implica dizer que para cada uma dessas aplicações, deve existir um aspecto que "conheça" seus componentes visuais. Na Figura 4.4 é ilustrado um diagrama de classes que demonstra a utilização de Aspectos na construção de tais Adaptadores. Tomando como exemplo o aspecto `AspectoInterfaceUm` dessa figura, sempre que um evento ocorrer no núcleo da arquitetura, esse aspecto fica encarregado de repassar essa informação para a Interface de Aplicação, assim como quando qualquer evento na

aplicação (como um clique de um botão para criar um grupo) é repassado para o núcleo do ADContexto. A vantagem dessa abordagem é o fato de que todo o código que represente algum comportamento na Interface visual fique dentro do Aspecto. Além disso, o Aspecto poderia ser usado para gerar um *log* das mensagens que chegassem à interface.

Em contrapartida, utilizar polimorfismo (Figura 4.5) pode simplificar a construção desses adaptadores, uma vez que as classes que representam as Interfaces de Aplicação irão apenas implementar uma interface que contém alguns métodos conhecidos pelo núcleo da arquitetura. Dessa forma, quando uma Interface de Aplicação for instanciada, o ambiente repassa essa aplicação para o núcleo da arquitetura e este, por sua vez, fica encarregado de repassar as mensagens para a camada de aplicação. A vantagem dessa abordagem gira em torno da simplicidade de código, embora exista a possibilidade de alguns códigos de interface serem implementados juntamente com o núcleo da arquitetura. Uma comparação sobre as vantagens e desvantagens encontradas em cada uma dessas abordagens pode ser vista na Tabela 4.1.

Tabela 4.1: Aspectos x Polimorfismo para Adaptadores de Interface

Abordagem	Vantagens	Desvantagens
Aspectos	- Código de integração entre a aplicação e o núcleo da arquitetura dentro do aspecto - Outras funcionalidades podem ser adicionadas ao aspecto, como a gravação das mensagens que são enviadas a interface visual sem que haja mudança no núcleo da arquietura	- Faz-se necessária a construção de um novo componente (o Aspecto) para cada Aplicação - O desenvolvedor da interface de aplicação deve conhecer bem a linguagem aspectual para criar um aspecto que faça a comunicação da sua interface com o núcleo da arquitetura
Polimorfismo	- Para integrar a interface visual ao núcleo da arquitetura, basta que a aplicação implemente um componente que, na Orientação a Objetos, é conhecido como interface. - Uma vez implementada a aplicação com a interface disponbilizada pelo ADContexto, nenhum outro componente é necessário para a integração	- O código que se refere à atualização de interfaces visuais fica dentro do núcleo da arquitetura - Falta de centralização de código, uma vez que as chamadas aos métodos que atualizam a interface visual ficam espalhados

4.3 Camada de Controle

A camada de controle abrange os componentes da arquitetura responsáveis pelo comportamento do ambiente. Nessa camada estão: os serviços que fornecem persistência das informações, o controle de quais serviços serão fornecidos por um

determinado contexto e os parâmetros de configuração da arquitetura (ver Figura 4.1). A camada de controle se divide em: Interface de Inicialização, Núcleo, Componente de Configuração, Componente de Persistência e Serviços de Contexto.

4.3.1 Interface de Inicialização

Este componente (Figura 4.6) é responsável por fornecer uma interface visual única para iniciar o ambiente. Ele permite escolher qual a interface específica para os serviços e qual a infra-estrutura de comunicação a ser utilizada pelo ambiente. É nesta interface que o usuário informa o seu nome de usuário e senha. Um resumo das informações fornecidas nessa interface pode ser visto na Tabela 4.2.

Figura 4.6: Interface de Inicialização

Tabela 4.2: Informações fornecidas na Interface de Inicialização

Informação	Descrição
Usuário e Senha	Informações de Autenticação para o ambiente
Infra-estrutura de Comunicação	Deve ser fornecido para indicar que framework de comunicação será utilizado pelo ambiente. Necessário para resolver o problema de independência de plataforma de comunicação
Interface de Serviços	Interface específica de serviços a ser utilizada. Necessário para resolver o problema de múltiplas interfaces

4.3.2 Núcleo da Arquitetura

Responsável por integrar os componentes de persistência, comunicação e serviços, o Núcleo da Arquitetura também mantém o controle da configuração do ambiente. A interface de inicialização (seção 4.3.1) do ambiente é responsável por instanciar o núcleo do ADContexto.

Assim que o núcleo é instanciado, o arquivo de configuração contendo as configurações necessárias é carregado pelo componente de configuração.

4.3.3 Componente de Configuração

O Componente de configuração é responsável por carregar a Interface Gráfica e a Infraestrutura de comunicação a serem utilizadas. A listagem 4.1 mostra como é feita essa configuração através de um arquivo XML. Para cada interface de aplicação inserida no ambiente, um novo elemento `<interfaceAplicacao>` deve ser criado. O mesmo vale para os frameworks de comunicação disponíveis no ambiente, sendo que estes devem incluir um novo elemento `<infraestrutura>`.

Listagem 4.1: Trecho do arquivo `ConfiguracaoADContexto.xml`

```
<configuracaoAmbiente>
   <nome>Teste</nome>
   <interfacesAplicacao>
        <interfaceAplicacao>
           <descricao>Interface Basica</descricao>
           <nomeClasse>avcontexto.aplicacao.AplicacaoInterfaceComum</nomeClasse>
        </interfaceAplicacao>
   </interfacesAplicacao>

   <intraestruturas>
      <infraestrutura>
         <descricao>Infraestrutura ClienteServidor</descricao>
         <nomeClasse>avcontexto.comunicacao.InfraestruturaCSFrame</nomeClasse>
         <tipo>ClienteServidor</tipo>
      </infraestrutura>
   </intraestruturas>
</configuracaoAmbiente>
```

4.3.4 Persistência

O componente de persistência é encarregado de armazenar e recuperar informações relativas ao ambiente, como o contexto de um grupo. A arquitetura proposta neste trabalho sugere o uso de arquivos XML para a persistência. A escolha do XML foi motivada pela sua simplicidade e pela disponibilidade de algumas bibliotecas para sua manipulação, incluindo o framework JAXB (JAXB, 2008) (*Java Architecture for XML Binding*), que fornece uma maneira eficiente e padronizada para o mapeamento de classes em elementos XML. Isto não impede, no entanto, o uso de outras formas de persistência, como a utilização de um Sistema de Gerenciamento de Banco de Dados (SGBD), como o *Oracle*(ORACLE, 2008) ou *PostgreSQL*(POSTGRESQL, 2008).

Esse componente é representado pela classe `Persistencia` no diagrama de classes da Figura 4.2 e contém basicamente dois métodos: `lerObjeto` e `gravarObjeto`.

4.3.5 Serviços de Contexto

A principal função do componente Serviços de Contexto na arquitetura é informar para o ambiente o contexto de um determinado grupo. No diagrama de classes do ADContexto apresentado na Figura 4.2, esse componente é representado pela classe `Grupo`. Essa classe pode ser persistida em um arquivo XML utilizando o componente de persistência da seção anterior. Na tabela 4.3 são descritos os principais atributos desta classe.

Tabela 4.3: Atributos da Classe Grupo

Atributos da Classe Sala	Descrição
id	Atributo do tipo `String` que armazena o nome do grupo
coordenadores	Atributo do tipo `List<String>` que armazena uma lista com os nomes dos Coordenadores do Grupo
descricao	Atributo do tipo `String` que armazena uma descrição mais detalhada do grupo
classeChat	Atributo do tipo `String` que aponta para a classe específica para o serviço de Chat
classeCalculo	Atributo do tipo `String` que aponta para a classe específica para o serviço de Cálculo
classeAudio	Atributo do tipo `String` que aponta para a classe específica para o serviço de Áudio
classeVideo	Atributo do tipo `String` que aponta para a classe específica para o serviço de Vídeo
participantes	Atributo do tipo `List<Participante>` que representa todos os participantes pertencentes ao grupo

Os atributos `classeChat`, `classeCalculo`, `classeAudio`, `classeVideo` representam os nomes das classes que serão instanciadas para os serviços de Chat, Cálculo, Áudio e Vídeo, respectivamente. Se na criação de um grupo, o usuário não preencher algum desses atributos, isso implica dizer que esse grupo não fornece tal serviço. Outro aspecto importante a ser observado é que a arquitetura ADContexto não define explicitamente as classes que irão fornecer os serviços, ou seja, grupos diferentes podem fornecer interfaces diferentes para um mesmo tipo de serviço.

Além disso, no momento que um usuário entrar em um grupo, o coordenador desse grupo deve enviar o contexto do ambiente, bem como as classes que irão fornecer cada tipo de serviço, caso esse novo participante não as tenha em sua máquina local.

4.4 Camada de Comunicação

A camada de comunicação contém os componentes necessários para integrar o framework de comunicação a ser utilizado com o núcleo da arquitetura.

4.4.1 Adaptador de Comunicação

Essa adaptador é simplesmente uma interface (classe `IAdaptadorInterface` da Figura 4.2) que deve ser implementada pelos frameworks de comunicação externos. Alguns dos métodos a serem implementados são descritos na Tabela 4.4. Para ver a classe e a assinatura dos métodos veja o Apêndice A.

Tabela 4.4: Alguns métodos da interface IAdaptadorInfraestrutura

Métodos	Descrição
iniciarInfraP2P	Iniciar um framework de comunicação P2P
iniciarInfraCS	Iniciar um framework de comunicação Cliente/Servidor
criarGrupo	Criar um grupo dentro do ambiente
enviarMensagemParticipante	Enviar uma mensagem para um participante específico
enviarMensagemGrupo	Enviar uma mensagem para um grupo específico
entrarGrupo	Implementa o comportamento para entrada em um grupo
sairGrupo	Implementa o comportamento para a saída de um grupo
removerGrupo	Implementa o comportamento para a remoção de um grupo

4.4.2 Framework de Comunicação Externo

O Framework de comunicação externo é responsável por fornecer os serviços de comunicação do ambiente. A interface de inicialização (tópico 4.3.1 é encarregada de ler um arquivo de configuração (tópico 4.3.3) que contém todos os frameworks que podem ser utilizados, os quais devem implementar a interface fornecida pelo ADContexto, descrita no tópico anterior. A implementação utilizada por esse framework é transparente para a arquitetura ADContexto, uma vez que o ambiente não sabe como está implementado o componente de comunicação. A vantagem dessa abordagem é permitir que os serviços fornecidos por uma aplicação através do ADContexto possam ser testados em diferentes infra-estruturas de comunicação (P2P ou C/S).

4.4.3 Considerações Finais

A arquitetura mostrada neste capítulo (veja Figuras 4.1 e 4.2) especificou uma abordagem para construir uma aplicação colaborativa que possa atender aos critérios

descritos na seção 2.3. Através da programação reflexiva, a arquitetura sugeriu como os serviços dinâmicos podem ser instanciados e como a infra-estrutura de comunicação pode ser selecionada. O uso de aspectos na construção dos Adaptadores de Interface sugere a separação dos interesses sistêmicos (atualização da interface, por exemplo) dos interesses funcionais. O próximo capítulo mostrará um protótipo desenvolvido utilizando a arquitetura proposta e como essas técnicas ajudaram a alcançar tais objetivos.

Capítulo 5

Um Ambiente Virtual Orientado a Contexto

Visando avaliar os conceitos propostos na arquitetura apresentada no Capítulo 4, foi desenvolvido um protótipo de um ambiente colaborativo denominado AVContexto. Neste Capítulo, serão mostrados detalhes relativos a implementação do protótipo e como os aspectos considerados importantes em aplicações colaborativas (veja tópico 2.3) puderam ser implementados.

5.1 Tecnologias utilizadas

As técnicas de engenharia de software utilizadas neste trabalho, mais especificamente o uso da orientação a aspectos e a programação reflexiva, foram aplicadas com o intuito de atender os aspectos considerados importantes para os Ambientes Virtuais Colaborativos discutidos no Tópico 2.3. Nas próximas seções serão mostradas as tecnologias utilizadas, como por exemplo as API´s `Java Reflection` e `AspectJ`.

5.1.1 Java Reflection

A motivação para o uso da `API Java Reflection` se baseou nas características da linguagem Java, que envolve:

- **Portabilidade:** por ser uma linguagem interpretada, o Java pode ser executado em qualquer plataforma ou equipamento que possua um

interpretador Java, e que tenha sido especialmente compilado para o sistema a ser utilizado;

- **Facilidade:** o Java é derivado da linguagem C e C++, sendo assim familiar. Além disso, o ambiente retira do programador a responsabilidade de gerenciar a memória e os ponteiros.

Uma das características principais da arquitetura ADContexto é a flexibilidade de escolha das interfaces visuais e dos frameworks de comunicação na implementação do AVContexto, essa flexibilidade (descrita no Capítulo 3) pôde ser alcançada através do uso da computação reflexiva (WIKI, 2008), mais precisamente com o uso da API `Java Reflection`. Outra API poderia ser usada para esse fim, como por exemplo a `cglib` (CGLIB, 2008). No entanto, esta API não é nativa do java (linguagem escolhida para a implementação do protótipo), o que adiciona mais uma dependência na arquitetura, enquanto que a API `Java Reflection` é instalada juntamente com os pacotes de desenvolvimento do Java.

5.1.2 AspectJ

Atualmente existem diversas opções para a utilização da programação orientada a aspectos. Neste trabalho, adotou-se o `AspectJ` como linguagem de aspectos e a linguagem Java como linguagem de componentes. O `AspectJ` é uma extensão para a linguagem de programação Java. Abaixo seguem algumas razões que levaram à escolha dessa API:

- **Compatibilidade com o Java:** todo programa Java válido é também um programa `AspectJ` válido;
- **Compatibilidade de Plataforma:** Todo programa `AspectJ` pode ser executado em uma máquina virtual java (JVM);
- **Baixa curva de aprendizado:** Ao programar com `AspectJ`, o programador deve sentir-se como se estivesse utilizando uma extensão da linguagem Java;

A estrutura de um aspecto em `AspectJ` pode ser vista na Figura 5.1. O Apêndice B mostra mais detalhes em relação a configuração e uso dessa linguagem de aspectos..

Na listagem 5.1 é apresentado um trecho do aspecto criado para uma das interfaces de aplicação disponibilizadas no protótipo. A linha 7 da

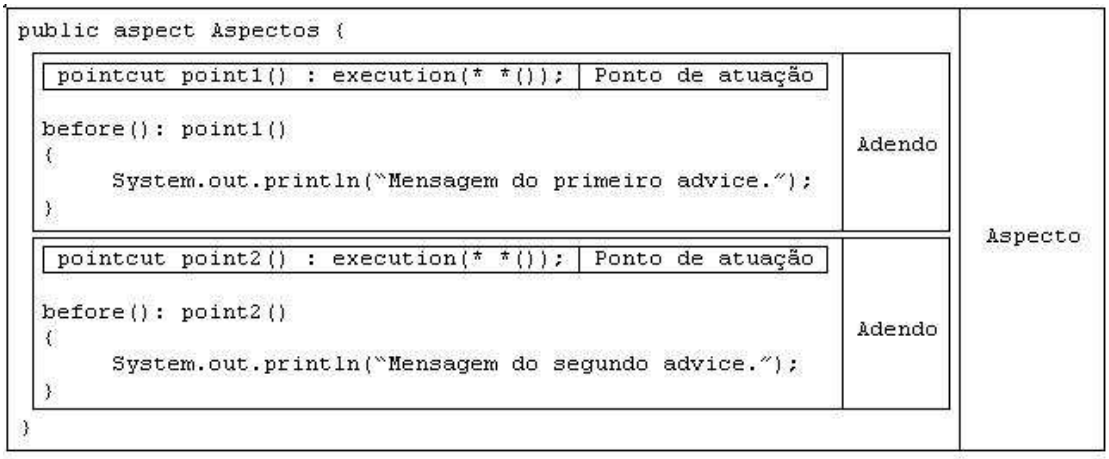


Figura 5.1: Estrutura de um aspecto (WINCK; JUNIOR, 2006)

listagem 5.1 cria um ponto de atuação chamado `removerSala` que será disparado sempre que o método `btnRemoverSalaActionPerformed(..)` da classe `AplicacaoInterfaceComum` for executado. As linhas de 11 a 13 especificam o que deve ser feito depois (palavra chave `after`) que o método for executado. Nesse caso a remoção de uma sala da infraestrutura do ambiente.

Listagem 5.1: Trecho do arquivo AspectoInterfaceComum.java

```
1  public aspect AspectoInterfaceComum {
2
3     private AVContexto ambiente;
4
5     /** REMOVER SALA */
6
7     pointcut removerSala(AplicacaoInterfaceComum aplicacao) :
8        execution (private void btnRemoverSalaActionPerformed(..))
9          && target(aplicacao);
10
11    after(AplicacaoInterfaceComum aplicacao) : removerSala(aplicacao) {
12       ambiente.getInfra().removerSala(salasAmbiente.getSelectedValue());
13    }
14 }
```

5.1.3 CSFrame

Para validar a possibilidade de utilizar a aplicação AVContexto sobre uma infra-estrutura de comunicação centralizada, foi desenvolvido um framework Cliente/Servidor denominado CSFrame, que fornece algumas funcionalidades necessárias para colaboração em um ambiente virtual. Dentre as funcionalidades providas pelo framework, as principais são:

- Criação e Remoção de grupos de trabalho
- Entrada e Saída de Grupos
- Recuperação dos Participantes do Ambiente
- Recuperação das Salas do Ambiente
- Recuperação dos participantes de uma determinada Sala

Na Figura 5.2 é mostrado o diagrama de classes do framework CSFrame. A tabela 5.1 descreve as principais classes desse framework. Apenas duas delas não são nativas do CSFrame: `IAdaptadorInfraestrutura`, que é a interface especificada pela arquitetura ADContexto que deve ser implementada pelos frameworks de comunicação; e a classe `InfraestruturaCSFrame`, que foi criada para implementar a interface `IAdaptadorInfraestrutura`.

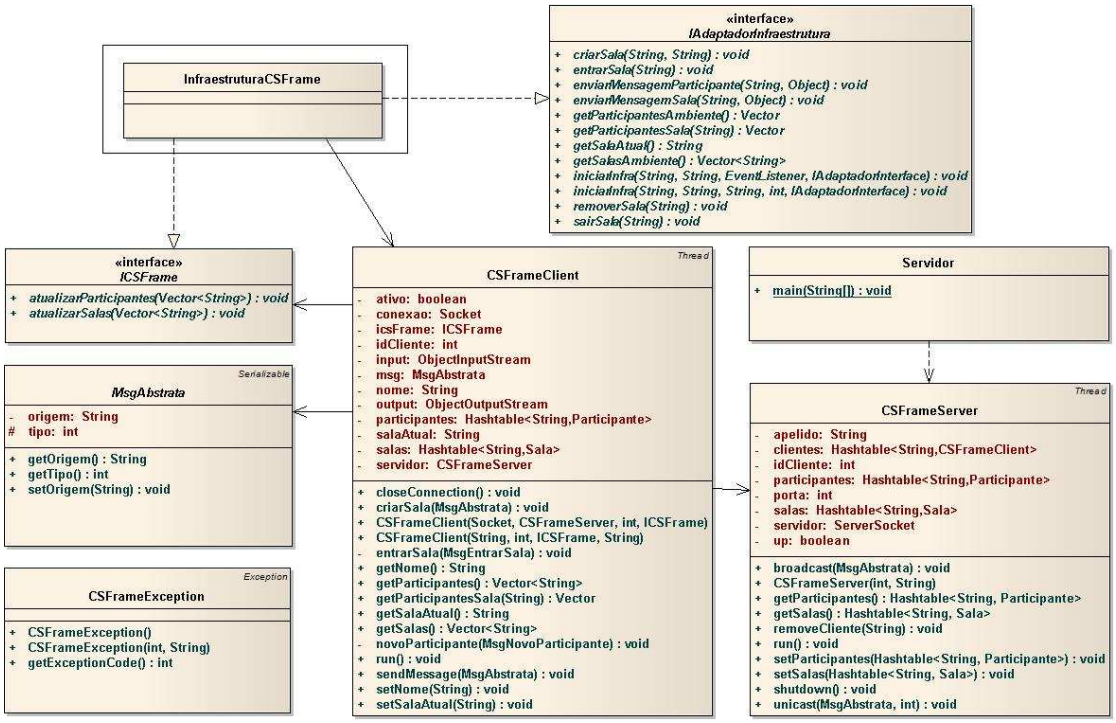


Figura 5.2: Arquitetura do CSFrame

Mais detalhes de como o framework foi concebido podem ser encontrados no Apêndice C, onde é disponbilizada uma listagem de todo o código-fonte do CSFrame.

Tabela 5.1: Principais classes do Framework CSFrame

Classe	Descrição
`Servidor`	Uma classe simples que contém um método `main` para iniciar o Servidor
`CSFrameServer`	Classe responsável por esperar o recebimento de novas conexões
`CSFrameClient`	Classe que age como cliente, enviando e recebendo mensagens do Servidor
`MsgAbstrata`	Superclasse de todas as classes de mensagens do framework. As principais são: `MsgClienteConectado`, `MsgCriarSala`, `MsgEntrarSala`, `MsgSairSala`, `MsgEnviarAmbiente`, `MsgIdCliente`, `MsgNovoParticipante`, `TipoMensagem`.
`CSFrameException`	Classe utilizada para disparar exceções do framework

5.1.4 JXTAula

No contexto da infra-estrutura P2P, a camada de comunicação do protótipo utilizou o framework JXTAula, uma abordagem baseada em P2P para sistemas de ensino a distância. A principal motivação para a escolha deste framework foi o trabalho realizado por (SAMPAIO et al., 2008), onde se criou uma nova arquitetura de comunicação P2P para o ambiente COGEST (SOARES, 2004), descrito no tópico 2.2.4. O JXTAula foi concebido em camadas, onde cada camada tem uma responsabilidade específica, funcionando como divisões lógicas de componentes agrupados para:

i. Reduzir a complexidade, uma vez que os componentes são agrupados com o objetivo de simplificar a comunicação entre os mesmos;

ii. Reduzir dependência e acoplamento, pois a utilização de interfaces bem definidas evita dependências entre os componentes de camadas distintas;

iii. Favorecer a coesão, levando-se em consideração que componentes de responsabilidades relacionadas estão agrupados dentro da mesma camada;

iv. Promover a reusabilidade, pois as camadas podem ser reutilizadas em outros sistemas;

Segundo (SAMPAIO et al., 2008), esta plataforma utiliza uma arquitetura que fornece as funcionalidades relacionadas com a criação de um ambiente colaborativo capaz de organizar seus participantes em salas virtuais. Dentro dessas salas, os participantes são capazes de localizar uns aos outros, bem como estabelecer

comunicação entre si. Finalmente, um participante do ambiente pode enviar mensagens a todos os participantes de uma determinada sala ou a outro participante em particular, além de compartilhar arquivos com os participantes que estejam no ambiente.

5.2 AVContexto - Um Ambiente Multi-interface, com Serviços Dinâmicos e Independente de Infraestrutura.

5.2.1 Interface de Inicialização do ambiente

Como discutido no tópico 4.3.1, a Arquitetura ADContexto disponibiliza na camada de controle uma interface de inicialização para o ambiente. Assim que o núcleo da arquitetura é instanciado (classe `ADContexto.java`), a interface de inicialização é visualizada. É nesse momento que o participante escolhe qual interface visual utilizar e qual o framework de comunicação irá fornecer a infra-estrutura para o ambiente. Nas figuras 5.3, 5.4, 5.5 e 5.6 são ilustrados quatro configurações diferentes de inicialização que podem ser adotadas ao iniciar o ambiente.

- **Inicialização 1:** Framework JXTAula com a Interface Visual 1
- **Inicialização 2:** Framework CSFrame com a Interface Visual 1
- **Inicialização 3:** Framework JXTAula com a Interface Visual 2
- **Inicialização 4:** Framework CSFramw com a Interface Visual 2

5.3 Escolhendo o Framework de Comunicação

Um ponto importante a ser ressaltado é que, quando um dos frameworks de comunicação é Cliente/Servidor, a interface disponibiliza duas caixas de texto - "Ip do Servidor" e "Porta" - para permitir a comunicação com a máquina que faz o papel de servidor. Essa informação deve ser fornecida dentro do arquivo de configuração. A listagem 5.2 representa o trecho do código do arquivo de configuração que define os frameworks de comunicação disponíveis. Na linha 06 desta listagem é mostrado que o framework "Infraestrutura CSFrame (C/S)" é do tipo "Cliente/Servidor",

Figura 5.3: Tipo de Inicialização 1

Figura 5.4: Tipo de Inicialização 2

Figura 5.5: Tipo de Inicialização 3

Figura 5.6: Tipo de Inicialização 4

portanto precisará especificar endereço e porta do servidor. A linha 12 da listagem abaixo define que o framework "Infraestrutura JXTAula (P2P)" é do tipo P2P.

Listagem 5.2: Configuração da Infraestrutura de comunicação

```
1  <configuracaoAmbiente>
2  ...
3     <infraestrutura>
4        <descricao>Infraestrutura CSFrame (C/S)</descricao>
5        <nomeClasse>avcontexto.comunicacao.InfraestruturaCSFrame</nomeClasse>
6        <tipo>ClienteServidor</tipo>
7     </infraestrutura>
8
9     <infraestrutura>
10       <descricao>Infraestrutura JXTAula (P2P)</descricao>
11       <nomeClasse>avcontexto.comunicacao.InfraestruturaPeerToPeer</nomeClasse>
12       <tipo>P2P</tipo>
13    </infraestrutura>
14 ...
15 </configuracaoAmbiente>
```

O código completo do arquivo de configuração pode ser visto no Apêndice D, tópico D.1.1.

5.4 Escolhendo a Interface Visual

Uma vez que o tipo de inicialização tenha sido definido, o ambiente é iniciado com o clique no botão "Iniciar Ambiente", presente no canto inferior da interface de inicialização. Nas Figuras 5.7 e 5.8 é mostrado o ambiente iniciado com as duas interfaces disponibilizadas.

Figura 5.7: Interface Visual 1

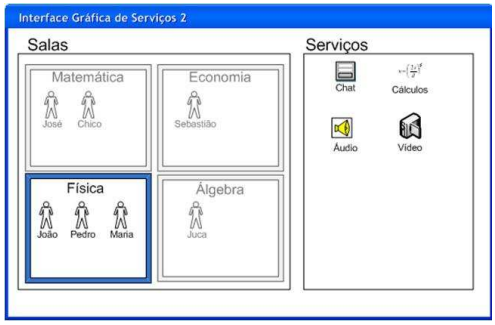


Figura 5.8: Interface Visual 2

A arquitetura ADContexto permite que um usuário altere a interface gráfica em tempo de execução. Isso é interessante quando se precisa reservar recursos de máquina para outros processos. Portanto, somente será preciso que o participante mude para uma interface mais simples e que exija menos recursos de máquina.

Outra vantagem desta abordagem é que usuários diferentes podem acessar o ambiente com interfaces diferentes, o que permite a integração e colaboração entre participantes com necessidades especiais ou não.

5.5 Serviços Orientados a Contexto

Na Tabela 5.2 são descritas as funcionalidades fornecidas pela interface da figura 5.7. Nesse exemplo, o ambiente possui quatro grupos criados: Matemática, Física, Economia e Álgebra. O campo "Sala Atual" indica qual o grupo em que o usuário está, nesse caso, o grupo de física. Percebe-se que apenas os serviços de Chat e Cálculo estão disponíveis para este grupo, ou seja, o ambiente pode fornecer mais serviços, mas apenas estes dois foram liberados pelo criador do grupo.

No Capítulo 4 foi explorada a questão dos serviços dinâmicos dentro de um contexto. Na Figura 5.9 é mostrada como é feita a criação de um grupo no protótipo

Tabela 5.2: Serviços fornecidos pela Interface Visual 1

Funcionalidade	Descrição
Botão Entrar	Faz com que o usuário entre no grupo selecionado
Botão Sair	Faz com que o usuário saia do grupo selecionado
Botão Criar	Permite que o usuário crie grupos e indique que tipos de serviços serão fornecidos por ele
Botão Chat	Instancia a tela de chat para o grupo selecionado
Botão Cálculos	Instancia a tela para cálculos do grupo selecionado

e como é possível configurar qual interface será executada para um determinado serviço.

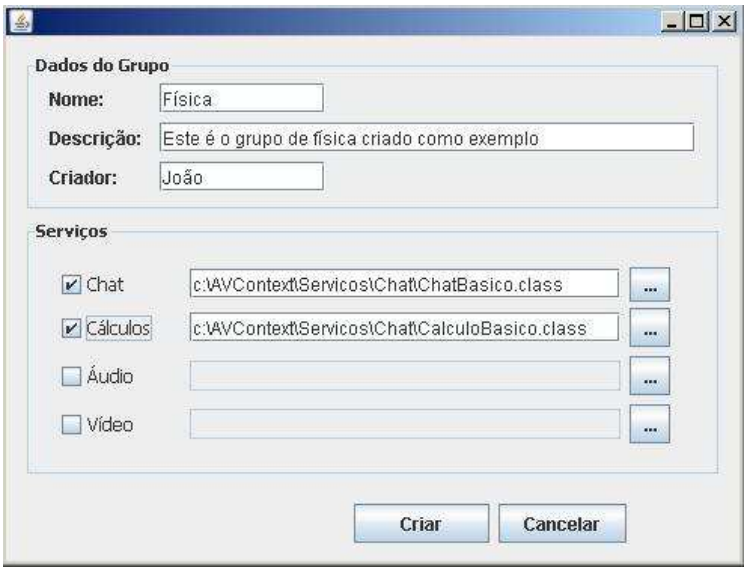


Figura 5.9: Exemplo de tela para criar um grupo e configurar os serviços

Quando um grupo é criado, o usuário que o criou deve informar que serviços serão fornecidos pelo grupo em questão. No entanto, além do ambiente permitir que o criador do grupo escolha os serviços (veja Figura 5.9), também é possível que seja indicado o objeto que será instanciado quando um determinado serviço for disparado. No exemplo dessa figura, quando o serviço de *Chat* for acionado dentro do contexto de *Física*, a classe `ChatBasico.class` será executada. Essa classe contém um método `main` que instancia uma aplicação de chat.

Ao clicar no botão "Criar", será criado localmente um arquivo XML chamado "Fisica.xml"(listagem 5.3), que armazena as informações de contexto da Sala.

Listagem 5.3: Arquivo Fisica.xml

```
1 <sala id="Fisica">
2    <descricao>Este é um grupo de física criado como exemplo</descricao>
```

```
3     <criador>João</criador>
4     <classeChat>c:\AVContext\Servicos\Chat\ChatBasico.class</classeChat>
5     <classeCalculo>c:\AVContext\Servicos\Chat\CalculoBasico.class </classeCalculo>
6     <classeAudio>c:\AVContext\Servicos\Chat\AudioBasico.class</classeCalculo>
7     <classeVideo>c:\AVContext\Servicos\Chat\VideoBasico.class</classeCalculo>
8
9     <participantes>
10        <nome>João</nome>
11        <isCoordenador>true</isCoordenador>
12    </participantes>
13 </sala>
```

Quando um novo participante entrar nesse grupo, uma mensagem será enviada para todos os outros participantes desse grupo. Quando essa mensagem chegar ao coordenador, este envia automaticamente as configurações do grupo para o novo participante, bem como os arquivos necessários para executar os serviços disponibilizados.

5.6 Resultados Obtidos

Como resultado da implementação do protótipo, verificou-se a possibilidade de desenvolver aplicações mais flexíveis no que se refere ao uso de diferentes interfaces visuais. Foi possível verificar a viabilidade de alterar entre interfaces diferentes sem que o participante perca o contexto do ambiente, já que a infra-estrutura do ambiente permanece a mesma (figuras 5.7 e 5.8). Além disso, a independência de infra-estrutura do AVContexto mostrou ser uma alternativa para a decisão sobre qual plataforma de comunicação utilizar. Isso permitiu, por exemplo, que a aplicação desenvolvida no protótipo pudesse alternar entre um framework de comunicação mais centralizado, como o CSFrame (Cliente/Servidor), ou mais distribuído, como o JXTAula (P2P).

Outro ponto que se mostrou bastante interessante foi o fato de que os participantes de um contexto tivessem a possibilidade de alterar as interfaces específicas de um serviço dentro do contexto (Serviços Dinâmicos, seção 4.3.5). Essa possibilidade permitiu aos participantes uma flexibilidade maior, pois eles próprios passaram a ter participação na montagem do ambiente, o que eleva o conceito de colaboração não somente ao conteúdo que é produzido pela ferramenta, mas a própria construção da ferramenta. Isso se mostrou muito interessante pelo fato de que os próprios participantes tinham o poder de refinar os serviços fornecidos. Tomando o exemplo da classe `ChatBasico.class`, os colaboradores do

ambiente poderiam fazer melhorias nessa classe e disponibilizar para ser utilizada pelo contexto. Nesse caso, um controle de permissões se faria necessário para saber quem teria direito a fazer essas mudanças.

5.7 Considerações finais

Neste Capítulo, foi apresentado o protótipo utilizado com a finalidade de avaliar a arquitetura descrita no Capítulo 4. Foi mostrado como uma aplicação que usa a arquitetura ADContexto pode se beneficiar com a implementação dos aspectos relevantes levantados no Capítulo 2 (tópico 2.3). Foi mostrado como a inicialização do ambiente é feita e como escolher a interface visual de serviços, bem como a infraestrutura de comunicação a ser utilizada. Para a interface visual dos serviços, o protótipo fez uso de uma interface básica (Figura 5.7). Também foi apresentado o uso da ferramenta AspectJ para a construção dos adaptadores de interface utilizando Aspectos (KICZALES et al., 1997) em Java e, por fim, os resultados obtidos foram devidamente enumerados. O próximo capítulo se destina a apresentar uma conclusão geral sobre o trabalho, as contribuições e trabalhos futuros.

Capítulo 6

Conclusão

Neste trabalho, alguns ambientes virtuais de colaboração (AulaNet, Teleduc, Moodle e COGEST) foram analisados mantendo-se o foco em aspectos ligados a: Interface, Contextualização de Serviços e Infra-estrutura de comunicação (tópico 2.2). Com base nessa análise foram indentificados aspectos que poderiam ser tratados com uma maior atenção, dentre os quais: múltiplas interfaces de acesso ao ambiente, contextualização de serviços, serviços dinâmicos e independência de infra-estrutura de comunicação (tópico 2.3).

Quatro aspectos foram considerados importantes no contexto deste trabalho:

- O uso de múltiplas interfaces torna possível uma integração ainda maior entre usuários de um ambiente colaborativo. Pessoas com algum tipo de limitação física ou tecnológica podem necessitar de interfaces específicas para interagir no ambiente;
- A possibilidade de participantes de ambientes colaborativos escolherem os serviços a serem utilizados em um determinado contexto;
- A Escolha de diferentes implementações para um mesmo tipo de serviço em contextos diferentes;
- A necessidade de se manter um certo grau de independência em relação à infra-estrutura de comunicação a ser utilizada pelo ambiente virtual de aprendizagem (P2P, Cliente/Servidor ou Híbrida).

Foi realizada uma busca à luz da engenharia de software com o objetivo de encontrar mecanismos que pudessem auxiliar na elaboração de uma solução que tratasse os aspectos considerados importantes, no âmbito deste trabalho, para a concepção e construção de ambientes virtuais de colaboração. A Programação Orientada a Aspectos e a Programação Reflexiva (Capítulo 3) se mostraram uma solução viável no atendimento de tais requisitos.

Como resultado dessa reflexão, foi concebida uma Arquitetura Dinâmica Orientada a Contexto, aqui chamada de ADContexto. A arquitetura possibilita o uso de interfaces múltiplas através da criação de "Adaptadores de interface" utilizando a Programação Orientada a Aspectos (tópico 4.2.2). A arquitetura também possui um núcleo que permite a contextualização de serviços e o uso de serviços dinâmicos, além da independência de infra-estrutura de comunicação, ambas usando a computação reflexiva (tópicos 4.3.5 e 4.4.1).

Para avaliar os conceitos utilizados na concepção da arquitetura proposta neste trabalho, foi implementado um protótipo de um ambiente virtual de colaboração denominado AVContexto (capítulo 6). Esse protótipo permitiu avaliar se os mecanismos disponibilizados pela arquitetura possibilitavam a implementação dos aspectos levantados no capítulo 2. Notadamente, o uso de interfaces múltiplas, de serviços dinâmicos orientados a contexto e independência de infra-estrutura de comunicação.

6.1 Contribuições e Resultados

Dentre as principais contribuições e resultados obtidos neste trabalho, pode-se destacar:

i. Identificação de aspectos considerados importantes na concepção e construção de uma aplicação colaborativa, como a Independência de Interfaces, Serviços Orientados a Contexto e Independência de Infra-estrutura de comunicação. Isso, através da análise de alguns ambientes colaborativos existentes;

ii. Identificação de soluções baseadas em engenharia de software para tratar aspectos que melhorem a colaboração em ambientes virtuais com a adoção de técnicas como Programação Orientada a Aspectos e Programação Reflexiva;

iii. Disponibilização de uma arquitetura que possa ser utilizada para a construção de ambientes colaborativos diversos, incluindo aqueles que levam em

consideração pessoas com necessidades especiais, contextualização de serviços e liberdade de escolha da plataforma de comunicação;

iv. Disponibilização de um ambiente de colaboração que seja utilizado e aprimorado através do desenvolvimento de um protótipo que faz uso da arquitetura proposta neste trabalho.

6.2 Limitações e trabalhos futuros

O presente trabalho ainda apresenta limitações e melhorias que serão devidamente tratadas em trabalhos futuros:

- Desenvolver um mecanismo de integração entre aplicações que façam uso da arquitetura ADContexto, focando aquelas que possuem infra-estruturas de comunicação diferentes;
- Integrar a arquitetura proposta neste trabalho (ADContexto) ao ambiente desenvolvido no projeto COGEST, visando avaliar o uso da arquitetura com uma interface visual avançada.
- Avaliar o uso da arquitetura para ambientes colaborativos voltados para o campo da telemedicina, onde alguns dos serviços a serem disponibilizados possuem um alto grau de complexidade.
- Criação de um framework Cliente/Servidor denominado CSFrame que pode ser utilizando por outras aplicações colaborativas que necessitem de uma infraestrutura centralizada.

Estes problemas possuem soluções complexas, necessitando de uma reflexão bem maior na busca de uma solução. Por isso, os mesmos foram deixados como trabalhos futuros.

6.3 Considerações finais

Os desafios encontrados durante o desenvolvimento deste trabalho mostraram como os Ambientes Virtuais de Aprendizagem ainda podem evoluir para permitir uma maior colaboração entre seus participantes. Este trabalho espera ter alcançado seu objetivo ao disponibilizar uma arquitetura que tenha a possibilidade de ser

adotada por diferentes tipos de aplicações colaborativas, em especial as que buscam: diversidade de interfaces visuais, serviços orientados a contexto e independência de infra-estrutura de comunicação.

Apêndice A

Classes do ADContexto

Este apêndice se destina a apresentar as principais classes da arquitetura ADContexto.

A.1 Camada de Aplicação

A.1.1 Classe `IAdaptadorInterface.java`

```
1  package avcontexto.aplicacao;
2  import java.util.List;
3  import java.util.Vector;
4  import javax.swing.DefaultListModel;
5  import avcontexto.controle.nucleo.AVContexto;
6  import avcontexto.controle.servicos.Grupo;
7
8  public interface IAdaptadorInterface {
9      public void iniciarInterface(AVContexto pAmbiente);
10     public void entrarGrupo(String grupo);
11     public void sairGrupo();
12     public void removerGrupo(String grupo);
13     public void setAmbiente(AVContexto ambiente);
14     public AVContexto getAmbiente();
15     public void atualizarGrupos(Vector<String> grupos);
16     public void atualizarParticipantesAmbiente(Vector<String> participantes);
17     public void atualizarParticipantesGrupo(Vector<String> participantes);
18     public void atualizaNome(String nome);
19 }
```

A.2 Camada de Controle

A.2.1 Núcleo

Classe `AVContexto.java`

```
1  package avcontexto.controle.nucleo;
2  import java.io.FileNotFoundException;
3  import java.util.Hashtable;
4
5  import avcontexto.aplicacao.IAdaptadorInterface;
6  import avcontexto.aplicacao.InterfaceInicializacao;
7  import avcontexto.comunicacao.IAdaptadorInfraestrutura;
8  import avcontexto.controle.configuracao.ConfiguracaoAmbiente;
9  import avcontexto.controle.servicos.Ambiente;
10 import avcontexto.controle.servicos.Participante;
11 import avcontexto.controle.servicos.Grupo;
12 import avcontexto.controle.persistencia.PersistenciaObjeto;
13
14 public class AVContexto  {
15
16     private IAdaptadorInfraestrutura adapInfra;
17     private IAdaptadorInterface adapInterface;
18     private ConfiguracaoAmbiente configuracaoAmbiente;
19     private String arquivoConfig;
20
21     private Hashtable<String, Grupo> grupos;
22
23    public AVContexto(String arquivoConfiguracao) {
24       try {
25          this.arquivoConfig = arquivoConfiguracao;
26          configuracaoAmbiente = (ConfiguracaoAmbiente)PersistenciaObjeto.lerObjeto
27                (arquivoConfiguracao, ConfiguracaoAmbiente.class);
28       } catch (FileNotFoundException e) {e.printStackTrace();
29       }
30    }
31    public void iniciarAmbienteP2P(String nomeParticipante, String senhaParticipante,
32             IAdaptadorInfraestrutura infra, IAdaptadorInterface inter)
33                throws Exception {
34       this.adapInfra = infra;
35       this.adapInterface = inter;
36       this.adapInfra.iniciarInfra(nomeParticipante, senhaParticipante, null,
37             this.adapInterface);
38       this.adapInterface.iniciarInterface(this);
39    }
40    public void iniciarAmbienteCS(String nomeParticipante, String senhaParticipante,
41             IAdaptadorInfraestrutura infra, IAdaptadorInterface inter,
42                String servidor, int porta)   throws Exception {
43       this.adapInfra = infra;
44       this.adapInterface = inter;
45       this.adapInterface.iniciarInterface(this);
46       this.adapInfra.iniciarInfra(nomeParticipante, senhaParticipante, servidor,
```

```
47              porta, this.adapInterface);
48      }
49      public IAdaptadorInfraestrutura getInfra() {return adapInfra;}
50
51      public ConfiguracaoAmbiente getConfiguracaoAmbiente() {
52          return configuracaoAmbiente;
53      }
54
55      public void setConfiguracaoAmbiente(ConfiguracaoAmbiente configuracaoAmbiente) {
56          this.configuracaoAmbiente = configuracaoAmbiente;
57      }
58
59      public static void main(String args[]) {
60          java.awt.EventQueue.invokeLater(new Runnable() {
61              public void run() {
62               String s = "/Paulo/Mestrado/ArPPAV/src/avcontexto/aplicacao/ConfiguracaoAvcontexto.xml";
63                  new InterfaceInicializacao(new AVContexto(s)).setVisible(true);
64              }
65          });
66      }
67
68      public Hashtable getGrupos() {
69          if (grupos == null) {
70              grupos = new Hashtable();
71          }
72          return grupos;
73      }
74
75      public void criarGrupo(Grupo grupo) {
76          try {
77              getGrupos().put(grupo.getId(), grupo);
78              getInfra().criarSala(grupo.getId(), grupo.getDescricao());
79          } catch (Exception e) {
80              // TODO Auto-generated catch block
81              e.printStackTrace();
82          }
83      }
84
85      public void addServico(String grupo, Class classe) {
86
87      }
88
89      public void trocarInterface() {
90
91      }
92  }
```

A.2.2 Persistência

Classe `PersistenciaObjeto.java`

```
1  package avcontexto.controle.persistencia;
```

```
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import javax.xml.bind.JAXBContext;
8  import javax.xml.bind.JAXBException;
9  import javax.xml.bind.Marshaller;
10 import javax.xml.bind.Unmarshaller;
11
12 public class PersistenciaObjeto {
13    public static Object lerObjeto(String arquivoXML, Class name)
14          throws FileNotFoundException {
15       JAXBContext context;
16       try {
17          Object objeto;
18          context = JAXBContext.newInstance(new Class[]{name});
19          Unmarshaller unmarshaller = context.createUnmarshaller();
20          objeto = unmarshaller.unmarshal(new File(arquivoXML));
21          return objeto;
22       } catch (JAXBException e) {
23          e.printStackTrace();
24          return null;
25       }
26    }
27
28    public static void gravarObjeto(Object objeto, String arq, Class name)
29          throws FileNotFoundException {
30       JAXBContext context;
31       try {
32          context = JAXBContext.newInstance(new Class[]{name});
33          Marshaller marshaller = context.createMarshaller();
34          marshaller.marshal(objeto, new FileWriter(arq));
35       } catch (JAXBException e) {
36          e.printStackTrace();
37       } catch (IOException e) {
38          e.printStackTrace();
39       }
40    }
41 }
```

A.2.3 Configuração

Classe `ConfiguracaoAmbiente.java`

```
1  package avcontexto.controle.configuracao;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import javax.xml.bind.annotation.XmlAccessorType;
6  import javax.xml.bind.annotation.XmlElement;
7  import javax.xml.bind.annotation.XmlElementWrapper;
```

```
8  import javax.xml.bind.annotation.XmlRootElement;
9  import javax.xml.bind.annotation.XmlAccessType;
10
11 @XmlRootElement(name="configuracaoAmbiente")
12 @XmlAccessorType(XmlAccessType.FIELD)
13 public class ConfiguracaoAmbiente extends ClassePersistida
14    implements java.io.Serializable{
15
16    private static final long serialVersionUID = 1L;
17
18    @XmlElement(name="nome")
19    private String nome;
20
21    @XmlElementWrapper(name="interfacesAplicacao")
22    private List<InterfaceAplicacao> interfaceAplicacao =
23          new ArrayList<InterfaceAplicacao>();
24
25    @XmlElementWrapper(name="intraestruturas")
26    private List<Infraestrutura> infraestrutura = new ArrayList<Infraestrutura>();
27
28    public List<InterfaceAplicacao> getInterfaceAplicacao() {
29       return interfaceAplicacao;
30    }
31    public void setInterfaceAplicacao(List<InterfaceAplicacao> interfaceAplicacao) {
32       this.interfaceAplicacao = interfaceAplicacao;
33    }
34    public String getNome() {return nome;}
35
36    public void setNome(String nome) {this.nome = nome;}
37    public List<Infraestrutura> getInfraestrutura() {return infraestrutura;}
38    public void setInfraestrutura(List<Infraestrutura> infraestrutura) {
39       this.infraestrutura = infraestrutura;
40    }
41 }
```

Classe `Infraestrutura.java`

```
1  package avcontexto.controle.configuracao;
2
3  import javax.xml.bind.annotation.XmlAccessType;
4  import javax.xml.bind.annotation.XmlAccessorType;
5  import javax.xml.bind.annotation.XmlElement;
6
7  @XmlAccessorType(XmlAccessType.FIELD)
8  public class Infraestrutura implements java.io.Serializable{
9
10    private static final long serialVersionUID = 1L;
11
12    @XmlElement(name="descricao")
13    private String descricao;
14
15    @XmlElement(name="nomeClasse")
```

```
16      private String nomeClasse;
17
18      @XmlElement(name="tipo")
19      private String tipo;
20
21      public String getDescricao() {return descricao;}
22      public void setDescricao(String descricao) {this.descricao = descricao;}
23      public String getNomeClasse() {return nomeClasse;}
24      public void setNomeClasse(String nomeClasse) {this.nomeClasse = nomeClasse;}
25      public String getTipo() {return tipo;}
26      public void setTipo(String tipo) {this.tipo = tipo;}
27  }
```

Classe `InterfaceAplicacao.java`

```
1   package avcontexto.controle.configuracao;
2   import javax.xml.bind.annotation.XmlAccessType;
3   import javax.xml.bind.annotation.XmlAccessorType;
4   import javax.xml.bind.annotation.XmlElement;
5
6   @XmlAccessorType(XmlAccessType.FIELD)
7   public class InterfaceAplicacao implements java.io.Serializable {
8
9       private static final long serialVersionUID = 1L;
10
11      @XmlElement(name="descricao")
12      private String descricao;
13
14      @XmlElement(name="nomeClasse")
15      private String nomeClasse;
16
17      public String getDescricao() {return descricao;}
18      public void setDescricao(String descricao) {this.descricao = descricao;}
19      public String getNomeClasse() {return nomeClasse;}
20      public void setNomeClasse(String nomeClasse) {this.nomeClasse = nomeClasse;}
21  }
```

A.2.4 Serviços de Contexto

Classe `Grupo.java`

```
1   package avcontexto.controle.servicos;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import javax.xml.bind.annotation.XmlAccessType;
6   import javax.xml.bind.annotation.XmlAccessorType;
7   import javax.xml.bind.annotation.XmlAttribute;
8   import javax.xml.bind.annotation.XmlElement;
9   import javax.xml.bind.annotation.XmlElementWrapper;
10  import javax.xml.bind.annotation.XmlRootElement;
```

```
11
12 @XmlRootElement(name="grupo")
13 @XmlAccessorType(XmlAccessType.FIELD)
14 public class Grupo {
15
16     @XmlAttribute(name="id")
17     private String id;
18
19     @XmlElement(name="coordenador")
20     private String coordenador;
21
22     @XmlElement(name="descricao")
23     private String descricao;
24
25     @XmlElement(name="classeChat")
26     private String classeChat;
27
28     @XmlElement(name="classeCalculo")
29     private String classeCalculo;
30
31     @XmlElement(name="classeAudio")
32     private String classeAudio;
33
34     @XmlElement(name="classeVideo")
35     private String classeVideo;
36
37     @XmlElementWrapper(name="participantes")
38     private List<Participante> participante = new ArrayList<Participante>();
39
40     public Grupo(){}
41
42     public String getClasseAudio() {return classeAudio;}
43
44     public void setClasseAudio(String classeAudio) {
45         this.classeAudio = classeAudio;
46     }
47
48     public String getClasseCalculo() {return classeCalculo;}
49
50     public void setClasseCalculo(String classeCalculo) {
51         this.classeCalculo = classeCalculo;
52     }
53     public String getClasseChat() {return classeChat;}
54     public void setClasseChat(String classeChat) {this.classeChat = classeChat;}
55     public String getClasseVideo() {return classeVideo;}
56     public void setClasseVideo(String classeVideo) {this.classeVideo = classeVideo;}
57     public String getCoordenador() {return coordenador;}
58     public void setCoordenador(String coordenador) {this.coordenador = coordenador;}
59     public String getDescricao() {return descricao;}
60     public void setDescricao(String descricao) {this.descricao = descricao;}
61     public String getId() {return id;}
62     public void setId(String id) {this.id = id;}
63     public List<Participante> getParticipante() {return participante;}
```

```
64      public void setParticipante(List<Participante> participante) {
65          this.participante = participante;
66      }
67  }
```

Classe `Participante.java`

```
1   package avcontexto.controle.servicos;
2
3   import javax.xml.bind.annotation.XmlAccessType;
4   import javax.xml.bind.annotation.XmlAccessorType;
5   import javax.xml.bind.annotation.XmlElement;
6
7   @XmlAccessorType(XmlAccessType.FIELD)
8   public class Participante {
9
10      @XmlElement(name="nome")
11      private String nome;
12
13      @XmlElement(name="isCoordenador")
14      private boolean isCoordenador;
15
16      public Participante() {}
17
18      public Participante(String nome, boolean isCoordenador) {
19          this.nome = nome;
20          this.isCoordenador = isCoordenador;
21      }
22
23      public boolean isCoordenador() {return isCoordenador;}
24
25      public void setCoordenador(boolean isCoordenador) {
26          this.isCoordenador = isCoordenador;
27      }
28
29      public String getNome() {return nome;}
30
31      public void setNome(String nome) {this.nome = nome;}
32  }
```

A.3 Camada de Comunicação

Classe `IAdaptadorInfraestrutura.java`

```
1   package avcontexto.comunicacao;
2
3   import java.util.EventListener;
4   import java.util.Vector;
5   import avcontexto.aplicacao.IAdaptadorInterface;
6
```

```
7  public interface IAdaptadorInfraestrutura {
8
9      public void iniciarInfra(String nomeParticipante, String senhaParticipante,
10             EventListener evento, IAdaptadorInterface adaptadorInter) throws Exception;
11     public void iniciarInfra(String nomeParticipante, String senhaParticipante,
12             String servidor, int porta, IAdaptadorInterface adaptadorInter)
13                 throws Exception;
14     public void criarSala(String nomeSala, String descSala) throws Exception;
15      public void entrarSala(String nomeSala) throws Exception;
16      public void enviarMensagemParticipante(String nomeParticipante, Object mensagem)
17             throws Exception;
18      public void enviarMensagemSala(String nomeSala, Object mensagem)
19             throws Exception;
20      public Vector getParticipantesAmbiente() throws Exception;
21      public Vector getParticipantesSala(String nomeSala) throws Exception;
22      public Vector<String> getSalasAmbiente() throws Exception;
23      public void sairSala(String nomeSala) throws Exception;
24      public void removerSala(String nomeSala) throws Exception;
25      public String getSalaAtual() throws Exception;
26 }
```

Apêndice B

Instalação e configuração do AspectJ

B.1 Instalação do AspectJ

O AspectJ, tanto para windows quanto para linux, é distribuído gratuitamente. O pacote a ser instalado para ambos os sistemas é o mesmo.

Para efetuar o download do instalador do AspectJ, basta acessar o endereço de download *http://www.eclipse.org/aspectj/downloads.php*. Na página de download, procure pela última versão estável do AspectJ, conforme ilustrado na figura B.1. A versão 1.6.1 foi a última versão estável na época que este trabalho foi escrito.

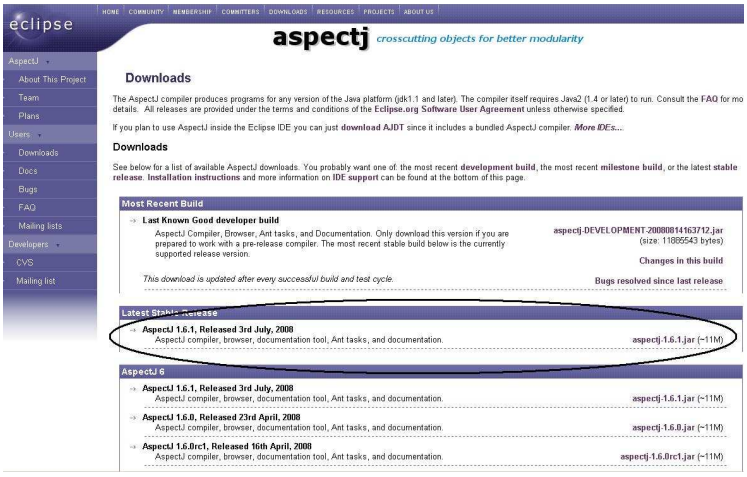


Figura B.1: Download do AspectJ

Assim que o download do arquivo for realizado, abra uma sessão do terminal (Prompt do MS-DOS no windows e Shell no Linux) e execute o comando `java`

`-jar aspectj-1.6.1.jar` para abrir a tela inicial de instalação do AspectJ (figura B.2). Clique em `Next`.

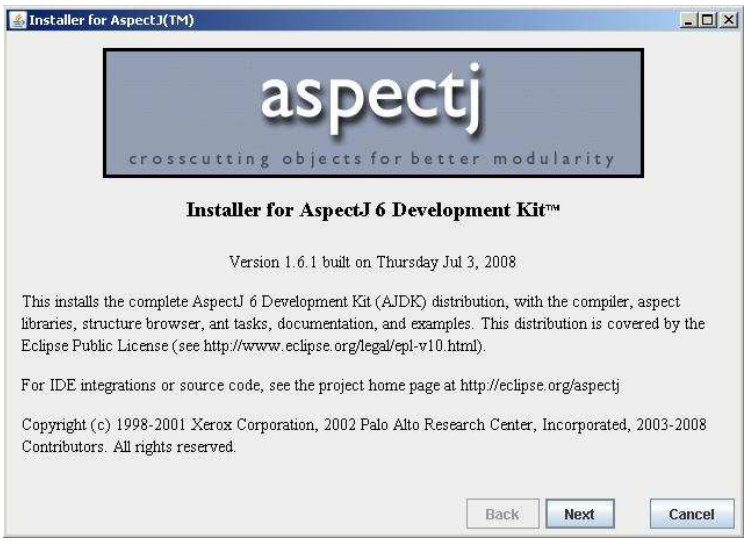


Figura B.2: Tela inicial de instalação do AspectJ

Agora será preciso informar o caminho de instalação da máquina virtual Java, como ilustrado na figura B.3. Depois de informar o local clique em `Next` novamente.

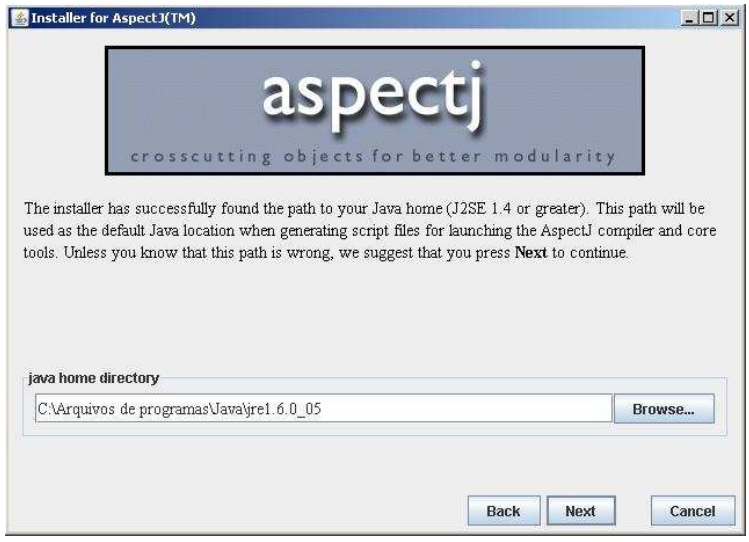


Figura B.3: Informar diretório da JRE

O próximo passo é escolher o diretório onde serão instalados os arquivos necessários para AspectJ (figura B.4). Clique no botão `Browse` para escolher o caminho desenhado. Após a escolha, clique no botão `Install`.

Quando a barra que indica o progresso da cópia dos arquivos estiver preenchida,

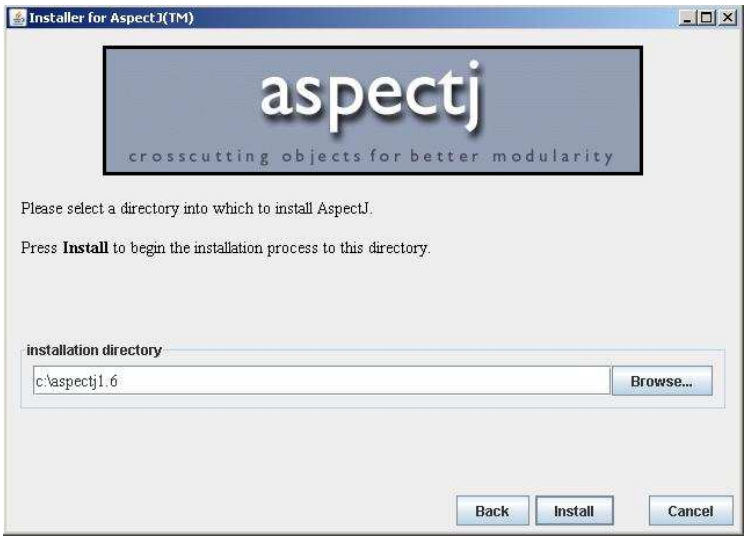


Figura B.4: Escolha do Diretório de Instalação do AspectJ

clique em `Next` novamente. Por fim, a tela da figura B.5. Clique em `Finish` para concluir a instalação.

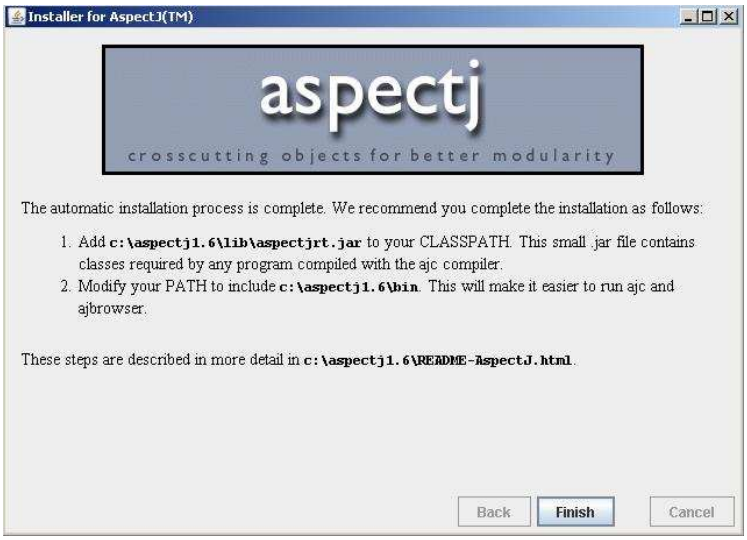


Figura B.5: Finalização da Instalação do AspectJ

B.2 Configurando o AspectJ no Windows

Para o seu correto funcionamento, o AspectJ precisa que uma nova variável seja criada e que outras duas sejam alteradas. São elas: `ASPECT_HOME`, `CLASSPATH` e `PATH`.

Para definir estas variáveis de ambiente no sistema operacional Windows, acesse

as propriedades do item Meu Computador, na aba avançados, e acesse a opção variáveis de ambiente. Deverão ser efetuadas as seguintes configurações:

- Variável `ASPECTJ_HOME`, com o valor `c:`
  `aspectj1.6`, caso a instalação tenha sido feita neste diretório.
- Variável `CLASSPATH`: acrescente ao valor dessa variável o parâmetro: `;%ASPECT_HOME/lib/aspectjrt.jar`
- Variável `PATH`, acrescente ao valor dessa variável o parâmetro `;%ASPECTJ_HOME%`
  `bin`

B.3 Configurando o AspectJ no Linux

Caso a instalação tenha sido efetuada no sistema operacional Linux, a configuração será um pouco diferente. O arquivo /etc/profile deve ser editado, conforme detalhado a seguir:

- Variável `ASPECTJ_HOJE`, com o valor /usr/local/aspectj, caso a instalação tenha sido efetuada nesse diretório.
- Variável `CLASSPATH`: acrescente ao valor dessa variável o parâmetro: `;%ASPECT_HOME/lib/aspectjrt.jar`
- Variável `PATH`, acrescente ao valor dessa variável o parâmetro `;%ASPECTJ_HOME%`
  `bin`

B.4 Testando a Instalação

Após a conclusão da instalação e também da configuração do AspectJ, pode ser feito um teste para saber o AspectJ está configurado corretamente. A seguir, será apresentada uma aplicação bastante simples (um "Alô Mundo"), extraída do livro "AspectJ: Programação Orientada a Aspectos com Java" (WINCK; JUNIOR, 2006).

Utilizando um editor de textos qualquer, crie os arquivos AloMundo.java e TesteAloMundo.java, apresentados nas listagens das tabelas

Tabela B.1: AloMundo.java

```
01 public class AloMundo {
02    public void exibeMensagem {
03       System.out.println("Alô Mundo - Versão AspectJ");
04    }
05
06    public static void main (String args[])
07       AloMundo alo = new AloMundo();
08       alo.exibeMensagem();
09    }
11 }
```

Tabela B.2: TesteAloMundo.java

```
01 public aspect TesteAloMundo {
02    pointcut callExibeMensagem():  call(public void AloMundo.*(..));
03
04    before():  callExibeMensagem() {
05       System.out.println("Antes da execução do método exibeMensagem()");
06    }
07
08    after():  callExibeMensagem() {
09       System.out.println("Depois da execução do método exibeMensagem()");
11    }
11 }
```

Os dois arquivos devem ser compilados com o compilador do AspectJ, com o comando `ajc AloMundo.java TesteAloMundo.java`. Depois, basta executar o arquivo AloMundo com o comando `java AloMundo`. O resultado que aparecer no console deve ser semelhante ao seguinte:

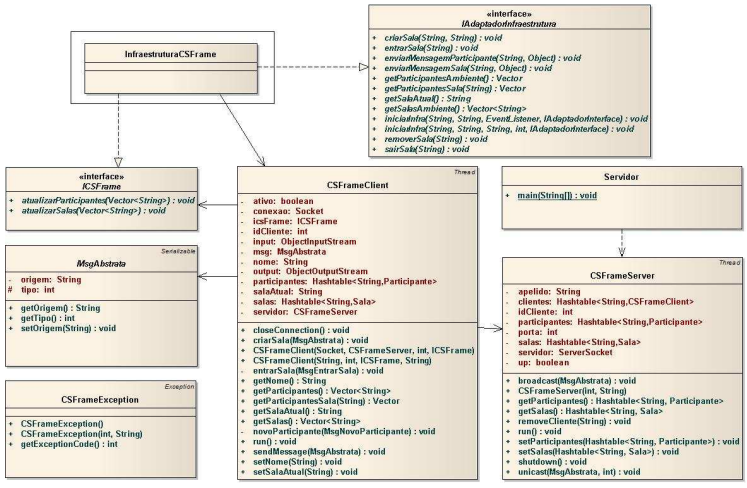
Resultado que deve aparecer no console:

```
Antes da execução do método exibeMensagem()
Alô Mundo - Versão AspectJ
Depois da execução do método exibeMensagem()
```

Apêndice C

Framework Cliente/Servidor para Comunicação

C.1 Diagrama de Classes



C.2 Classes do Núcleo

C.2.1 CSFrameServer

```
1  package csframe;
2
3  import java.io.*;
4  import java.net.*;
5  import java.util.Enumeration;
6  import java.util.Hashtable;
7  import java.util.Vector;
```

```
8
9  import csframe.mensagens.MsgAbstrata;
10 import csframe.servicos.Participante;
11 import csframe.servicos.Sala;
12
13 public class CSFrameServer extends Thread {
14
15    private ServerSocket servidor;
16    private Hashtable<String,CSFrameClient> clientes;
17    private Hashtable<String,Sala> salas;
18    private Hashtable<String,Participante> participantes;
19    private Hashtable<String,Vector<String>> partSala;
20    private int idCliente, porta;
21    private String apelido;
22    private boolean up;
23
24
25    public CSFrameServer (int porta, String apelido) throws CSFrameException {
26       try {
27          this.porta = porta;
28          System.out.println("[Atendendo na porta "+ porta + "]");
29          this.salas = new Hashtable<String, Sala>();
30          this.participantes = new Hashtable<String, Participante>();
31          this.partSala = new Hashtable<String, Vector<String>>();
32          servidor = new ServerSocket(porta);
33          clientes = new Hashtable<String,CSFrameClient>();
34          idCliente = 0;
35          System.out.println("[Servidor " + apelido + " ativado]");
36          this.apelido = apelido;
37       } catch (IOException e) {
38          throw new CSFrameException(CSFrameException.SOCKET,
39                "ServerSocket não pode ser criado");
40       }
41    }
42
43    public void run () {
44       try {
45          up = true;
46          while (true) {
47             Socket s = servidor.accept();
48             System.out.println("Conexão com cliente recebida");
49             if (!up) break;
50             CSFrameClient cliente = new CSFrameClient(s, this, ++idCliente, null);
51             cliente.start();
52             clientes.put(""+idCliente, cliente);
53          }
54       } catch (Exception e) {
55          System.out.println("ERRO: Conexão não pode ser efetivada\n");
56       }
57    }
58
59    public void shutdown() {
60       try {
```

```
61            up = false;
62            new Socket("localhost", porta);
63         } catch (Exception e) {
64            System.out.println("ERRO: Thread servidora não pode ser conectada\n");
65         }
66      }
67
68      public synchronized void broadcast(MsgAbstrata msg) throws CSFrameException {
69         for (Enumeration e = clientes.elements(); e.hasMoreElements();) {
70            ((CSFrameClient) e.nextElement()).sendMessage(msg);
71         }
72      }
73
74      public synchronized void unicast(MsgAbstrata msg, int pIdCliente)
75            throws CSFrameException {
76         clientes.get(""+pIdCliente).sendMessage(msg);
77      }
78
79      public synchronized void removeCliente(String id) {
80         clientes.remove(id);
81      }
82
83      public Hashtable<String, Participante> getParticipantes() {
84         return participantes;
85      }
86
87      public void setParticipantes(Hashtable<String, Participante> participantes) {
88         this.participantes = participantes;
89      }
90
91      public Hashtable<String, Sala> getSalas() {
92         return salas;
93      }
94
95      public void setSalas(Hashtable<String, Sala> salas) {
96         this.salas = salas;
97      }
98
99      public Hashtable<String, Vector<String>> getPartSala() {
100        if (partSala==null) {
101           partSala = new Hashtable();
102        }
103        return partSala;
104     }
105
106     public void setPartSala(Hashtable<String, Vector<String>> partSala) {
107        this.partSala = partSala;
108     }
109
110  }
```

C.2.2 CSFrameClient

```
1  package csframe;
2
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.net.Socket;
7  import java.util.Hashtable;
8  import java.util.Iterator;
9  import java.util.Map;
10 import java.util.Vector;
11 import javax.swing.DefaultListModel;
12 import csframe.mensagens.MsgAbstrata;
13 import csframe.mensagens.MsgCriarSala;
14 import csframe.mensagens.MsgEntrarSala;
15 import csframe.mensagens.MsgEnviarAmbiente;
16 import csframe.mensagens.MsgIdCliente;
17 import csframe.mensagens.MsgNovoParticipante;
18 import csframe.mensagens.MsgSairSala;
19 import csframe.mensagens.TipoMensagem;
20 import csframe.servicos.Participante;
21 import csframe.servicos.Sala;
22
23 public class CSFrameClient extends Thread {
24
25    private Socket conexao;
26    private CSFrameServer servidor;
27    private int idCliente;
28    private ObjectInputStream input;
29    private ObjectOutputStream output;
30    private boolean ativo;
31    private ICSFrame icsFrame;
32
33    private MsgAbstrata msg;
34    private String salaAtual;
35
36    private Hashtable<String,Sala> salas;
37    private Hashtable<String,Participante> participantes;
38    private Hashtable<String,Vector<String>> partSala;
39    private String nome;
40
41    public CSFrameClient(Socket conexao, CSFrameServer servidor, int idCliente,
42          ICSFrame icsFrame) throws CSFrameException {
43       try {
44          this.conexao = conexao;
45          this.servidor = servidor;
46          this.idCliente = idCliente;
47          this.nome = "Servidor";
48          output = new ObjectOutputStream(conexao.getOutputStream());
49          output.flush();
50          output.writeObject(null);
51          input = new ObjectInputStream(conexao.getInputStream());
```

```
52          salas = new Hashtable();
53          participantes = new Hashtable();
54          this.icsFrame = icsFrame;
55       } catch (IOException e) {
56          e.printStackTrace();
57          throw new CSFrameException(CSFrameException.STREAM,
58                   "Canais de I/O não foram criados");
59       }
60    }
61
62    public CSFrameClient(String srvIP, int srvPort, ICSFrame icsFrame, String nome)
63          throws CSFrameException {
64       try {
65          this.icsFrame = icsFrame;
66          this.nome = nome;
67          conexao = new Socket(srvIP, srvPort);
68          servidor = null;
69          input = new ObjectInputStream(conexao.getInputStream());
70          output = new ObjectOutputStream(conexao.getOutputStream());
71          output.flush();
72          MsgNovoParticipante msg = new MsgNovoParticipante(this.getNome(),
73                idCliente);
74          output.writeObject(msg);
75          salas = new Hashtable();
76          participantes = new Hashtable();
77       } catch (IOException e) {
78          e.printStackTrace();
79          throw new CSFrameException(CSFrameException.STREAM,
80                "Canais de I/O não foram criados");
81       }
82    }
83
84    public void run() {
85       try {
86          ativo = true;
87          while(ativo){
88             msg = (MsgAbstrata)input.readObject();
89             if (msg != null) {
90                if (servidor ==null) {
91                   System.out.println(nome+" recebeu mensagem de "+msg.getOrigem());
92                   switch (msg.getTipo()) {
93                   case TipoMensagem.MSG_CRIAR_SALA:
94                      criarSala((MsgCriarSala)msg);
95                      break;
96                   case TipoMensagem.MSG_ENTRAR_SALA:
97                      entrarSala((MsgEntrarSala)msg);
98                      break;
99                   case TipoMensagem.MSG_SAIR_SALA:
100                     sairSala((MsgSairSala)msg);
101                     break;
102                  case TipoMensagem.MSG_NOVO_PARTICIPANTE:
103                     novoParticipante((MsgNovoParticipante)msg);
104                     break;
```

```
105                     case TipoMensagem.MSG_ID_CLIENTE:
106                         this.idCliente =
107                             Integer.parseInt(((MsgIdCliente)msg).getIdCliente());
108                         System.out.println("Cliente recebeu ID "+idCliente);
109                         break;
110                     case TipoMensagem.MSG_ENVIAR_AMBIENTE:
111                         MsgEnviarAmbiente msgEnviar = (MsgEnviarAmbiente)msg;
112                         this.salas = msgEnviar.getSalas();
113                         this.partSala = msgEnviar.getPartSala();
114                         icsFrame.atualizarParticipantesAmbiente(getParticipantes());
115                         icsFrame.atualizarSalas(getSalas());
116                         break;
117                     }
118                 } else {
119
120                     switch (msg.getTipo()) {
121                     case TipoMensagem.MSG_CRIAR_SALA:
122                         MsgCriarSala msgCriar = (MsgCriarSala)msg;
123                         servidor.getSalas().put(msgCriar.getSala(),
124                                 new Sala(msgCriar.getSala()));
125                         servidor.getPartSala().put(msgCriar.getSala(), new Vector());
126                         servidor.broadcast(msg);
127                         break;
128                     case TipoMensagem.MSG_ENTRAR_SALA:
129                         MsgEntrarSala msgEntrar = (MsgEntrarSala)msg;
130                         servidor.getPartSala().get(msgEntrar.getSala()).add(msgEntrar.getNomeParticipant
131
132                         servidor.broadcast(msgEntrar);
133                         break;
134                     case TipoMensagem.MSG_SAIR_SALA:
135                         MsgSairSala msgSair = (MsgSairSala)msg;
136                         servidor.getPartSala().get(msgSair.getSala()).remove((msgSair.getParticipante()
137
138                         servidor.broadcast(msgSair);
139                         break;
140                     case TipoMensagem.MSG_NOVO_PARTICIPANTE:
141                         MsgNovoParticipante msgNovo = (MsgNovoParticipante)msg;
142                         servidor.getParticipantes().put(msgNovo.getNome(),
143                                 new Participante(msgNovo.getNome(), null));
144
145                         MsgEnviarAmbiente msgAmbiente1 =
146                             new MsgEnviarAmbiente(servidor.getSalas(),
147                                                   servidor.getParticipantes(),servidor.getPartSala());
148                         servidor.broadcast(msgAmbiente1);
149                         break;
150                     default:
151                         servidor.broadcast(msg);
152                         break;
153                     }
154                 }
155             }
156         }
157     } catch (Exception e) {
```

```
158             System.out.println("ERRO: Canal de I/O não pode ser lido\n");
159             e.printStackTrace();
160         }
161     }
162
163     private void novoParticipante(MsgNovoParticipante msg) {
164         Participante p = new Participante(msg.getNome(), null);
165         this.participantes.put(msg.getNome(), p);
166         if (icsFrame != null) {
167             icsFrame.atualizarParticipantesAmbiente(getParticipantes());
168         }
169     }
170
171     private void entrarSala(MsgEntrarSala pMsg) {
172         getPartSala().get(pMsg.getSala()).add(pMsg.getNomeParticipante());
173         if (getSalaAtual().equals(pMsg.getSala())){
174             if (icsFrame != null) {
175                 icsFrame.atualizarParticipantesGrupo(getPartSala().get(pMsg.getSala()));
176             }
177         }
178     }
179
180     private void sairSala(MsgSairSala pMsg) {
181         getPartSala().get(pMsg.getSala()).remove(pMsg.getParticipante());
182         if (icsFrame != null) {
183             if (pMsg.getParticipante().equals(getNome())) {
184                 icsFrame.atualizarParticipantesGrupo(new Vector());
185             } else
186                 icsFrame.atualizarParticipantesGrupo(getPartSala().get(pMsg.getSala()));
187
188         }
189     }
190
191     public void sendMessage(MsgAbstrata msg) throws CSFrameException {
192         try {
193             output.writeObject(msg);
194         } catch (Exception e) {
195             throw new CSFrameException(CSFrameException.STREAM,
196                     "Não foi possível enviar mensagem");
197         }
198     }
199
200     public void closeConnection() throws CSFrameException {
201         try {
202             ativo = false;
203             String s = conexao.getInetAddress().toString();
204             input.close(); output.close();
205             conexao.close();
206             if (servidor != null) {
207                 servidor.removeCliente(""+idCliente);
208             }
209         } catch (IOException e) {
210             throw new CSFrameException(CSFrameException.SOCKET,
```

```
211                 "Conexão não pode ser fechada");
212         }
213     }
214 
215     public void criarSala(MsgAbstrata pMsg) {
216         MsgCriarSala msgCriarSala = (MsgCriarSala)pMsg;
217         this.salas.put(msgCriarSala.getSala(), new Sala(msgCriarSala.getSala()));
218         getPartSala().put(msgCriarSala.getSala(), new Vector());
219         if (icsFrame != null) {
220             icsFrame.atualizarSalas(getSalas());
221         }
222     }
223 
224     public Vector<String> getSalas() {
225         DefaultListModel v = new DefaultListModel();
226         Map<String, Sala> map = salas;
227         Iterator it = map.values().iterator();
228 
229         Vector<String> vet = new Vector();
230         while (it.hasNext()) {
231             Sala s = (Sala) it.next();
232             v.addElement(s.getNomeSala());
233             vet.add(s.getNomeSala());
234         }
235         return vet;
236     }
237 
238     public Vector<String> getParticipantes() {
239         DefaultListModel v = new DefaultListModel();
240         Map<String, Participante> map = participantes;
241         Iterator it = map.values().iterator();
242         Vector<String> vet = new Vector();
243         while (it.hasNext()) {
244             Participante p = (Participante) it.next();
245             v.addElement(p.getNome());
246             vet.add(p.getNome());
247         }
248         return vet;
249     }
250 
251     public Vector<String> getParticipantesSala(String sala) {
252         return partSala.get(sala);
253     }
254 
255     public void setSalaAtual(String nomeSala) {
256         this.salaAtual = nomeSala;
257     }
258 
259     public String getSalaAtual() {
260         if (salaAtual == null) {
261             salaAtual = new String();
262         }
263         return salaAtual;
```

```
264     }
265
266     public String getNome() {
267         return nome;
268     }
269
270     public void setNome(String nome) {
271         this.nome = nome;
272     }
273
274     public Hashtable<String, Vector<String>> getPartSala() {
275         if (partSala == null) {
276             partSala = new Hashtable();
277         }
278         return partSala;
279     }
280
281     public void setPartSala(Hashtable<String, Vector<String>> partSala) {
282         this.partSala = partSala;
283     }
284 }
```

C.3 Classe de Interface

C.3.1 ICSFrame.java

```
1  package csframe;
2
3  import java.util.Vector;
4
5  public interface ICSFrame {
6      public void atualizarSalas(Vector<String> salas);
7      public void atualizarParticipantesAmbiente(Vector<String> part);
8      public void atualizarParticipantesGrupo(Vector<String> part);
9  }
```

C.4 Classes de Negócio

C.4.1 Participante.java

```
1  package csframe.servicos;
2
3  import java.io.Serializable;
4  import java.net.Socket;
5
6  public class Participante implements Serializable{
7      private String nome;
8      private Socket conexao;
```

```
9
10    public Participante(String nome, Socket conexao) {
11       this.nome = nome;
12       this.conexao = conexao;
13    }
14    public Socket getConexao() {return conexao;}
15    public void setConexao(Socket conexao) {this.conexao = conexao;}
16    public String getNome() {return nome;}
17    public void setNome(String nome) {this.nome = nome;}
18 }
```

C.4.2 Sala.java

```
1  package csframe.servicos;
2
3  import java.io.Serializable;
4  import java.util.Vector;
5
6  public class Sala implements Serializable {
7
8     private String nomeSala;
9     private Vector<Participante> participantesSala;
10
11    public Sala(String nomeSala) {
12       this.nomeSala = nomeSala;
13    }
14    public String getNomeSala() {
15       return nomeSala;
16    }
17    public void setNomeSala(String nomeSala) {
18       this.nomeSala = nomeSala;
19    }
20    public Vector<Participante> getParticipantesSala() {
21       return participantesSala;
22    }
23    public void setParticipantesSala(Vector<Participante> participantesSala) {
24       this.participantesSala = participantesSala;
25    }
26
27 }
```

C.5 Mensagens

C.5.1 TipoMensagem.java

```
1  package csframe.mensagens;
2
3  public class TipoMensagem {
4
5     public static final int MSG_CLIENTE_CONECTADO=1;
```

```
6       public static final int MSG_CLIENTE_DESCONECTADO=2;
7       public static final int MSG_CRIAR_SALA=3;
8       public static final int MSG_GET_SALAS=4;
9       public static final int MSG_ENTRAR_SALA=5;
10      public static final int MSG_NOVO_PARTICIPANTE = 6;
11      public static final int MSG_ID_CLIENTE = 7;
12      public static final int MSG_ENVIAR_AMBIENTE = 8;
13      public static final int MSG_SAIR_SALA = 9;
14
15  }
```

C.5.2 MsgAbstrata.java

```
1   package csframe.mensagens;
2
3   import java.io.Serializable;
4
5   public abstract class MsgAbstrata implements Serializable {
6
7       private String origem;
8       protected int tipo;
9
10      public int getTipo() {
11          return tipo;
12      }
13
14      public String getOrigem() {
15          return origem;
16      }
17
18      public void setOrigem(String origem) {
19          this.origem = origem;
20      }
21
22  }
```

C.5.3 MsgClienteConectado.java

```
1   package csframe.mensagens;
2
3   public class MsgClienteConectado extends MsgAbstrata {
4
5       private String nome;
6       private String ipCliente;
7
8       public MsgClienteConectado() {
9
10      }
11
12      public MsgClienteConectado(String nome, String ipCliente) {
13          this.tipo = TipoMensagem.MSG_CLIENTE_CONECTADO;
```

```
14          this.nome = nome;
15          this.ipCliente = ipCliente;
16      }
17      public String getIpCliente() {
18          return ipCliente;
19      }
20      public void setIpCliente(String ipCliente) {
21          this.ipCliente = ipCliente;
22      }
23      public String getNome() {
24          return nome;
25      }
26      public void setNome(String nome) {
27          this.nome = nome;
28      }
29
30  }
```

C.5.4 MsgCriarSala.java

```
1   package csframe.mensagens;
2
3   public class MsgCriarSala extends MsgAbstrata {
4
5       private String sala;
6       private String descricao;
7
8       public MsgCriarSala(String sala, String descricao) {
9           this.tipo = TipoMensagem.MSG_CRIAR_SALA;
10          this.sala = sala;
11          this.descricao = descricao;
12      }
13
14      public String getSala() {
15          return sala;
16      }
17
18      public String getDescricao() {
19          return descricao;
20      }
21
22      public void setDescricao(String descricao) {
23          this.descricao = descricao;
24      }
25  }
```

C.5.5 MsgEntrarSala.java

```
1   package csframe.mensagens;
2
3   public class MsgEntrarSala extends MsgAbstrata {
```

```
4     private String nomeParticipante;
5     private String sala;
6
7     public MsgEntrarSala(String sala, String nomeParticipante) {
8        this.tipo = TipoMensagem.MSG_ENTRAR_SALA;
9        this.nomeParticipante = nomeParticipante;
10       this.sala = sala;
11    }
12
13    public String getNomeParticipante() {
14       return nomeParticipante;
15    }
16
17    public void setNomeParticipante(String nomeParticipante) {
18       this.nomeParticipante = nomeParticipante;
19    }
20
21    public String getSala() {
22       return sala;
23    }
24
25    public void setSala(String sala) {
26       this.sala = sala;
27    }
28
29 }
```

C.5.6 MsgEnviarAmbiente.java

```
1  package csframe.mensagens;
2
3  import java.util.Hashtable;
4  import java.util.Vector;
5
6  import csframe.servicos.Participante;
7  import csframe.servicos.Sala;
8
9  public class MsgEnviarAmbiente extends MsgAbstrata {
10
11    private Hashtable<String, Sala> salas;
12    private Hashtable<String, Participante> participantes;
13    private Hashtable<String, Vector<String>> partSala;
14
15    public MsgEnviarAmbiente(Hashtable<String, Sala> salas,
16          Hashtable<String, Participante> participantes,
17          Hashtable<String, Vector<String>> partSala) {
18       this.tipo = TipoMensagem.MSG_ENVIAR_AMBIENTE;
19       this.salas = salas;
20       this.participantes = participantes;
21       this.partSala = partSala;
22    }
23
```

```
24      public Hashtable<String, Participante> getParticipantes() {
25          return participantes;
26      }
27      public void setParticipantes(Hashtable<String, Participante> participantes) {
28          this.participantes = participantes;
29      }
30      public Hashtable<String, Sala> getSalas() {
31          return salas;
32      }
33      public void setSalas(Hashtable<String, Sala> salas) {
34          this.salas = salas;
35      }
36
37      public Hashtable<String, Vector<String>> getPartSala() {
38          return partSala;
39      }
40
41      public void setPartSala(Hashtable<String, Vector<String>> partSala) {
42          this.partSala = partSala;
43      }
44
45  }
```

C.5.7 MsgIdCliente.java

```
1   package csframe.mensagens;
2
3   public class MsgIdCliente extends MsgAbstrata {
4
5       private String idCliente;
6
7       public MsgIdCliente(String idCliente) {
8           this.tipo = TipoMensagem.MSG_ID_CLIENTE;
9           this.idCliente = idCliente;
10      }
11
12      public String getIdCliente() {
13          return idCliente;
14      }
15
16      public void setIdCliente(String idCliente) {
17          this.idCliente = idCliente;
18      }
19
20
21  }
```

C.5.8 MsgNovoParticipante.java

```
1   package csframe.mensagens;
2
```

```
3  import java.util.Hashtable;
4  import java.util.Vector;
5
6  public class MsgNovoParticipante extends MsgAbstrata {
7
8      private String nome;
9      private int idCliente;
10     private Vector<String> salas;
11     private Hashtable<String, Vector<String>> participantes;
12
13     public MsgNovoParticipante(String nome, int idCliente) {
14         this.tipo = TipoMensagem.MSG_NOVO_PARTICIPANTE;
15         this.nome = nome;
16         this.idCliente = idCliente;
17     }
18
19     public String getNome() {
20         return nome;
21     }
22
23     public void setNome(String nome) {
24         this.nome = nome;
25     }
26
27     public int getIdCliente() {
28         return idCliente;
29     }
30
31     public void setIdCliente(int idCliente) {
32         this.idCliente = idCliente;
33     }
34 }
```

Apêndice D

Código do Protótipo

Este apêndice se destina a apresentar os principais elementos do protótipo, bem como uma breve descrição de cada um deles.

D.1 Interface de Inicialização do ADContexto

A Interface de inicialização (figura D.1), como descrito no capítulo 4, tem como objetivos:

i. Escolher a Interface Gráfica para acessar os serviços da Aplicação

ii. Escolher a Infra-estrutura de comunicação

iii. Iniciar o ambiente

D.1.1 Arquivo `ConfiguracaoAVContexto.xml`

```
1  <configuracaoAmbiente>
2     <nome>Ambiente do Protótipo</nome>
3
4     <interfacesAplicacao>
5        <interfaceAplicacao>
6           <descricao>Interface Visual 1</descricao>
7           <nomeClasse>avcontexto.aplicacao.AplicacaoInterfaceComum</nomeClasse>
8        </interfaceAplicacao>
9
10       <interfaceAplicacao>
11          <descricao>Interface Visual 2 - Protótipo</descricao>
12          <nomeClasse>avcontexto.aplicacao.avancada.AplicacaoInterfaceAvancada</nomeClasse>
13       </interfaceAplicacao>
14
```

Figura D.1: Interface de Inicialização do protótipo

```
15    </interfacesAplicacao>
16
17    <intraestruturas>
18       <infraestrutura>
19          <descricao>Infraestrutura CSFrame (C/S)</descricao>
20          <nomeClasse>avcontexto.comunicacao.InfraestruturaCSFrame</nomeClasse>
21          <tipo>ClienteServidor</tipo>
22       </infraestrutura>
23
24       <infraestrutura>
25          <descricao>Infraestrutura JXTAula (P2P)</descricao>
26          <nomeClasse>avcontexto.comunicacao.InfraestruturaPeerToPeer</nomeClasse>
27          <tipo>P2P</tipo>
28       </infraestrutura>
29
30    </intraestruturas>
31
32 </configuracaoAmbiente>
```

D.2 Interface Gráfica de Serviços

A figura D.2 mostra a interface de serviços implementada para o protótipo AVContexto. Nesta interface, é possível criar, remover, entrar e sair de Grupos, além de acessar os serviços dinâmicos do ambiente, como *Chat*.

D.2.1 Código do Aspecto da Interface Gráfica de Serviços

Como descrito no capítulo 4, para cada interface visual de serviços, um aspecto é criado. A listagem D.2.1 mostra o código do arquivo `AspectoInterfaceBasica.asp`

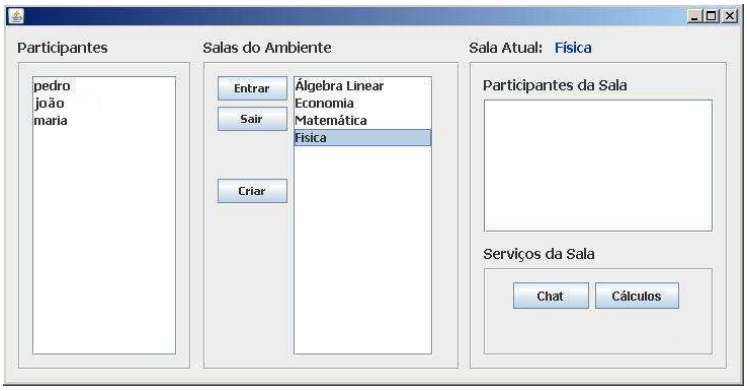


Figura D.2: Interface Gráfica de Serviços do protótipo

criado para fazer a comunicação entre a interface do protótipo com o núcleo da arquitetura.

```
1  package arppav.aplicacao;
2
3  import javax.swing.JList;
4  import javax.swing.JOptionPane;
5  import arppav.controle.eventos.ServicoEvent;
6  import arppav.controle.nucleo.Arppav;
7
8  public aspect AspectoInterfaceComum{
9
10    // ATRIBUTOS
11    private Arppav ambiente = new Arppav();
12    private String salaAtual;
13    private JList partAmbiente;
14    private JList salasAmbiente;
15    private JList partSala;
16    private String coordenadorSala;
17
18    /**
19     * INICIAR AMBIENTE
20     */
21    pointcut iniciarAmbiente(AplicacaoInterfaceComum aplicacao) : \\
22         execution(private void btnIniciarAmbienteActionPerformed(..))
23               && target(aplicacao);
24
25    @SuppressWarnings("deprecation")
26    after(final AplicacaoInterfaceComum aplicacao) : iniciarAmbiente(aplicacao) {
27         aplicacao.getMensagem().setText("Iniciando ambiente...");
28         System.out.println("Participantes Ambiente: "+
29          aplicacao.getParticipantesAmbiente().toString());
30         try {
31              ambiente.setCaminhoXML(aplicacao.getTxtCaminhoXML().getText());
32              ambiente.iniciarAmbiente(aplicacao.getTfNome().getText(),
33           aplicacao.getTfSenha().getText(), aplicacao);
34
35              salaAtual = "Ambiente";
```

```
36              coordenadorSala = "";
37
38              partSala = aplicacao.getParticipantesSala();
39
40              partAmbiente = aplicacao.getParticipantesAmbiente();
41              partAmbiente.setListData(ambiente.getInfra().getParticipantesAmbiente());
42
43              salasAmbiente = aplicacao.getSalasAmbiente();
44              salasAmbiente.setListData(ambiente.getInfra().getSalasAmbiente());
45
46              aplicacao.getMensagem().setText("Você está conectado no ambiente.");
47
48          } catch (Exception e) {
49              e.printStackTrace();
50          }
51      }
52
53      /**
54       * ATUALIZAR PARTICIPANTES DO AMBIENTE
55       */
56      pointcut atualizarParticipantes(AplicacaoInterfaceComum aplicacao) :
57        execution (private void btnAtualizarParticipantesAmbienteActionPerformed(..))
58           && target(aplicacao);
59
60      after(AplicacaoInterfaceComum aplicacao) : atualizarParticipantes(aplicacao) {
61          try {
62              aplicacao.getMensagem().setText("Atualizando participantes...");
63              aplicacao.getParticipantesAmbiente().setListData(
64           ambiente.getInfra().getParticipantesAmbiente());
65              aplicacao.getMensagem().setText("Atualização feita com sucesso!");
66              aplicacao.getMensagem().setText("");
67          } catch (Exception e) {
68              e.printStackTrace();
69          }
70      }
71
72      /**
73       * ATUALIZAR PARTICIPANTES DA SALA
74       */
75      pointcut atualizarParticipantesSala(AplicacaoInterfaceComum aplicacao) :
76         execution (private void btnAtualizarParticipantesSalaActionPerformed(..))
77              && target(aplicacao);
78      after(AplicacaoInterfaceComum aplicacao):atualizarParticipantesSala(aplicacao){
79          try {
80              aplicacao.getParticipantesSala().setListData(
81           ambiente.getInfra().getParticipantesSala(salaAtual));
82          } catch (Exception e) {
83              e.printStackTrace();
84          }
85      }
86
87      /**
88       * ENTRAR NA SALA
```

```
89      */
90      pointcut entrarSala(AplicacaoInterfaceComum aplicacao) :
91          execution (private void btnEntrarSalaActionPerformed(..))
92       && target(aplicacao);
93
94      after(AplicacaoInterfaceComum aplicacao) : entrarSala(aplicacao) {
95          try {
96              String sala = (String)salasAmbiente.getSelectedValue();
97              aplicacao.getMensagem().setText("Entrando na sala "+sala);
98              ambiente.getInfra().entrarSala(sala);
99              partSala.setListData(ambiente.getInfra().getParticipantesSala(sala));
100             salaAtual = ambiente.getInfra().getSalaAtual();
101             aplicacao.getMensagem().setText("Você acabou de entrar na sala "+sala);
102             aplicacao.getMensagem().setText("");
103         } catch (Exception e) {
104             e.printStackTrace();
105         }
106     }
107
108     /**
109      * REMOVER SALA
110      */
111     pointcut removerSala(AplicacaoInterfaceComum aplicacao) :
112         execution (private void btnRemoverSalaActionPerformed(..))
113      && target(aplicacao);
114
115     after(AplicacaoInterfaceComum aplicacao) : removerSala(aplicacao) {
116         try {
117             String sala = (String)salasAmbiente.getSelectedValue();
118             ambiente.getInfra().removerSala(sala);
119             salaAtual = ambiente.getInfra().getSalaAtual();
120         } catch (Exception e) {
121             e.printStackTrace();
122         }
123     }
124
125     /**
126      * SAIR SALA
127      */
128     pointcut sairSala(AplicacaoInterfaceComum aplicacao) :
129         execution (private void btnSairActionPerformed(..)) && target(aplicacao);
130
131     after(AplicacaoInterfaceComum aplicacao) : sairSala(aplicacao) {
132         try {
133             String sala = (String)salasAmbiente.getSelectedValue();
134             ambiente.getInfra().sairSala(sala);
135             salaAtual = ambiente.getInfra().getSalaAtual();
136         } catch (Exception e) {
137             e.printStackTrace();
138         }
139     }
140
141     /**
```

```
142      * ENVIAR MENSAGEM
143      */
144     pointcut enviarTexto(AplicacaoInterfaceComum aplicacao) :
145         execution (private void btnEnviarMensagemTextoActionPerformed(..))
146             && target(aplicacao);
147 
148     after(AplicacaoInterfaceComum aplicacao) : enviarTexto(aplicacao) {
149         try {
150             ambiente.getInfra().enviarMensagemParticipante(
151          (String)partSala.getSelectedValue(),
152                     aplicacao.getMensagemTexto().getText());
153         } catch (Exception e) {
154             e.printStackTrace();
155         }
156     }
157 
158     /**
159      * RECEBEU MENSAGEM
160      */
161     pointcut mensagemRecebida(AplicacaoInterfaceComum aplicacao,
162       ServicoEvent servent) :
163          execution (public void recebeuMensagem(..)) && (target(aplicacao)
164       && args(servent));
165 
166     after(AplicacaoInterfaceComum aplicacao, ServicoEvent servent):
167         mensagemRecebida(aplicacao, servent) {
168 
169         try {
170             JOptionPane.showMessageDialog(null, "Você recebeu a mensagem: "+
171          servent.getMensagem());
172         } catch (Exception e) {
173             e.printStackTrace();
174         }
175     }
176 
177     /**
178      * Atualizar interface
179      */
180     public pointcut atualizarInterface(AplicacaoInterfaceComum aplicacao) :
181         (execution (private void btnIniciarAmbienteActionPerformed(..)) ||
182          execution (private void btnSalasEscolaActionPerformed(..))     ||
183          execution (private void btnEntrarSalaActionPerformed(..)))
184                 && target(aplicacao);
185 
186     after (AplicacaoInterfaceComum aplicacao) : atualizarInterface(aplicacao) {
187         aplicacao.getSalaAtual().setText(salaAtual);
188         aplicacao.getNomeCoordenador().setText(coordenadorSala);
189         aplicacao.setParticipantesAmbiente(partAmbiente);
190         aplicacao.setSalasAmbiente(salasAmbiente);
191     }
192 }
```

D.3 Classe PersistenciaObjeto.java

D.4 Infra-estrutura de comunicação

D.4.1 Classe InfraestruturaPeerToPeer.java

```
1  package avcontexto.comunicacao;
2
3  import java.util.EventListener;
4  import java.util.Vector;
5  import jxtaula.GestaoAmbiente.JXTAula;
6  import jxtaula.GestaoComunicacao.ComunicacaoEvent;
7  import jxtaula.GestaoComunicacao.ComunicacaoListener;
8  import avcontexto.aplicacao.IAdaptadorInterface;
9  import avcontexto.controle.eventos.AVContextoListener;
10
11 public class InfraestruturaPeerToPeer extends JXTAula implements
12       IAdaptadorInfraestrutura, ComunicacaoListener {
13
14    private AVContextoListener al;
15    private boolean comunicacao;
16    private IAdaptadorInterface gui;
17
18    public InfraestruturaPeerToPeer() {super();}
19
20    public void iniciarInfra(String nomeParticipante, String senhaParticipante,
21             EventListener evento, IAdaptadorInterface adaptadorInter)
22                throws Exception {
23       comunicacao=false;
24       al = (AVContextoListener)evento;
25       this.gui = adaptadorInter;
26       super.inicializar(nomeParticipante, senhaParticipante, this);
27       gui.atualizarGrupos(super.getSalasEscola());
28    }
29
30    public void iniciarInfra(String nomeParticipante, String senhaParticipante,
31          String servidor, int porta, IAdaptadorInterface adaptadorInter)
32             throws Exception {
33    }
34
35    public void criarSala(String nomeSala, String descSala) throws Exception {
36       super.criarSala(nomeSala, descSala);
37    }
38
39    public void entrarSala(String nomeSala) throws Exception {
40       super.entrarSala(nomeSala);
41       gui.atualizarParticipantesGrupo(super.getParticipantesSala(nomeSala));
42    }
43
44    public void enviarMensagemParticipante(String nomeParticipante, Object mensagem)
45          throws Exception {
```

```
46          super.iniciarComunicacaoParticipante(nomeParticipante);
47          while (comunicacao==false) {}
48          super.enviarMensagemParticipante(nomeParticipante, mensagem);
49          super.finalizarComunicacaoParticipante(nomeParticipante);
50      }
51
52      public void enviarMensagemSala(String nomeSala, Object mensagem)
53              throws Exception {
54          super.enviarMensagemSala(nomeSala, mensagem);
55      }
56
57      public Vector getParticipantesAmbiente() throws Exception {
58          return super.getParticipantesEscola();
59      }
60
61      public Vector getParticipantesSala(String nomeSala) throws Exception {
62          return super.getParticipantesSala(nomeSala);
63      }
64
65      public String getSalaAtual() throws Exception {
66          return super.getSalaAtual();
67      }
68
69      public Vector<String> getSalasAmbiente() throws Exception {
70          return super.getSalasEscola();
71      }
72
73      public Vector getSalasParticipante() throws Exception {
74          return null;
75      }
76
77      public void removerSala(String nomeSala) throws Exception {
78          super.removerSala(nomeSala);
79      }
80
81      public void sairSala(String nomeSala) throws Exception {
82          super.sairSala(nomeSala);
83          gui.atualizarParticipantesGrupo(new Vector());
84      }
85
86      public void comunicacaoFinalizada(ComunicacaoEvent e) {}
87      public void comunicacaoInicializada(ComunicacaoEvent e) {}
88      public void mensagemRecebida(ComunicacaoEvent e) {}
89  }
```

D.4.2 Classe `InfraestruturaCSFrame.java`

```
1   package avcontexto.comunicacao;
2
3   import java.util.EventListener;
4   import java.util.Vector;
5   import avcontexto.aplicacao.IAdaptadorInterface;
```

```
6  import csframe.CSFrameClient;
7  import csframe.ICSFrame;
8  import csframe.mensagens.MsgCriarSala;
9  import csframe.mensagens.MsgEntrarSala;
10 import csframe.mensagens.MsgSairSala;
11
12 public class InfraestruturaCSFrame implements IAdaptadorInfraestrutura, ICSFrame{
13
14    CSFrameClient cliente;
15    IAdaptadorInterface gui;
16
17    // Método para iniciar Infraestrutura P2P - Não utilizado aqui
18    public void iniciarInfra(String nomeParticipante, String senhaParticipante,
19       EventListener evento, IAdaptadorInterface adaptadorInter) throws Exception {
20    }
21
22    // Método para iniciar Infraestrutura Cliente/Servidor
23    public void iniciarInfra(String nomeParticipante, String senhaParticipante,
24          String servidor,
25          int porta, IAdaptadorInterface adaptadorInter) throws Exception {
26       cliente = new CSFrameClient(servidor, porta, this, nomeParticipante);
27       cliente.start();
28       if (adaptadorInter != null) {
29          gui = adaptadorInter;
30          gui.atualizarParticipantesAmbiente(getParticipantesAmbiente());
31          gui.atualizarGrupos(getSalasAmbiente());
32          gui.atualizaNome(cliente.getNome());
33       }
34    }
35
36    public void criarSala(String nomeSala, String descSala) throws Exception {
37       MsgCriarSala msg = new MsgCriarSala(nomeSala, descSala);
38       msg.setOrigem(cliente.getNome());
39       cliente.sendMessage(msg);
40    }
41
42    public void entrarSala(String nomeSala) throws Exception {
43       MsgEntrarSala msg = new MsgEntrarSala(nomeSala, cliente.getNome());
44       if (!cliente.getSalaAtual().equals("")) {
45          MsgSairSala msgSair = new MsgSairSala(nomeSala, cliente.getNome());
46          cliente.sendMessage(msgSair);
47       }
48       cliente.setSalaAtual(nomeSala);
49       cliente.sendMessage(msg);
50    }
51
52    public void enviarMensagemParticipante(String nomeParticipante,
53          Object mensagem) throws Exception {
54    }
55
56    public void enviarMensagemSala(String nomeSala, Object mensagem)
57          throws Exception {
58    }
```

```
59
60      public Vector getParticipantesAmbiente() throws Exception {
61          return cliente.getParticipantes();
62      }
63
64      public Vector getParticipantesSala(String nomeSala) throws Exception {
65          return cliente.getParticipantesSala(nomeSala);
66      }
67
68      public String getSalaAtual() throws Exception {
69          return cliente.getSalaAtual();
70      }
71
72      public void removerSala(String nomeSala) throws Exception {}
73
74      public void sairSala(String nomeSala) throws Exception {
75          MsgSairSala msg = new MsgSairSala(nomeSala, cliente.getNome());
76          cliente.setSalaAtual("");
77          atualizarParticipantesGrupo(new Vector());
78          cliente.sendMessage(msg);
79      }
80
81      public Vector<String> getSalasAmbiente() throws Exception {
82          return cliente.getSalas();
83      }
84
85      public IAdaptadorInterface getGui() {
86          return gui;
87      }
88
89      public void atualizarSalas(Vector<String> salas) {
90          gui.atualizarGrupos(salas);
91      }
92
93      public void atualizarParticipantesAmbiente(Vector<String> part) {
94          gui.atualizarParticipantesAmbiente(part);
95      }
96
97      public void atualizarParticipantesGrupo(Vector<String> part) {
98          if (gui != null) {
99              gui.atualizarParticipantesGrupo(part);
100         }
101     }
102 }
```

Referências Bibliográficas

AMORIM, J. A.; ARMENTANO, V. A.; MISKULIN, M. S.; MISKULIN, R. G. S. Uso do teleduc como um recurso complementar no ensino presencial. *As Novas Mídias e Ferramentas*, 2005. Último acesso em 06 de Agosto de 2008.

ANDERSON, D. P.; COBB, J.; KORPELA, E.; LEBOFSKY, M.; WERTHIMER, D. Seti@home: An experiment in public-resource computing. *Communications of the ACM. Space Sciences Laboratory. U.C. Berkeley*, 2002. Vol. 45 No. 11, November 2002, pp. 56-61.

BREITKREUZ, H. Emule. *Wikipedia*, 2002. Último acesso em 05 de Agosto de 2008.

CAMPOS, A. de; TEIXEIRA, A. C. criativo: um ambiente hipermídia de autoria colaborativa. *RENOTE - Revista Novas Tecnologias na Educação. Universidade Feredal do Rio Grande do Sul*, v. 4, n. 2, p. 10, Dezembro 2006.

CGLIB. Code generation library. *http://cglib.sourceforge.net/*, 2008. Último acesso em 12/08/2008.

CHIBA, S. Load-time structural reflection in java. *14th European Conference on Object-Oriented Programming*, p. p. 313–336, Junho 2000. Cannes, França.

COLE, J.; FOSTER, H. *Using Moodle: Teaching with the Popular Open Source Course Management System*. 2. ed. [S.l.]: Orelly, 2007. ISBN 059652918X.

COUTAZ, J.; CROWLEY, J. L.; DOBSON, S.; GARLAN, D. Context is key. *Communications of the ACM*, 2005.

DAMASCENO, K.; CACHO, N.; GARCIA, A.; LUCENA, C. Tratamento de exceções sensível ao contexto. *XX Simpósio Brasileiro de Engenharia de Software*, 2006.

DEY, A. K.; ABOWD, G. D. Towards a better understanding of context and context-awareness. *Conf on Human Factors in Computing Systems*, p. 12, Abril 1999.

FABRE, J.; NICOMETTE, V. Implementing fault tolerant applications using reflective object-oriented programming. *Proceedings of th 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena, CA, EUA*, Junho 1995.

FANNING, S. Napster. *Wikipedia*, 1999. Último acesso em 05 de Agosto de 2008.

FERBER, J. Computational reflection in class based object orientedlanguages. *Proceedings of the Conference on Object-Oriented Programming:Systems, Languages and Applications*, p. p. 317–326, Outubro 1989. New Orleans, EUA.

FUKS, H.; GEROSA, M. A.; RAPOSO, A. B.; LUCENA, C. J. P. de. O modelo de colaboração 3c no ambiente aulanet. *Informática na Educação: teoria e prática*, Junho 2005. Porto Alegre-RS.

GEROSA, M. A.; RAPOSO, A. B.; FUKS, H.; LUCENA, C. J. P. de. Uma arquitetura para o desenvolvimento de ferramentas colaborativas para o ambiente de aprendizagem aulanet. *Anais do Simpósio Brasileiro de Informática na Educação (SBIE)*, Novembro 2004.

GOMES, R. L.; RIVERA-HOYOS, G. J.; COURTIAT, J. Integrating collaborative applications with leica. *3rd IEEE International Conference on Information*, Junho 2005. Taiwan.

GRADECKI; LESIECKI. *Mastering AspectJ - Aspect Oriented Programming in Java*. 1. ed. [S.l.]: Indianapolis. Wiley Publishing, 2003.

GRADECKI, J. D. *Mastering JXTA: Building Java Peer-to-Peer Applications*. [S.l.]: John Wiley & Sons, 2002. ISBN 0471250848.

JAXB. Java architecture for xml binding (jaxb). *Sun Microsystems*, 2008. Http://java.sun.com/javase/6/docs/technotes/guides/xml/jaxb/index.html.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J.-M.; IRWIN, J. Aspect-oriented programming. *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

LIMA, C. V. *CONTROLADOR DE CONFERÊNCIAS PARA SISTEMAS COLABORATIVOS*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DE SANTA CATARINA, Novembro 2007.

LUCENA, M. Aulanet: Uma ferramenta de interação e formação na diversidade. *Anais do XXVIII Congresso da SBC / WIE - Workshop sobre Informática na Escola*, Julho 2008.

MAES, P. Concepts and experiments in computational reflection. *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages and Applications*, p. p. 147–155, Dezembro 1987. Orlando, EUA.

MUHAMMAD, H.; FERREIRA, A. P. L. Exploração de reflexão computacional através de um modelo de objetos sem classes. *Revista Eletrônica de Iniciação Científica - REIC*, 2003.

ORACLE. Banco de dados oracle 11g. *Site Oracle Brasil*, 2008. Http://www.oracle.com/global/br/database/index.html.

PIVETA, E. K. Um modelo de suporte a programação orientada a aspectos. *Dissertação de Mestrado. Universidade Federal de Santa Catarina*, 2001.

POSTGRESQL. Postgresql database. *PostgreSQL Web Site*, 2008. Http://www.postgresql.org/docs/.

RIVERA, G. de J. H.; GOMES, R. L.; WILLRICH, R.; COURTIAT, J. P. Colab: Co-navigation sur le web. *NOuvelles TEchnologies de la REpartition (NOTERE)*, Junho 2006. Toulose, França.

RIZZETTI, T. A.; TROIS, C.; LIMA, J. C. D.; AUGUSTIN, I. Ambiente colaborativo p2p1 para o projeto pdsce. *V Simpósio de Informática da Região Centro/RS*, Junho 2006.

ROCHA, H. V. da. *Educação a distância: Fundamentos e práticas*. [S.l.]: NIED, 2002. Cap. 11, p. 197-212. Em português.

SAMPAIO, A. P.; SERRA, P. R. F.; ANSELMO, F. J. M.; SOARES, J. M.; SERRA, A. de B.; BARROSO, G. C.; SOUZA, J. N. Jxtaula: Uma plataforma baseada em p2p para sistemas de ensino a distância. *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Maio 2008.

SCORM. Scorm 2004 3rd edition. *Advanced Distributed Learning*, 2008. Último acesso em 22 de Novembro de 2008.

SOARES, J. M. *Contribution à la communication gestuelle dans les environnements virtuels collaboratifs*. Tese (Doutorado) — Instituto Nacional de Telecomunicações da França, Setembro 2004. Tese apresentada para a obtenção do grau de Doutor no Instituto Nacional de Telecomunicação.

SOUSA, F. C. de; GROTT, M. C. Utilização da reflexão computacional para implementação de um monitor de software orientado a objetos em java. *VI Encontro de Estudantes de Informática do Estado do Tocantins - ENCOINFO*, Novembro 2004. Palmas, Tocantins.

SUN. Microsystems java standard edition platform documentation. *Site oficial da SUN*, 1998.

VALENTE, J. A.; PRADO, M. E. B. B.; ALMEIDA, M. E. B. D. *EDUCAÇÃO A DISTÂNCIA VIA INTERNET*. 1. ed. [S.l.]: Avercamp, 2003. ISBN 8589311147.

WIKI, E. Programação reflexiva. *Wiki*, 2008. Último acesso em 09 de Agosto de 2008. http://pt.wikipedia.org/wiki/Reflexao.

WILSON, B. J. *JXTA*. 1. ed. [S.l.]: David Dwyer, 2002. Última data de acesso: 03/06/2007. ISBN 0-73571-234-4.

WINCK, D. V.; JUNIOR, V. G. *AspectJ: Programação Orientada a Aspectos com Java*. 1. ed. [S.l.]: Novatec, 2006. ISBN 85-7522-087-X.

ZAVADSKI, A. C. *Utilizando reflexão computacional no desenvolvimento de aplicações distribuídas*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Junho 2003.