Eliane Gomes Guimarães

Um modelo de componentes para aplicações telemáticas e ubíquas

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Mauricio Ferreira Magalhães

Campinas, SP
2004

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

Guimarães, Eliane Gomes
G947m Um modelo de componentes para aplicações telemáticas
e ubíquas / Eliane Gomes Guimarães - Campinas, SP: [s.n.], 2004.

Orientador: Mauricio Ferreira Magalhães.
Tese (doutorado) - Universidade Estadual de Campinas,
Faculdade de Engenharia Elétrica e de Computação.

1. Agentes móveis (Software). 2. Componentes de
software. 3. Computação ubíqua. 4. Robôs móveis
5. Software - desenvolvimento.
I. Magalhães, Mauricio Ferreira. II. Universidade Estadual de Campinas.
Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês:	A component model for telematic and ubiquitous applications
Palavras-chave em Inglês:	Mobile agents, Component software, Ubiquitous computing, Mobile robots e Software development
Área de concentração:	Engenharia de Computação
Titulação:	Doutora em Engenharia Elétrica
Banca Examinadora:	Graça Bressan, Ivan Luiz Marques Ricarte, Marcel Bergerman e Mario Jino
Data da defesa:	21/12/2004

Eliane Gomes Guimarães

Um modelo de componentes para aplicações telemáticas e ubíquas

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 21/12/2004

Banca Examinadora:
Profa. Dra. Graça Bressan - USP
Prof. Dr. Ivan Luiz Marques Ricarte - UNICAMP
Dr. Marcel Bergerman - Instituto Genius
Prof. Dr. Mario Jino - UNICAMP
Prof. Dr. Mauricio Ferreira Magalhães - UNICAMP

Campinas, SP
2004

Resumo

Esta tese descreve CM-tel, um modelo de componentes para aplicações telemáticas e ubíquas. CM-tel é neutro em termos de tecnologia, sendo especificado por meio da linguagem UML (*Unified Modeling Language*). Componentes CM-tel são capazes de executar em plataformas destinadas tanto a computadores tradicionais quanto a dispositivos com limitado poder computacional tais como dispositivos móveis. CM-tel define os três tipos de interfaces prescritas pelo Modelo de Referência para Processamento Distribuído Aberto (RM-ODP), as interfaces operacional, de sinal e de fluxo contínuo. Interfaces de fluxo contínuo são fundamentais para o desenvolvimento de aplicações telemáticas. A arquitetura do contêiner CM-tel integra componentes e agentes móveis em um único ambiente computacional. Esta integração permite que aplicações implementem suas funcionalidades combinando componentes e agentes móveis. Esta tese propõe ainda uma arquitetura para plataformas de software que suportam o modelo CM-tel. A arquitetura utiliza XSLT (*XML Stylesheet Language Transformation*) para transformação de modelos e geração de código. Uma plataforma baseada na tecnologia CORBA (*Common Object Request Broker Architecture*) e uma aplicação na área de laboratórios virtuais foram implementadas com a finalidade de avaliar o modelo CM-tel.

Palavras-chave: Componentes de Software, Agentes Móveis, Plataformas de Software, Aplicações Telemáticas e Ubíquas, Laboratórios Virtuais.

Abstract

This thesis describes CM-tel, a component model for telematic and ubiquitous applications. CM-tel is neutral in terms technology and is specified through the Unified Modeling Language (UML). CM-tel components can execute on platforms targeted to both conventional computers and devices with limited computing power such as mobile devices. CM-tel defines the three types of interfaces prescribed by the Reference Model for Open Distributed Processing (RM-ODP), the operational, signal, and stream interfaces. Stream interfaces are central to the development of telematic applications. The CM-tel container architecture integrates both components and mobile agents into a single computing environment. This integration allows applications to implement their functionalities by combining components and mobile agents. This thesis also proposes an architecture for software platforms supportting the CM-tel component model. The architecture relies on XSLT (XML Stylesheet Language Transformation) for model transformation and code generation. A platform based on the CORBA (Common Object Request Broker Architecture) technology and an application in the field of virtual laboratories were implemented in order to assess the CM-tel component model.

Keywords: Software Components, Mobile Agents, Software Platforms, Telematic and Ubiquitous Applications, Virtual Laboratories.

Agradecimentos

A Deus.

À Mariza e Geraldo, mãe e saudoso pai, um profundo sentimento de gratidão, admiração e respeito.

Ao meu marido Eleri, pelo amor, apoio, amizade e carinho dedicados, e, também pelo exemplo de competência sempre presente na busca do conhecimento.

Às minhas irmãs Vivianne, Maria Helena e Cristiane e sobrinhos Ronny e Rafael, pela compreensão face à minha constante ausência.

Ao meu orientador, Prof. Mauricio F. Magalhães, a minha sincera gratidão pela orientação, encorajamento e oportunidade de desenvolver este trabalho.

Ao Dr. Marcel Bergerman, coordenador do projeto REAL no período 1997-2001 no CenPRA, pela oportunidade, confiança e incentivo para a realização deste trabalho, bem como por conceder recursos financeiros obtidos junto à Fapesp.

Ao Centro de Pesquisas Renato Archer (CenPRA) pelo estímulo ao aperfeiçoamento de seu corpo técnico.

Ao Dr. Samuel S. Bueno, chefe da DRVC do CenPRA, e demais colegas do CenPRA, pelo tempo disponibilizado para o desenvolvimento deste trabalho.

Aos alunos de mestrado Miglinski, Rossano, Tadeu e Wander pelas valiosas contribuições em implementações diretamente vinculadas a este trabalho.

Aos alunos de iniciação científica Bruno, James, Mateus, Nicholas, Sassi e Victor, pela dedicação ao projeto REAL.

À FEEC/Unicamp pela oportunidade de desenvolver esta tese.

Ao meu marido Eleri
com carinho, admiração e amor.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

AAL5 - ATM Adaptation Layer 5

API - Application Programming Interface

ATM - Asynchronous Transfer Mode

CASE - Computer-Aided Software Engineering

CCM - CORBA Component Model

CCM-tel - CORBA CM-tel

CIDL - Component Implementation Definition Language

CIF - Component Implementation Framework

CM-tel - Component Model for Telematic Applications

COM - Component Object Model

CORBA - Common Object Request Broker Architecture

DCE - Distributed Computing Environment

DCOM - Distributed Component Object Model

DiffServ - Differentiated Services

DLL - Dynamic Linking Library

DTD - Document Type Definition

EDOC - Enterprise Distributed Object Computing

EJB - Enterprise Java Beans

ESBC - Engenharia de Software Baseada em Componentes

HTML - Hypertext Markup Language

HTTP - Hipertext Transport Protocol

IDL - Interface Definition Language

IIOP - Internet Inter-ORB Protocol

ISO - International Organization for Standardization

ITU-T - International Telecommunication Union - Telecommunication Standardization Sector

J2EE - Java 2 Enterprise Edition

J2ME - Java 2 Micro Edition

J2SE - Java 2 Standard Edition

JAR - Java Archive

JMF - Java Media Framework

JMRP - Java Remote Method Protocol

JNDI - Java Naming and Directory Interface

JSP - Java Server Pages

JTA - Java Transaction API

LAN - Local Area Network

MAF - Mobile Agent Facility

MDA - Model Driven Architecture

MTS - Microsoft Transaction Server

OCL - Object Constraint Language

OMG - Object Management Group

ORB - Object Request Broker

OSD - Open Software Description

PIM - Platform Independent Model

POA - Portable Object Adapter

PSM - Platform Specific Model

QoS - Quality of Service

REAL - Remotely Accessible Laboratory

RFP - Request for Proposal

RM-ODP - Reference Model for Open Distributed Processing

RMC - Remote Method Calls

RMI - Remote Method Invocation

RPC - Remote Procedure Call

RT-CORBA - Real Time CORBA

RTCP - Real Time Control Protocol

RTP - Real Time Protocol

SLA - Service Level Agreement

SNMP - Simple Network Management Protocol

SOAP - Simple Object Access Protocol

TCP/IP - Transfer Control Protocol/Internet Protocol

TINA-C - Telecommunication Information Network Architecture Consortium

UDDI - Universal Description, Discovery and Integration

UDP - User Datagram Protocol

UML - Unified Modeling Language

URL - Uniform Resource Locator

UUID - Universally Unique IDs

W3C - World Wide Web Consortium

WLAN - Wireless Local Area Network

WSCM-tel - Web Services CM-tel

WSDL - Web Services Description Language

WWW - World Wide Web

XMI - XML Metadata Interchange

XML - Extensible Markup Language

XSLT - XML Stylesheet Language Transformation

Trabalhos Publicados pela Autora

1. E. G. Guimarães, M. Bergerman, E. Cardozo, M. F. Magalhães, "Um Framework de Transmissão de Áudio e Vídeo para Novos Serviços de Telecomunicações", 18º Simpósio Brasileiro de Redes de Computadores (SBRC00), Belo Horizonte, MG, maio de 2000.

2. E. G. Guimarães, E. Cardozo, M. Bergerman, M. F. Magalhães, "Um Framework de Transmissão de Áudio e Vídeo para Laboratórios de Acesso Remoto", XIII Congresso Brasileiro de Automática (CBA00), Florianópolis , setembro de 2000.

3. L. F. Faina, R. P. Pinto, E. G. Guimarães, E. Cardozo, "Mobile Agents for Supporting Ubiquity in Telecommunication Services", Second Latin American Network Operation, Management Symposium (LNNOMS), A. Loureiro, J. Nogueira(eds.), Belo Horizonte, MG, agosto de 2001,

4. E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, M. Bergerman, E. Cardozo, M. F. Magalhães, "REAL: A Virtual Laboratory for Mobile Robot Experiments", First IFAC Conference on Telematics Applications in Automation, Robotics, Weingarten, Germany, julho de 2001.

5. E. G. Guimarães, E. Cardozo, M. Magalhães, M. Bergerman, A. T. Maffeis, J. L. Pereira, B. G. Russo, C. A. Miglinsk, R. P. Pinto, "Desenvolvimento de Software Orientado a Componentes para Novos Serviços de Telecomunicações", 19º Simpósio Brasileiro de Redes de Computadores (SBRC01), Florianópolis, SC, maio de 2001.

6. R. P. Pinto, L. F. Faina, A. T. Maffeis, E. G. Guimarães, C. A. Miglinsk, E. Cardozo, "Uma Arquitetura para Disponibilização e Gerência de Serviços na Internet", 20º Simpósio Brasileiro de Redes de Computadores (SBRC02), Buzios, RJ, maio de 2002.

7. R. P. Pinto, E. G. Guimarães, E. Cardozo, M. F. Magalhães, "Incorporação de Qualidade de Serviços em Aplicações Telemáticas", 21º Simpósio Brasileiro de Redes de Computadores (SBRC03), Natal, RN, maio de 2003.

8. E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, E. Cardozo, M. Bergerman M. F. Magalhães, "REAL: A Virtual Laboratory for Mobile Robot Experiments", IEEE Transactions on Education, Vol. 46, No. 1, fevereiro de 2003.

9. E. G. Guimarães, A. T. Maffeis, R. P. Pinto, C. A. Miglinski, E. Cardozo, M. Bergerman, M. F. Magalhães, "REAL: A Virtual Laboratory Built from Software Components", Proceedings of the IEEE, Vol. 91, No. 3, março de 2003.

10. E. G. Guimarães, E. Cardozo, M.F. Magalhães, W. P. Gomes, R. P. Pinto, L. F. Faina, "CCM-tel- uma Plataforma para Aplicações Telemáticas e Ubíquas", 22º Simpósio Brasileiro de Redes de Computadores (SBRC04), Gramado, RS, maio de 2004.

Capítulo 1

Introdução

Este capítulo apresenta uma visão geral de sistemas distribuídos, onde abordamos a evolução das plataformas de *middleware* tradicionais para as plataformas orientadas a componentes de *software*. O objetivo desta evolução é diminuir a complexidade no desenvolvimento de novas aplicações em ambientes heterogêneos e distribuídos. As contribuições e os principais trabalhos relacionados à nossa proposta também são discutidos.

1.1 Visão Geral de Sistemas Distribuídos

As tecnologias de informação, computação distribuída, telecomunicações e, particularmente a Internet, têm convergido e rapidamente constituído o cerne da empresa moderna, tornando-se deste modo muito importante para o seu funcionamento. O impacto proporcionado por estas tecnologias tem contribuído de forma significativa para a evolução econômica e tecnológica das sociedades. Uma empresa explorando estas tecnologias pode tirar vantagens e obter boas oportunidades em vários aspectos: compartilhamento e acesso a informações úteis, comunicação mais rápida e barata, divulgação de serviços, facilidade para adaptar-se a novos requisitos impostos por novas tecnologias, ampliação de mercado e baixos custos operacionais. O uso destas tecnologias para aumentar a qualidade dos serviços e a competitividade destas empresas tem sido reconhecido como uma estratégia a ser encorajada, bem como uma vantagem que beneficia uma corporação em relação a outras.

O espectro da utilização de tecnologias emergentes é altamente extenso e, em conjunto com modernas infra-estruturas de redes de computadores, têm possibilitado o desenvolvimento de sofisticadas aplicações que exploram as potencialidades e os benefícios de ambientes computacionais de natureza distribuída. Estas aplicações, pela disponibilização distribuída das suas funcionalidades, alcançam um considerável desempenho, melhor compartilhamento de recursos, além de uma maior disponibilidade de seus serviços e escalabilidade. Estas tecnologias inovadoras têm atraído um novo perfil de clientes

e de desenvolvedores, mais exigentes e com expectativas de que estes serviços sejam implementados com rapidez, baixo custo, segurança, confiabilidade e performance.

Entretanto, o desenvolvimento de aplicações em ambientes distribuídos é extremamente complexo. A diversidade de tópicos associada à computação distribuída somada ao constante avanço tecnológico, defronta o desenvolvedor destas aplicações com problemas inexistentes em ambientes centralizados. No desenvolvimento de tais aplicações algumas questões importantes devem ser consideradas, tais como: heterogeneidade de plataformas, sistemas operacionais e linguagens de programação; concorrência; comunicação entre os elementos distribuídos; segurança e qualidade de serviço. Via de regra, os sistemas que fornecem soluções para estas questões escondem do desenvolvedor os detalhes inerentes à distribuição do processamento, tais como a localização geográfica dos elementos funcionais, replicação de dados e funcionalidades, além de ocorrência de falhas. Esta propriedade de ocultar detalhes é denominada transparência e constitui um ponto crítico no desenvolvimento de sistemas distribuídos.

Portanto, para a construção de serviços escaláveis e adequados às expectativas do mercado, os desenvolvedores de aplicações necessitam de soluções técnicas para o suporte ao desenvolvimento e a disponibilização destes serviços; caso contrário, este desenvolvimento pode tornar-se tão complexo que inviabilize a utilização de ambientes distribuídos em situações reais.

Historicamente, os primeiros sistemas distribuídos utilizavam a troca de mensagens como elo de comunicação entre os seus componentes. Este mecanismo empregava apenas os recursos oferecidos pela interface de programação dos protocolos de transporte; por exemplo, a interface de *sockets* comumente associada à pilha de protocolos TCP/IP (*Transfer Control Protocol/Internet Protocol*).

Um marco importante no desenvolvimento de sistemas distribuídos foi a extensão da programação estruturada para ambientes distribuídos pelo modelo cliente/servidor. Neste modelo, os elementos assumem o papel de clientes (requisitantes de serviços) ou servidores (provedores de serviços). Assim sendo, a aplicação é estruturada segundo serviços (implementados em procedimentos) que servidores colocam à disposição de clientes. Os clientes invocam serviços por meio de um mecanismo de interação denominado Chamada de Procedimento Remoto (RPC: *Remote Procedure Call*). Este mecanismo permite que um elemento invoque procedimentos definidos fora do seu espaço de endereçamento, passando parâmetros e recebendo resultados, de forma similar à chamada de procedimentos locais. Esta propriedade torna a programação de sistemas distribuídos similar à programação estruturada de sistemas centralizados.

Da mesma forma que o paradigma da programação estruturada foi estendido para ambientes distribuídos, o mesmo fato ocorreu com o paradigma de programação orientada a objetos. Nesta extensão, os objetos passam a se localizar em diferentes espaços de endereçamento (objetos distribuídos) e são capazes de conduzir computações em paralelo e de forma autônoma (isto é, os objetos são ativos).

Nesta linha, a necessidade do mercado para suporte a objetos distribuídos foi suprida pelas tecnologias de *middleware*, desenvolvidas para o novo paradigma de computação de objetos distribuídos. Tipicamente, as plataformas de *middleware* se interpõem entre as aplicações e o sistema operacional do processador, fornecendo uma interface de programação uniforme para os desenvolvedores de aplicações. O *middleware* é a camada de *software* que fornece o suporte às interações entre as diferentes partes das aplicações distribuídas, utilizando uma infra-estrutura de computação e comunicação. As plataformas de *middleware* mais empregadas na atualidade são CORBA (*Common Object Request Broker Architecture*) [3] do consórcio OMG (*Object Management Group*) [4], Java/RMI (*Java Remote Method Invocation*) [5] da Sun Microsystems e DCOM (*Distributed Component Object Model*) [6], que é a extensão do COM (*Component Object Model*) da Microsoft.

Sob o ponto de vista do desenvolvedor de aplicações, estas plataformas permitem às aplicações interagirem de forma totalmente transparente, sem qualquer conhecimento da infra-estrutura de comunicação entre os diversos objetos da aplicação distribuída. Os aspectos de localização, comunicação remota e de formatos de dados são totalmente escondidos do programador. Além disso, estas plataformas proporcionam abstrações e um conjunto de serviços comuns a diversas categorias de aplicações, que facilitam o seu desenvolvimento e instalação em ambientes distribuídos e heterogêneos. Entre estes serviços encontram-se os serviços de nomes, de ciclo de vida de objetos, de segurança, de persistência, de transações, entre outros. A infra-estrutura de suporte fornecida, tanto para o lado cliente quanto para o lado servidor da interação, permite a transparência de distribuição de forma que clientes possam utilizar os serviços disponibilizados por objetos sem se preocupar com os aspectos de localização e implementação destes objetos.

Em paralelo com as soluções oriundas da indústria, a ISO (*International Organization for Standardization*) em conjunto com a ITU-T (*International Telecommunication Union - Telecommunication Standardization Sector*) iniciou um processo de padronização com vistas à definição de um conjunto de regras e padrões para guiar e uniformizar o desenvolvimento de sistemas distribuídos. O Modelo de Referência para Processamento Distribuído Aberto (RM-ODP) [7][8] foi o resultado deste trabalho. RM-ODP é considerado um metamodelo para processamento distribuído aberto, genérico e igualmente aplicável a diferentes domínios, que fornece as bases conceituais para que modelos concretos sejam desenvolvidos. TINA-C (*Telecommunication Information Network Architecture Consortium*) [9] e a especificação do padrão industrial CORBA destacam-se como exemplos de adoção, direta ou indiretamente, de diversos conceitos recomendados pelo RM-ODP. Em adição, RM-ODP também tem sido muito referenciado na avaliação das atuais plataformas de *middleware* [10], bem como na literatura especializada que apresenta o "estado da arte" em engenharia de *software* orientada a componentes [11].

Embora largamente empregadas na indústria e academia, as plataformas de *middleware* tradi-

cionais apresentam muitas deficiências no suporte às aplicações distribuídas inovadoras [12]. As principais deficiências são:

- estas plataformas, por simplificar o lado do cliente, impõem várias restrições no desenvolvimento do lado servidor; por exemplo, o desenvolvedor deve gerenciar todo o ciclo de vida dos objetos servidores;
- a utilização dos serviços comuns da plataforma é de responsabilidade do desenvolvedor. Por exemplo, uma aplicação que emprega mobilidade de código, deve possuir todas as chamadas para o serviço de agentes embutidas em seu próprio código. Isto demanda do desenvolvedor um profundo conhecimento sobre o serviço (uso, limitações, desempenho, etc.), forçando-o a se concentrar simultaneamente na lógica da aplicação e na complexidade dos serviços da plataforma de *middleware*;
- não existe uma visibilidade no que diz respeito às interconexões entre os objetos. As conexões entre os objetos são encapsuladas nos próprios objetos e não podem ser facilmente configuradas por arquitetos de *software*, o que impossibilita explicitar as interações e dependências entre os objetos, dificultando a obtenção de uma visão arquitetural precisa das aplicações;
- a facilidade de geração automática de código, oferecida por estas plataformas, é restrita ao suporte à invocação remota de métodos (*stubs* e *skeletons*);
- estas plataformas não oferecem nenhum suporte para a configuração e controle dos recursos do sistema;
- o reuso de *software* é limitado, uma vez que a granularidade dos objetos distribuídos é baixa;
- o processo de distribuição e instalação dos objetos da aplicação é ad hoc, ou seja, não existe nenhum padrão para instalar os objetos em seus respectivos ambientes de execução.

As desvantagens citadas acima restringem o uso das plataformas de *middleware* tradicionais no desenvolvimento de aplicações, o que o torna uma atividade ainda muito complexa, resultando em um baixo potencial para reutilização e portabilidade.

Com o intuito de minimizar as deficiências citadas acima e atender aos requisitos impostos pelas novas aplicações surgiu um novo paradigma de computação distribuída: a computação de componentes distribuídos. Este paradigma enfatiza as características de alto índice de reuso, evolução (substituição de elementos sem interferir nos demais elementos do sistema), especialização ("customização") de componentes e geração automática de código. No momento, projetos acadêmicos e industriais desenvolvidos utilizando recentes avanços em engenharia de *software*, notadamente em

engenharia de *software* baseada em componentes (ESBC), têm demonstrado que o paradigma de componentes preenche com vantagens os requisitos de engenharia de *software* quando comparado com o paradigma de objetos [13][14][15]. Uma outra contribuição importantíssima é o conceito de padrões de projeto (*design patterns*) [16], considerado também um elemento chave em todos os desenvolvimentos. Nesta linha, os desenvolvedores de *software* podem reusar um projeto existente. Esses recentes avanços em ESBC, em conjunto com o desenvolvimento das tecnologias citadas acima, fornecem os meios para melhorar a produtividade e a qualidade de desenvolvimento de *software*, minimizando o esforço para este desenvolvimento.

A evolução do conceito de objeto para o conceito de componente de *software* facilita o projeto, implementação e instalação da aplicação, por meio da sua decomposição em um conjunto de elementos executáveis que podem ser independentemente projetados, construídos, executados, testados, agrupados (empacotados) e mantidos. Esta evolução exerce um profundo impacto sobre as tecnologias de *middleware*. As tecnologias tradicionais de *middleware* evoluíram incorporando o suporte a componentes e mudando de um contexto orientado a objetos para um contexto orientado a componentes. O consórcio OMG publicou, em 2002, o modelo de componentes CCM (*CORBA Component Model*) [17] como uma extensão do modelo de objetos padrão da especificação CORBA; a Sun Microsystems estendeu o seu modelo de objetos distribuídos da plataforma Java com o modelo de componentes EJB (*Enterprise Java Beans*) [18]; e a Microsoft incorporou o modelo de componentes COM+ [6] na plataforma .Net, sendo uma evolução do seu modelo de objetos padrão COM/MTS/DCOM [6], onde MTS (*Microsoft Transaction Server*) é um conjunto de serviços que suporta objetos distribuídos e processamento de transação.

É importante salientar que estes modelos de componentes constituem uma camada fornecida acima das plataformas de *middleware* tradicionais, ou seja, é uma camada incorporada entre a aplicação e as plataformas tradicionais. Estas novas plataformas que implementam esta camada utilizam abordagens semelhantes, embora com características próprias, e permitem o desenvolvimento de aplicações como uma composição de componentes distribuídos. Segundo Marvie e Merle [19], neste contexto de evolução, o modelo de componentes CCM oferece um maior potencial para os processos de projeto, desenvolvimento, empacotamento, instalação e execução de componentes distribuídos. Atualmente, existem vários projetos acadêmicos em desenvolvimento [12][19][20][11] para avaliar, implementar e estender o CCM e a sua utilização. No entanto, como a versão final da especificação CCM foi disponibilizada há pouco tempo ainda não existe uma implementação comercial completa. Um projeto de pesquisa, denominado OpenCCM [21], implementa um subconjunto do CCM, sendo uma das plataformas CCM disponível no momento.

As novas tecnologias e plataformas de *middleware* citadas, aliadas à crescente difusão de serviços oferecidos na Internet, mostra que estamos presenciando um período de mudanças muito rápidas.

Este processo tem se refletido em inúmeras oportunidades e motivado a concepção de categorias de aplicações distribuídas ainda mais sofisticadas.

1.2 Motivações

Nos últimos anos observamos o aparecimento de aplicações distribuídas emergentes que combinam características de aplicações telemáticas e aplicações ubíquas. As aplicações telemáticas implementam serviços que são disponibilizados em localizações remotas e que proporcionam algum mecanismo de telepresença, onde pela comunicação multimídia distribuída é realizada a "imersão" do usuário em um ambiente geograficamente distante, ambiente este real ou virtual. As aplicações ubíquas permitem o acesso aos serviços através de múltiplos terminais, notadamente os terminais móveis (de pequeno ou grande poder computacional), além de capacidade de adaptação a tais terminais e ao ambiente de execução. Tripathi [22] enumera quatro requisitos como fundamentais para aplicações telemáticas e ubíquas:

1. suporte para a integração de componentes em oposição à programação;

2. suporte para novos paradigmas de computação distribuída, principalmente aqueles baseados em código móvel e em tecnologias de agentes móveis;

3. suporte para adaptação dinâmica das aplicações aos ambientes de execução;

4. suporte para interações baseadas em multimídia distribuída em tempo real.

As tecnologias e plataformas de *middleware* orientadas a componentes de *software* existentes atualmente são incapazes de atender aos requisitos acima, principalmente aos três últimos. Esta ausência tem forçado os desenvolvedores a adotar soluções pontuais, ou seja, soluções que atendam determinados requisitos apenas. A integração destas soluções pontuais ocorre sempre com impacto negativo na confiabilidade, desempenho e, principalmente, custo. Por exemplo, para o desenvolvimento de uma aplicação telemática uma das possíveis soluções proporcionadas pelas tecnologias disponíveis poderia ser:

- para o acesso a esta aplicação podem ser empregadas tecnologias centradas na Web, tais como JSP (*Java Server Pages*) e Java *servlet*;

- para a lógica da aplicação podem ser implementados componentes de *software*, desenvolvidos utilizando plataformas comerciais, por exemplo, EJB;

- para o suporte à comunicação multimídia pode ser utilizado um *framework* para a gerência a fluxos de mídia contínua, tal como o serviço A/VStreams (*Audio/Video Streams*) [23] do OMG;

- caso a aplicação necessite de suporte para mobilidade de código, é necessário uma plataforma de suporte à mobilidade de código, por exemplo, a plataforma JADE [24].

Esta ausência de soluções abrangentes para atender aos requisitos identificados por Tripathi motivou-nos a desenvolver a pesquisa apresentada nesta tese. Esta pesquisa tem como motivação principal contribuir para o desenvolvimento das novas aplicações distribuídas citadas acima.

A tendência na direção de desenvolvimento baseado em componentes de *software* aponta para uma solução centrada neste paradigma. Outra área de interesse é o fornecimento de suporte às propriedades não-funcionais destes sistemas. Entretanto, pode-se constatar que as deficiências das plataformas de componentes de *software* atuais têm na sua raiz o fato das mesmas estarem vinculadas a uma tecnologia específica, além de terem como aplicações alvo aquelas típicas das áreas comerciais onde propriedades não-funcionais tais como transação, segurança e persistência são importantes.

Portanto, a proposta mais natural é desenvolver um modelo de componentes desvinculado (independente) de qualquer tecnologia e capaz de atender alguns requisitos ausentes nos modelos existentes, notadamente o suporte à comunicação multimídia distribuída com facilidades de qualidade de serviço. Recentemente, a migração para as redes sem fio está impondo novos desafios com a utilização de terminais portáteis tais como, computadores de mão (*handhelds*) e telefones celulares. Neste contexto, a mobilidade tem sido uma característica cada vez mais presente nas novas aplicações, necessitando que este modelo forneça suporte a outros requisitos também importantes, tais como: código móvel e adaptação dinâmica da aplicação ao seu ambiente de execução.

A motivação para a concepção de um novo modelo de componentes de *software* independente de tecnologia é privilegiar as fases de especificação e projeto da aplicação, onde os aspectos relacionados à tecnologia são secundários. Esta estratégia vai ao encontro do reuso de projeto em adição ao reuso de código, cuja conseqüência mais importante é preservar as soluções de projeto independentemente da evolução ou substituição das tecnologias adotadas.

Para a efetiva utilização do modelo proposto no desenvolvimento de aplicações faz-se necessário a construção de plataformas capazes de suportar o modelo de componentes de *software* proposto. Tais plataformas devem ser capazes de sintetizar os componentes especificados pela aplicação (de forma independente de tecnologia) em uma dada tecnologia (ou conjunto de tecnologias). Neste contexto, a principal motivação é a busca por arquiteturas de plataformas que abstraiam para o desenvolvedor as peculiaridades da tecnologia alvo, além de permitirem a geração automática de código a partir do projeto - crescente tendência em plataformas de *middleware* - visando a redução do esforço de codificação e o aumento da confiabilidade do código da aplicação.

1.3 Contribuições do Trabalho Proposto

As seguintes contribuições foram identificadas no âmbito desta tese:

- definição de um modelo independente de tecnologia para o desenvolvimento orientado a componentes de *software* de novas aplicações, definido por meio da linguagem UML (*Unified Modeling Language*);
- construção de protótipos de plataformas de suporte ao modelo de componentes de *software* proposto;
- validação das plataformas (e por conseguinte, do modelo) pelo protótipo de uma aplicação telemática e ubíqua de grande porte, o laboratório virtual REAL[1].

Consideramos estas contribuições um passo importante para atender aos requisitos impostos pelas aplicações telemáticas e ubíquas listados na seção anterior. Em adição, o desenvolvimento baseado em componentes de *software* atende o requisito de suporte para a integração de componentes em oposição à programação. A integração entre os paradigmas de componentes de *software* e agentes móveis atende o requisito de suporte para novos paradigmas de computação distribuída. O suporte a interfaces de fluxo contínuo com facilidades de qualidade de serviço atende ao requisito de suporte para interações baseadas em multimídia distribuída em tempo real. Finalmente, o requisito de suporte para adaptação dinâmica das aplicações aos ambientes de execução pode ser atendido pelo emprego de agentes móveis como relatado em Guimarães [25].

Modelo para o Desenvolvimento Orientado a Componentes de Software de Novas Aplicações

A principal contribuição deste trabalho é a proposta e validação experimental de um modelo para o desenvolvimento orientado a componentes de *software* de aplicações telemáticas e ubíquas. O modelo proposto pela autora, denominado CM-tel[2] [26][25] apresenta características inéditas em relação aos modelos de componentes existentes.

O modelo CM-tel é independente de tecnologia e facilita o processo de desenvolvimento de *software* fornecendo uma solução integrada em que os componentes de *software*, agentes móveis e ambientes de execução são fornecidos de forma automatizada por uma plataforma, a partir de especificações de alto nível e também independentes de tecnologia. Alguns aspectos não-funcionais proporcionados incluem distribuição, mobilidade de código, comunicação multimídia distribuída com facilidades de qualidade de serviço e acesso transparente aos serviços requisitados por componentes e agentes.

[1] *Remotely Accessible Laboratory.*
[2] *Component Model for telematic applications.*

O modelo de componentes de *software* proposto utiliza o mecanismo de perfis (extensões) da linguagem UML (*UML profile*) para definir os seus artefatos (componentes e seus ambientes de execução) que serão usados para descrever aplicações telemáticas e ubíquas de forma independente de tecnologia. Um perfil UML [27] é uma especificação que especializa o metamodelo UML [28] para um domínio específico de referência. O perfil UML definido permite validar os componentes e ambientes de execução aderentes ao modelo proposto e especificados em UML. Posteriormente, o modelo é mapeado para plataformas de *middleware* com diferentes tecnologias de implementação, mantendo as mesmas especificações em UML. Acreditamos que o desenvolvimento utilizando um alto nível de abstração em UML terá um papel central nas propostas das plataformas de suporte da próxima geração. As futuras aplicações distribuídas se concentrarão na especificação e terão a maior parte do código gerada automaticamente.

Componentes especificados segundo o perfil UML definido pelo modelo CM-tel oferecem um conjunto de interfaces que abrangem as três interfaces prescritas pelo modelo de referência RM-ODP: interfaces de fluxo contínuo (ou *stream*) para suporte à comunicação multimídia distribuída, interfaces de notificação (de eventos) e interfaces operacionais para suporte à interação tipo cliente/servidor. A combinação destas três interfaces integradas no mesmo modelo de componentes é inédita e possibilita especificar interações de elevada complexidade pela composição de portas complementares no nível de projeto. O processo de composição é bastante simplificado, o que viabiliza inclusive a construção de plataformas leves para o modelo (por exemplo, para execução em dispositivos móveis sem fio). Em adição, o modelo fornece uma interface para a especialização de componentes.

O modelo de componentes CM-tel proposto é inovador também no que diz respeito ao ambiente de execução, os chamados contêineres. Nos modelos de componentes existentes, o papel reservado ao contêiner é a instanciação e controle do ciclo de vida dos componentes, além de prover funções, tais como transação, persistência e segurança. No modelo de componentes proposto, o contêiner tem um papel adicional, qual seja, abrigar também código móvel (isto é, o contêiner desempenha o papel de agência na terminologia de agentes móveis). Desta forma, além de suportar de forma integrada as interfaces citadas acima, uma solução integrada entre os paradigmas de agentes móveis e de componentes de *software* também é facilitada, posto que ambos compartilham o mesmo ambiente de execução. O ponto forte desta solução integrada é permitir a sua utilização em alto nível sem que o desenvolvedor se preocupe com os detalhes de acesso a estas funcionalidades providenciadas pelo contêiner. De forma semelhante a componentes, o contêiner fornece uma interface para a sua especialização, bem como para a gerência do conjunto de componentes e agentes hospedados.

Plataformas de Suporte ao Modelo de Componentes de Software CM-tel

Plataformas de suporte ao modelo de componentes de software CM-tel viabilizam a geração, instalação, configuração e gerência de componentes, contêineres e agentes móveis para uma dada tec-

nologia. O trabalho proposto apresenta uma estratégia de desenvolvimento de plataformas baseada no conceito de mapeamento. Um mapeamento pode ser visto como uma transformação de uma especificação neutra (no caso, em linguagem UML) para uma especificação em uma dada tecnologia (por exemplo, CORBA). Este trabalho propõe uma estratégia geral de mapeamento baseada em transformações XML (*Extensible Markup Language*) [29].

Para a construção do protótipo da primeira plataforma para o suporte ao modelo de componentes de *software* CM-tel, denominada plataforma CCM-tel[3] [30][25], escolhemos um mapeamento para as tecnologias CORBA e Java. A primeira versão da ferramenta de geração de código para esta plataforma foi desenvolvida pelo aluno Miglinski, como tema de sua dissertação de mestrado [2]. Esta ferramenta gera código com o emprego de um analisador XML, que transforma especificações XML de componentes e contêineres CM-tel em código Java e CORBA.

Uma versão mais elaborada desta plataforma, desenvolvida pela autora com contribuição do professor Eleri Cardozo, inclui o desenvolvimento e integração na plataforma dos quatro serviços CORBA de suporte que foram utilizados nos protótipos[4]. Nesta nova versão, a plataforma CCM-tel emprega transformações XSLT (*XML Stylesheet Language Transformation*) [31] na ferramenta para geração de código a partir de especificações XML de componentes e contêineres do modelo CM-tel. O aluno Gomes [32] contribuiu com a implementação de novas transformações XSLT na plataforma CCM-tel (transformação de UML para XML). Estas transformações possibilitam a geração de código a partir de especificações de componentes e contêineres em UML, conforme definido pelo modelo CM-tel. A inclusão de facilidades de qualidade de serviço no nível do modelo, do serviço de comunicação multimídia e da plataforma foram adicionados pela autora, enquanto que no nível de rede houve a contribuição dos alunos Pinto [33][34] e Souza [35].

O protótipo de uma segunda plataforma, denominada WSCM-tel, está em desenvolvimento e contempla a implementação de um mapeamento do modelo CM-tel proposto para as tecnologias de Serviços Web (SOAP[5], WSDL[6] e UDDI[7]) e J2ME (*Java 2 Micro Edition*). Este protótipo gera código para utilização tanto em dispositivos móveis sem fio de baixo poder computacional, quanto em computadores convencionais. Este protótipo faz parte da dissertação de mestrado de Gomes [32].

Validação do Modelo e da Plataforma de Suporte

Uma contribuição que também consideramos importante é a validação do modelo CM-tel e da plataforma CCM-tel. Para isto, desenvolvemos dois protótipos de uma aplicação telemática e ubíqua

[3] *CORBA Component Model for telematic applications.*
[4] Serviços de eventos, de propriedades, de comunicação multimídia distribuída e de gerência de agentes móveis.
[5] Simple Object Access Protocol [36].
[6] Web Service Description Language [37].
[7] Universal Description, Discovery and Integration [38].

no âmbito do laboratório virtual REAL [39][40][30]. Esta aplicação permite o acesso remoto, através da intranet ou Internet, a equipamentos robóticos autônomos e móveis localizados no CenPRA[8].

O projeto REAL foi iniciado em 1997 pelo Dr. Marcel Bergerman, pesquisador do CenPRA na época. O Dr. Bergerman coordenou o projeto até o seu desligamento do CenPRA em 2001. No desenvolvimento do primeiro protótipo do laboratório REAL utilizamos uma plataforma de objetos distribuídos (CORBA e Java), implementada no início desta tese e, no segundo protótipo utilizamos a plataforma CCM-tel. Participaram do desenvolvimento do primeiro protótipo os alunos Maffeis [41] e Pereira [42] (modo avançado de interação), Russo [43] (modo básico de interação), a autora (modo assistido de interação) e Pinto [25] (sessão de acesso aderente a TINA). Esta implementação foi demonstrada na mostra de Ciência e Tecnologia para o Desenvolvimento (CIENTEC) [44], onde o usuário localizado na UNICAMP enviou e acompanhou remotamente a execução dos seus algoritmos de navegação executados pelo robô localizado no CenPRA.

O segundo protótipo foi desenvolvido pelos alunos Maffeis (modo avançado de interação), Souza [45] (modo básico de interação), Sassi [46] (modo assistido de interação) e pela autora (nova sessão de acesso e aplicação de controle de QoS [25]). A autora atua como coordenadora do projeto REAL junto ao CenPRA, tendo participado da concepção dos protótipos das aplicações.

Para a validação da segunda plataforma (WSCM-tel), duas aplicações estão sendo desenvolvidas para a utilização de dispositivos móveis como terminais de acesso: uma primeira aplicação na área de gerência de redes e uma segunda que reutiliza o projeto do modo básico de interação do laboratório REAL. Estas aplicações estão em fase de desenvolvimento pelos alunos Gomes e Pereira [47], respectivamente.

1.4 Trabalhos Correlatos

A proposta de um novo modelo para o desenvolvimento orientado a componentes de *software* para aplicações emergentes envolve a integração de vários resultados de pesquisa e cobre muitos aspectos relacionados às áreas de engenharia de *software*, sistemas distribuídos, mobilidade e redes de computadores. Exemplos de tais temas incluem desenvolvimento orientado a componentes de *software*, suporte aos aspectos não-funcionais das aplicações, geração automática de código, comunicação multimídia distribuída, suporte à mobilidade, suporte à adaptação dinâmica da aplicação ao ambiente de execução e suporte à qualidade de serviço (QoS) para fluxos de mídia contínua. A literatura apresenta contribuições importantes a estes temas, mas de forma isolada (ou seja, sem qualquer tentativa de integrá-los).

A necessidade da integração dos temas citados acima em um modelo coerente fez com que, re-

[8]Centro de Pesquisas Renato Archer.

centemente, o consórcio OMG anunciasse duas RFPs (*Request for Proposals*) com o objetivo de acrescentar ao modelo de componentes CCM o suporte para a comunicação baseada em fluxos de mídia contínua (*stream*) [48] e para a negociação de contratos de QoS [49]. Os objetos destas RFPs são contemplados pelo modelo de componentes de *software* proposto nesta tese, cujas soluções foram publicadas já em 2000 [50][51][26][34].

Os trabalhos correlatos aqui citados foram os que mais se aproximaram da proposta desta tese em termos de identidade dos temas que abordam. A autora desconhece outros trabalhos que contemplem no mesmo grau de integração as propostas apresentadas no trabalho aqui proposto. Dividimos os trabalhos correlatos em cinco categorias:

1. extensões de modelos de componentes e plataformas de *middleware* existentes;
2. trabalhos na linha do padrão MDA (*Model-Driven Architecture*);
3. novos modelos de componentes de *software*;
4. temas correlatos (QoS, adaptação dinâmica de aplicações e mobilidade de código);
5. infra-estruturas e aplicações telerobóticas distribuídas.

1.4.1 Extensões de Modelos e Plataformas de Componentes

Em termos de suporte ao desenvolvimento orientado a componentes de *software*, a proposta mais comum tem sido a de implementar uma plataforma que emprega um modelo de componentes de *software* bem estabelecido e aceito. Esta solução foi adotada pelas plataformas OpenCCM e OpenEJB [52], duas plataformas de código fonte aberto que implementam uma parte da especificação fornecida para os modelos de componentes de *software* CCM e EJB, respectivamente.

Os modelos CCM e EJB utilizam muitos conceitos básicos do paradigma de componentes, igualmente utilizados pelo modelo de componentes de *software* proposto nesta tese. No entanto, eles não propõem soluções para os requisitos fundamentais necessários para as aplicações emergentes, conforme citado anteriormente. As diferenças mais significativas com relação a nossa proposta serão descritas nos próximos capítulos.

Alguns trabalhos estendem os modelos de componentes de *software* existentes, adicionando novas funcionalidades. Outros dotam as plataformas de *middleware* de algum suporte ao desenvolvimento orientado a componentes. Alguns esforços nesta direção são o modelo de componentes QoS [53], a plataforma de componentes CIAO [54] e o projeto Cadena [55].

O modelo de componentes QoS estende o modelo EJB com o objetivo de adicionar a esta especificação os requisitos de QoS. O novo modelo introduz um novo tipo de componente, denominado

QoSBean, em adição aos três tipos de componentes definidos pelo EJB (*Session Bean*, *Entity Bean* e *Message-driven Bean*). Este novo tipo de componente define semânticas e atributos de configuração baseados nas funcionalidades de QoS declaradas no descritor de distribuição do EJB. O contêiner gerado proporciona negociação e reserva de recursos no nível de rede para o *QoSBean*.

A plataforma CIAO propõe soluções para temas tais como componentes de *software*, qualidade de serviço e requisitos de adaptação. A plataforma implementa em C++ o modelo de componentes CCM e é construída sobre um ORB (*Object Request Broker*) denominado TAO [56], que por sua vez é uma implementação do ORB da especificação CORBA com funcionalidades de tempo real segundo a especificação RT-CORBA (*Real Time CORBA*) [57] do OMG. TAO é otimizado para sistemas distribuídos e embutidos de tempo real.

O objetivo do projeto Cadena é o desenvolvimento de um ambiente integrado para a construção e modelagem de sistemas distribuídos usando CCM. Cadena proporciona facilidades para definir tipos de componentes por meio da linguagem IDL (*Interface Definition Language*) do CCM, para configurar sistemas de componentes CCM e uma variedade de facilidades para verificação de modelos e para análise dos sistemas configurados. Em adição, proporciona suporte para geração automática de código de componentes a partir de especificações IDL usando duas implementações CCM: OpenCCM (geração de código de *stubs e skeletons* na linguagem Java) e CIAO (geração de código de *stubs e skeletons* na linguagem C++). Cadena é implementado como uma extensão (*plug-in*) no ambiente de desenvolvimento integrado Eclipse [58] da IBM, onde proporciona um conjunto de editores para arquivos IDL do OMG. Esta extensão acrescenta ao ambiente Eclipse editores dedicados para a linguagem IDL e CIDL (*Component Implementation Definition Language*) do padrão CCM.

A principal diferença observada ao se comparar o modelo de componentes proposto com os trabalhos citados acima é que em vez de adotar ou estender um modelo de componentes existente, como nestes trabalhos, optamos por propor um novo modelo de componentes de *software* neutro e independente de tecnologia.

1.4.2 MDA (Model-Driven Architecture)

Durante o desenvolvimento desta tese, o OMG publicou uma especificação de arquitetura de *software* denominada MDA (*Model-Driven Architecture*) [59]. A idéia básica do MDA é a mesma adotada para a parte de independência de tecnologia proposta nesta tese, qual seja, especifica-se inicialmente um modelo neutro para a aplicação, denominado PIM (*Platform Independent Model*). Em seguida, utilizando-se transformações, gera-se a partir deste outros modelos dependentes de tecnologias denominados PSMs (*Platform Specific Model*). As ferramentas em desenvolvimento para MDA gerarão código a partir da especificação PSM.

Na primeira parte da nossa abordagem, um perfil UML para o domínio de aplicações telemáticas

e ubíquas define o modelo de componentes de *software*, onde componentes e contêineres da aplicação são descritos segundo este perfil de forma independente de tecnologia. A seguir, os componentes e contêineres são transformados em uma representação intermediária em XML. Esta representação em XML acrescentada, por exemplo, das interfaces IDL geradas para o mapeamento do modelo para a tecnologia CORBA, constitui-se exatamente de um PSM da especificação MDA. Na nossa proposta, a partir desta representação intermediária procede-se à geração automática da maior parte do código da aplicação (também por transformação).

O consórcio OMG ainda não definiu a linguagem de transformação de PIMs para PSMs. Porém, o consórcio OMG anunciou outra RFP, com o objetivo de padronizar esta linguagem de transformação. Na nossa plataforma, utilizamos transformações XSLT padronizadas pelo consócio W3C (*World Wide Web Consortium*) [31]. Acreditamos que a linguagem a ser definida pelo OMG será muito próxima da linguagem XSLT, dado que modelos PIMs são descritos em UML que por sua vez, possuem representação XML equivalente segundo o padrão XMI (*XML Metadata Interchange*) [57].

Alguns projetos de pesquisa na linha do padrão MDA utilizam regras de transformação para mapeamentos entre modelos diferentes e para domínios específicos empregando linguagens de transformação proprietárias. Uma utilização do padrão XSLT para transformar PIM em PSM é descrita em Batista [60]. Nesta publicação, o modelo PIM segue a sintaxe proposta pelo perfil UML para processos de negócios da especificação EDOC (*Enterprise Distributed Object Computing*) [61] do OMG. O modelo PSM é representado por interfaces IDL da tecnologia CORBA.

1.4.3 Novos Modelos de Componentes de Software

COMQUAD

O modelo de componentes COMQUAD (*Components with Quality properties and Adaptivity*) [62] tem como objetivo propiciar a especificação e suporte em tempo de execução de alguns aspectos não-funcionais do sistema, tais como propriedades de QoS (tempo de resposta, precisão, banda de rede e uso de CPU) e propriedades de mais alto nível para o usuário final (qualidade de vídeo e número de clientes ativos no servidor) para o domínio de aplicações tempo-real. Este modelo, definido recentemente, utiliza alguns conceitos dos modelos EJB e CCM, bem como adiciona a estes a especificação dos aspectos não-funcionais citados (realizados via interceptadores como mecanismo de programação reflexiva), a gerência pelo contêiner da instanciação e conexão de implementações de componentes, além de interfaces de fluxos (*streams*). COMQUAD suporta ainda adaptação dinâmica de componentes via interceptadores disponíveis no servidor de aplicação JBOSS [63].

COMQUAD restringe o modelo de componentes em função da tecnologia à qual está vinculado. Está ainda em desenvolvimento uma implementação de um contêiner sobre o sistema operacional

de tempo real DROPS [61]. Componentes COMQUAD da aplicação executarão parte sobre este contêiner e parte sobre o servidor de aplicação JBOSS. Outra restrição imposta pela vinculação de aspectos dependentes da tecnologia (recursos tempo real) é a imposição de que componentes sofram composição apenas através do contêiner. Esta restrição visa garantir que os recursos necessários para as interações de tempo real serão corretamente alocados. COMQUAD exige ainda a descrição de componentes em COMQUAD IDL, uma extensão de CCM-IDL para incorporar interfaces de fluxo contínuo e o uso de interceptadores portáveis CORBA para monitoração e adaptação dinâmica de componentes. Com estas restrições, a utilização ampla do modelo fica comprometida.

A proposta apresentada nesta tese de desvincular o modelo de componentes da tecnologia de implementação é uma vantagem em relação ao modelo COMQUAD. Por exemplo, no modelo CM-tel é possível implementar uma outra plataforma que represente um mapeamento para um ambiente de tempo real. As particularidades deste ambiente ficariam circunscritas ao mapeamento (mais especificamente, ao descritor de distribuição deste mapeamento), e não no modelo de componentes como no COMQUAD. Modelos neutros permitem ainda a especificação de componentes em alto nível, como UML, em oposição a notações vinculadas a tecnologias como IDLs.

A plataforma que implementa o modelo CM-tel para as tecnologias CORBA/Java permite também o uso de interceptadores portáveis CORBA para adaptação dinâmica do comportamento de componentes, de forma semelhante ao COMQUAD. Interceptadores permitem ainda incorporar novas funções à plataforma tais como criptografia, controle de acesso, auditoria, monitoramento e segurança. Quanto à adaptação dinâmica ao ambiente de execução, nosso modelo permite um segundo mecanismo ainda mais rico que interceptadores, agentes móveis.

Finalmente, apesar de certas similaridades com o modelo aqui proposto, notadamente interfaces de fluxos, de sinal e capacidade de adaptação dinâmica para comportamento de componentes e ambiente de execução, os artigos sobre o modelo COMQUAD publicados recentemente informam que aspectos de implementação ainda serão divulgados e não citam aplicações que fazem uso deste modelo, sugerindo um estágio preliminar de desenvolvimento.

OpenORB

A plataforma OpenORB [64] e seu modelo de *middleware* reflexivo apresentam uma solução construída no topo de um modelo de componentes denominado OpenCOM [65], derivado do modelo centralizado COM [66] da Microsoft. Portanto, diferentemente da nossa proposta, OpenORB trata de uma proposta dependente de tecnologia (CORBA, C++ e plataforma Windows). OpenORB apresenta uma plataforma de *middleware*, onde os componentes são organizados em dois espaços (níveis): nível base e nível meta. Os componentes pertencentes ao nível base implementam os serviços que são comumente encontrados nos *middlewares* existentes. Os componentes pertencentes ao nível meta

são eventualmente conectados àqueles do nível base e oferecem facilidades que permitem aos programadores inspecionar e adaptar a plataforma às necessidades da aplicação.

Nossa proposta para adaptação dinâmica baseada em código móvel não exclui o uso de reflexão, mas deixa para a plataforma dependente de tecnologia a sua implementação. Por exemplo, um mapeamento poderia incluir componentes situados no nível meta (como no caso do OpenORB), utilizar a introspecção oferecida pela linguagem de programação (como no caso de Java) ou ainda utilizar propriedades para inspeção e alteração de parâmetros da plataforma[9]. Para um modelo independente de tecnologia, acreditamos que o único suporte à introspecção possível seria o componente expor, via interface ou propriedades, a sua especificação em uma linguagem neutra.

A plataforma OpenORB define uma API (*Application Programming Interface*) básica [67] para suporte a conexões de componentes ponto-ponto. Esta API pode ser estendida para suportar diferentes tipos de conexões (*bindings*) entre componentes OpenCOM, tais como invocação síncrona e assíncrona de métodos remotos, filas de mensagens, esquemas publicador/subscritor, fluxos de mídia contínua e protocolos de alocação de recursos. A desvantagem da estratégia adotada pelo OpenORB aos diferentes tipos de conexões é que a API básica deve ser estendida pelo desenvolvedor da aplicação, com a adição de novas funções, para cada tipo de conexão que a aplicação venha a requerer. Esta estratégia é bastante limitada dado que exige do desenvolvedor o fornecimento de todo o código para a realização da conexão.

Uma linha de trabalho semelhante à API da plataforma OpenORB relata uma arquitetura [68] que propõe a construção de aplicações distribuídas utilizando componentes de interação denominados *mediuns* para diferenciá-los dos componentes funcionais da aplicação. *Mediuns* encapsulam serviços ou protocolos de comunicação reusáveis (por exemplo, difusão de vídeo, protocolos de votação e caixas de mensagens). Desta forma, os *mediuns* propiciam aos componentes funcionais um conjunto restrito de esquemas de interações. *Mediuns* não executam em contêineres, sendo portanto entidades distintas dos componentes de aplicação que os utilizam.

O modelo proposto nesta tese provê as duas formas de interação (ponto-ponto e ponto-multiponto) para cada um dos três tipos de interfaces prescritos pelo modelo RM-ODP. Este modelo de interação é genérico o suficiente para atender a maioria das aplicações distribuídas, conforme descrito na Seção 3.2.2. Outra característica importante do modelo proposto é permitir a adição de novos modos de conexão com a utilização de componentes baseados no modelo e não por extensões de APIs como no projeto OpenORB. A plataforma que implementa o mapeamento do modelo CM-tel para as tecnologias de Serviços Web e J2ME gera componentes de configuração a partir de diagramas UML de colaboração [32]. Acreditamos que esta é a maneira desejável pela qual as aplicações podem

[9] A plataforma implementada para CORBA-Java no escopo desta tese utiliza propriedades para inspeção e alteração de parâmetros de qualidade de serviço podendo ser considerada uma forma simplificada de instrospecção.

obter benefícios ao especificar diferentes formas de conexão em alto nível e não pela utilização de APIs que exigem do desenvolvedor da aplicação grande esforço de codificação.

Finalmente, uma característica fundamental, recomendada pela engenharia de *software* para modelos de componentes e que é adotada pelo nosso modelo de componentes, é a capacidade do componente armazenar as referências do outro extremo da conexão [11]. Este mecanismo, denominado receptáculo no CCM[10], armazena referências de interfaces necessárias para a execução deste componente e é descrito na Seção 3.2. Dos modelos de componentes reportados na literatura, além do CCM, apenas os modelos CM-tel e COMQUAD possuem este mecanismo.

1.4.4 Trabalhos com Temas Relacionados

Qualidade de Serviço e Adaptação

O projeto QuA (*Quality of Service Aware Component Architecture*) [69] apresenta uma outra solução de arquitetura construída no topo do modelo de componentes OpenCOM. O principal objetivo do projeto QuA é investigar como desenvolver aplicações de tempo-real sobre uma plataforma de componentes e avaliar por experimentação como a gerência e adaptação de QoS podem ser suportadas em arquiteturas de componentes de propósito geral. Entre os trabalhos em desenvolvimento deste grupo, um possui uma característica semelhante à nossa proposta que é a produção de transformadores para mapear especificações de QoS em alto nível (UML) em configurações dependentes de plataforma. Um metamodelo de QoS define as semânticas para especificações de QoS genéricas, independentemente de arquitetura de *middleware* ou de componente. Está em desenvolvimento uma plataforma de acordo com o padrão MDA para a execução e distribuição de componentes *QoS-aware*.

No contexto de suporte a QoS e sua realização em plataformas de *middleware* baseadas em componentes, outro trabalho apresenta um perfil UML para QoS que permite uma especificação de contratos de QoS negociados entre componentes de *software* [70]. O perfil UML descreve extensões para o modelo CCM de forma a suportar qualidade de serviço nos componentes e na infra-estrutura da plataforma Qedo (*QoS Enabled Distributed Objects*) [71]. O trabalho apresenta uma extensão para o modelo CCM, onde interfaces de fluxo (*streams*) são incorporadas ao modelo CCM e a garantia de banda para o fluxo é especificada por meio de um contrato de QoS. Desta forma, os componentes aderentes ao modelo CCM podem negociar um contrato antes que iniciem a sua interação. Os parâmetros deste contrato serão submetidos para os respectivos contêineres dos componentes produtor e consumidor de fluxo por meio de descritores de distribuição CCM. O uso de interceptadores portáveis CORBA é proposto como um mecanismo de extensão da plataforma Qedo para incorporação de suporte à QoS. Interceptadores portáveis CORBA também foram adotados na plataforma desenvolvida

[10]Mantivemos a mesma denominação em nosso modelo.

no escopo desta tese para esta mesma finalidade de suporte à QoS.

Agentes Móveis

Na literatura especializada encontramos poucos trabalhos publicados na linha de modelos de componentes que forneçam suporte de forma integrada a componentes de *software* e agentes móveis. Dois projetos nesta linha são DynamicTAO [72] e COBE Framework [73].

DynamicTAO é uma plataforma reflexiva que implementa um ORB CORBA desenvolvido em C++, para suportar reconfiguração dinâmica dos ORBs participantes. Em adição, suporta facilidades de QoS para aplicações que têm requisitos brandos de tempo real, além de interceptadores portáveis. Esta plataforma emprega um conjunto de agentes móveis configuráveis para gerência do ORB da própria plataforma de *middleware*. Estes agentes são injetados na rede por administradores de sistema e viajam de um ORB para outro inspecionando-os e, quando necessário, requerendo uma ação de reconfiguração do ORB, de acordo com instruções programadas pelos administradores nos agentes. Diferentemente de nossa proposta, DynamicTAO não permite a utilização de agentes móveis por parte da aplicação.

O *framework* COBE está em fase de desenvolvimento sobre a plataforma DeEVOLVE [74], que por sua vez implementa um modelo de componentes de software para arquiteturas *peer-to-peer*. DeEVOLVE é capaz de detectar e de resolver exceções não previstas antecipadamente tais como falhas em um dos componentes pares ou indisponibilidade de serviços. COBE pode ser utilizado por outras aplicações para notificar pares sobre as mudanças ocorridas na aplicação. O projeto COBE pretende integrar um sistema leve de agentes móveis dentro da plataforma DeEVOLVE. Desta forma, a plataforma permitirá a utilização de componentes e de agentes em arquiteturas *peer-to-peer* para fornecer adaptação desta arquitetura em tempo de execução. Atualmente, os agentes são desenvolvidos no sistema de agentes JADE que é executado em paralelo com o sistema DeEVOLVE.

A integração de agentes móveis e componentes perseguida pelo projeto COBE vem ao encontro da nossa proposta que já disponibiliza um sistema leve de agentes móveis para utilização pelas aplicações em uma plataforma de componentes de *software*. A plataforma para as tecnologias CORBA/Java desenvolvida integra agentes móveis pela instanciação em contêineres de agências que suportam agentes móveis. Por compartilharem o mesmo contêiner, componentes e agentes são perfeitamente integrados em termos de interação componente-agente e acesso a recursos comuns propiciados pelo contêiner.

1.4.5 Infra-estrutura e Aplicações Telerobóticas Distribuídas

Um trabalho de infra-estrutura de suporte para o domínio de aplicações telerobóticas distribuídas está em desenvolvimento no laboratório RIMLab (*Robotics and Intelligent Machines Laboratory*) da

Universidade de Parma, Itália [75]. Este projeto tem como objetivo o desenvolvimento de um *framework* de classes para sistemas telerobóticos que utiliza a arquitetura CORBA com extensões de tempo real (RT-CORBA) propiciadas pela plataforma ACE-TAO [56]. Tais extensões suportam prioridade na invocação de métodos, invocação assíncrona de métodos e uso de *pool* de *threads* nos servidores. Exemplos de aplicações alvo para este *framework* são laboratórios virtuais, teleoperação e colaboração distribuída. Este *framework* utiliza serviços CORBA (controle de concorrência, eventos e de notificação) para que múltiplos clientes possam controlar um robô e para assegurar que o servidor realize operações de tempo real e de gerência de controle concorrente, bem como transferência de dados. No *framework* do RIMLab, um único usuário tem o controle do robô pelo uso de um *lock* de exclusão mútua provido pelo serviço de concorrência da plataforma CORBA. Cada recurso compartilhado estará associado com um *lock* e um cliente deve solicitar o *lock* apropriado para acessar o recurso.

Dois trabalhos de desenvolvimento de aplicações na linha de laboratórios virtuais são apresentados a seguir. Dos trabalhos pesquisados na literatura especializada, estes foram os que apresentaram funcionalidades mais próximas à nossa implementação de laboratório virtual.

O projeto TIGER [76] é um projeto de aprendizagem eletrônica para a área de robótica que tem como objetivo permitir a estudantes controlar veículos autônomos através da Web. Este projeto proporciona funcionalidades de telepresença para acompanhamento do experimento por meio de fluxos de áudio e imagens de vídeo. Além destas funcionalidades, proporciona também alguns instrumentos clássicos de comunicação providos pela Web: *chat*, *e-mail* e fóruns de discussão. Funcionalidades específicas para simulações são igualmente propiciadas. Este projeto não disponibiliza nenhuma forma de controle de acesso aos robôs.

Khamis [77] apresenta um trabalho no âmbito do projeto IECAT [78] e na linha de pesquisa de laboratórios de acesso remoto, que oferece alguns experimentos em robôs móveis para estudantes. Este trabalho propõe uma arquitetura de *software* para realização de experimentos previamente desenvolvidos que são disponibilizados para uso em cursos de sistemas robóticos móveis e autônomos a distância. Os experimentos cobrem sensoriamento, planejamento de trajetória e de tarefas, localização e controle inteligente de robôs móveis. Para acompanhar estes experimentos, o sistema disponibiliza interfaces no navegador do lado do usuário para interagir com os robôs. Estas interações dos usuários com o laboratório remoto são realizadas sem necessidade de nenhum *software* adicional ou *plug-ins*. A interação com o usuário se dá por meio de *applets* Java que interagem com *servlets* Java no lado servidor. Estes *servlets*, por sua vez, atuam como clientes CORBA para acesso aos serviços disponibilizados pelo robô.

Comparando-se estes projetos com o laboratório virtual REAL, observa-se inicialmente que estes foram desenvolvidos por meio de plataformas de objetos distribuídos, enquanto REAL foi desen-

volvido sobre um modelo de componentes de *software* neutro. Esta diferença entre objetos distribuídos e componentes de *software* é extensivamente analisada no Capítulo 2. A especificação de componentes de forma neutra se dá em nível alto de abstração, permitindo o reuso do projeto da aplicação por meio de plataformas de componentes que implementam o modelo de componentes em diferentes tecnologias. A capacidade de geração automática de código e a separação entre configuração, instalação e montagem dos componentes são também pontos importantes. A título de exemplo, a sessão de comunicação do REAL composta de dois fluxos de vídeo oriundos de câmeras panorâmica e de bordo do robô é implementada por dois componentes, um produtor e um consumidor de vídeo. Estes componentes são especificados por um diagrama de classes UML. A interação entre estes componentes (montagem da sessão de comunicação) é especificada por um diagrama de colaboração UML. A partir destes diagramas, praticamente (próximo de 100%) todo o código da sessão de comunicação é gerado automaticamente pela plataforma.

Outro ponto de destaque do laboratório vitual REAL é o controle de acesso aos recursos compartilhados (robôs). REAL utiliza uma completa infra-estrutura de acesso baseada em tecnologia J2EE (*Java 2 Enterprise Edition*) [79] com funções de cadastramento de usuários e serviços, reserva antecipada de horário, suporte a diferentes modos de compartilhamento dos recursos e monitoramento do uso de recursos.

Alguns aspectos onde os projetos na linha de aplicações telerobóticas distribuídas citados acima oferecem soluções mais abrangentes às adotadas pelo laboratório virtual REAL são discutidos na seqüência. No REAL, apenas funções básicas de tempo real como escalonamento de *threads* por prioridade são empregadas. Neste aspecto, as soluções de tempo real propostas no *framework* do RIMLab são mais abrangentes. REAL utiliza CORBA tanto no lado servidor quanto no lado do cliente, o que pode inviabilizar o acesso ao laboratório virtual caso algum *firewall* da Internet impeça o tráfego do protocolo de comunicação IIOP (*Internet Inter-ORB Protocol*) empregado no CORBA. Neste aspecto, o sistema desenvolvido por Khamis é mais adequado por utilizar CORBA apenas no lado do servidor, restringindo a comunicação entre o cliente e o laboratório virtual apenas ao protocolo HTTP (*Hypertext Transfer Protocol*). Esta mesma solução é empregada na sessão de acesso do REAL, bem como em desenvolvimentos futuros sobre a plataforma baseada em Serviços Web. Finalmente, ao contrário do projeto TIGER, REAL não oferece qualquer facilidade para a implementação de ambientes simulados.

1.5 Organização do Texto

Este trabalho contém seis capítulos e está estruturado da forma descrita a seguir. O Capítulo 2 apresenta os principais fundamentos relacionados com o paradigma de componentes de *software*. O

Capítulo 3 descreve uma proposta de modelo independente de tecnologias (modelo CM-tel) para o desenvolvimento orientado a componentes de *software* de aplicações telemáticas e ubíquas. O Capítulo 4 apresenta a plataforma CCM-tel que implementa o modelo CM-tel para as tecnologias CORBA e Java. O Capítulo 5 apresenta a validação do modelo CM-tel e da plataforma CCM-tel por meio do protótipo do laboratório virtual REAL. O Capítulo 6 conclui este trabalho tecendo considerações finais com base nos resultados obtidos com o desenvolvimento desta pesquisa e sugerindo possíveis extensões ao trabalho aqui apresentado.

Duas notas sobre convenções adotadas nesta tese:

1. Os termos relacionados com implementação, tais como nomes de classes, métodos, relacionamentos, etc., tanto no corpo do texto quanto nas figuras, foram mantidos em inglês. A razão para tal convenção é manter fidelidade com os termos (todos em inglês) utilizados nas implementações. Estes termos advém, na sua maioria, das especificações publicadas pelo OMG, ISO e W3C.

2. O termo "componente de *software*" será doravante referenciado no texto apenas como "componente".

Capítulo 2

Componentes de Software

O capítulo anterior salientou a importância do paradigma de componentes para a engenharia de *software*, notadamente para o desenvolvimento de aplicações distribuídas emergentes. Este capítulo apresenta o paradigma de componentes como uma evolução do paradigma de objetos. Neste contexto, apresenta-se a infra-estrutura necessária para caracterizar um modelo de componentes. Modelos de componentes, como o proposto nesta tese, devem definir padrões arquiteturais e de projeto para a arquitetura de componentes. Finalmente, os principais modelos abertos de componentes existentes são apresentados, enfatizando as soluções que estes modelos estabelecem em termos dos elementos da arquitetura de *software* que os caracterizam.

2.1 Visão Geral de Componentes de Software

O ciclo clássico de desenvolvimento de *software* composto de quatro fases (análise, projeto, implementação e teste) tem sido empregado desde os primórdios da engenharia de *software*, sendo genérico o suficiente para ser aplicado a qualquer processo de desenvolvimento de *software* [80]. Atualmente, as técnicas de desenvolvimento de *software* devem contemplar também a qualidade, capacidade de integração, especialização e manutenção de sistemas de *software* de natureza distribuída [11].

Um ponto crucial para o *software* de natureza distribuída é a incorporação ao modelo do sistema de fatores pertinentes à distribuição. No ciclo clássico de desenvolvimento, a natureza distribuída do sistema é modelada na fase de projeto onde a arquitetura do *software* é concebida. Comumente, uma arquitetura cliente/servidor é adotada e posteriormente sintetizada na fase de implementação com o auxílio de uma plataforma de *middleware* tal como CORBA, DCOM ou Java RMI. É importante observar que mesmo os modernos processos de desenvolvimento são deficientes para o desenvolvimento de tais sistemas [80]. Esta deficiência se deve à ausência de estratégias que permitam construir sistemas com alto índice de reuso, evolução e geração automática de código.

Recentemente, devido aos avanços da tecnologia e introdução de padrões de desenvolvimento de *software*, associados a ferramentas de apoio, surge um novo paradigma que visa aprimorar os processos de desenvolvimento de *software*: o paradigma de componentes [11]. Na realidade, não se pode considerar uma inovação o conceito de componentes como blocos de construção para o desenvolvimento de sistemas. Os desenvolvedores de *software* sempre tiveram a convicção de que sistemas de grande porte e complexos podiam ser construídos pela interconexão de blocos de construção menores, pré-definidos e oriundos de desenvolvimentos anteriores ou adquiridos no mercado [81]. Entretanto, somente agora as tecnologias envolvidas têm evoluído no sentido de atingir este objetivo.

Os benefícios, com a utilização deste novo paradigma, incluem produção de *software* de forma rápida, com menor custo de desenvolvimento, de alta qualidade e de fácil manutenção. Esses objetivos são atingidos principalmente pelo reuso de componentes, interligação explícita entre os componentes (resultando em baixo acoplamento), alta coesão (componentes são altamente especializados) e geração automática de código. Este novo paradigma, ao combinar as vantagens oferecidas pelas tecnologias tradicionais de *middleware* e as novas tecnologias orientadas a componentes, provê um melhor suporte à interoperabilidade (interação entre componentes de diferentes fornecedores), portabilidade (componentes distribuídos em diferentes plataformas), extensibilidade (adição de novas funcionalidades aos componentes) e coexistência com sistemas legados.

Entre os conceitos empregados no desenvolvimento orientado a componentes, aqueles considerados importantes são descritos a seguir. Outras referências tratam extensivamente destes temas [11] [15][82][83].

2.1.1 Componentes e Conceitos RM-ODP

Para uma melhor compreensão das próximas sessões apresentamos a definição dos principais termos do modelo de referência RM-ODP [7][8] utilizados neste trabalho.

> *Interface* é uma abstração do comportamento de um componente que consiste de um subconjunto de interações deste componente, associado a um conjunto de restrições que descreve quando elas podem ocorrer.
>
> *Elemento de software* é uma seqüência de instruções de um programa que descreve as computações a serem executadas por uma máquina.
>
> *Padrão (Standard)* é um objeto, uma qualidade ou uma unidade de medida que serve como uma base à qual outros se sujeitam, ou pela qual a exatidão ou a qualidade de outros é avaliada.
>
> *Interação* é uma ação entre dois ou mais elementos de *software*.

> *Composição* é a combinação de dois ou mais componentes que produzem um novo comportamento dos componentes envolvidos em um diferente nível de abstração. As características deste novo comportamento são determinadas pelos componentes que participam da composição e pelo modo como eles são combinados.

2.1.2 Definições de Componentes

Na literatura especializada existem dezenas de definições de componentes. Neste trabalho concordamos com as definições propostas por Heineman e Councill (estas definições estão de acordo com as de Szyperski) que publicaram o livro *Component-Based Software Engineering* [11] e se propuseram a apresentar o "estado-da-arte" em desenvolvimento de *software* baseado em componentes. Com base na variedade de textos técnicos, examinados e avaliados da literatura, estes autores apresentaram uma definição rigorosa de um componente e de seus elementos constituintes, bem como a importante relação entre estes e um modelo de componentes associado com a sua implementação e com a sua infra-estrutura de suporte.

> Um *componente* é um elemento de *software* que se sujeita a um modelo de componentes, pode ser instalado independentemente e se compor sem modificação conforme um padrão de composição.
>
> Um *modelo de componentes* define os padrões de interação e de composição para componentes.
>
> Um *padrão de interação* define a forma e as operações que permitem componentes de *software* interagirem, direta ou indiretamente, com outros componentes de *software*.
>
> Um *padrão de composição* define os requisitos que permitem a composição de componentes de *software* visando criar elementos de *software* maiores.

Na literatura especializada encontram-se muitas outras definições de componentes. Com o intuito de mostrar as tendências e interpretações que surgem no paradigma de componentes, apresentamos duas definições a seguir. Constata-se que não existem diferenças substanciais em relação à definição anteriormente apresentada. Szyperski [15] relaciona outras definições.

De acordo com Szyperski, um componente é uma unidade de composição com interfaces especificadas contratualmente e dependências de contextos explícitas, que pode ser instalado independentemente e exposto à composição por terceiros.

Cheesman e Daniels [83] apresentam uma outra forma de definir um componente. Eles definem os vários aspectos que um componente apresenta durante um ciclo de vida de um projeto (requisitos,

especificação, projeto, implementação, montagem, instalação e execução). Estes aspectos mostram as características de um componente, bem como as várias formas que ele adquire neste ciclo.

De acordo com esta definição, um componente possui:

- uma *especificação de componente*, que descreve o que o componente faz. A maior parte desta especificação é a definição das interfaces do componente;
- pelo menos uma *implementação do componente*, em que um componente é implementado em uma linguagem de programação;
- um *padrão de componentes*, ao qual um componente deve se sujeitar, para que seja possível construir aplicações pela composição destes elementos de *software*;
- um componente é empacotado em arquivos que armazenam a sua implementação e as suas dependências;
- um componente é instalado em determinado processador quando o arquivo que o empacota é transferido e processado neste processador.

A definição de componente empregada neste trabalho de tese está em concordância com as definições apresentadas acima. Algumas definições são mais restritas que outras, por exemplo a definição de Booch [84], que considera componente qualquer elemento de *software* que pode ser empacotado tais como bibliotecas, arquivos, códigos executáveis e documentos. Neste caso, a definição de um componente é muito genérica, sendo este definido como uma parte física e substituível de um sistema que está de acordo com um conjunto de interfaces e proporciona a realização destas interfaces. Esta última definição não está de acordo com a apresentada nesta tese.

2.1.3 Componentes X Objetos

Meyer [85] afirma que a noção de componentes é um refinamento do modelo de objetos, sendo considerado uma extensão natural deste modelo. Entretanto, existe um consenso na literatura de que os componentes, apesar de ser uma extensão, não são idênticos a objetos. Os conceitos de objetos e de componentes são independentes, embora praticamente todos os modelos de componentes se baseiem no paradigma de programação orientada a objetos.

Algumas semelhanças são apresentadas a seguir:

- ambas as abordagens favorecem a construção de programas a partir de entidades que interagem e colaboram entre si;

- os componentes e as classes (que definem o comportamento e a estrutura de objetos) oferecem as suas funcionalidades por meio de interfaces;
- os componentes e os objetos são instanciados em tempo de execução.

As principais diferenças são apresentadas na Tabela 2.1.

CRITÉRIO	OBJETOS DISTRIBUÍDOS	COMPONENTES
Constituição	Unidade de instanciação	Unidade de composição
Aderência a Padrões	Adere a padrões de Middleware como CORBA	Adere a padrões de componentes como CORBA Component Model
Interação	Implícita na implementação do objeto	Explícita nas dependências de contexto do componente
Especialização	Via herança em tempo de projeto e implementação	Via configuração em tempo de instalação e execução
Acoplamento	Alto (objetos interagem com muitos objetos)	Baixo (componentes interagem com poucos componentes)
Granularidade	Baixa (implementação demanda pouco código)	Alta (implementação demanda muito código)
Composição	Ad hoc	Via padrão de composição
Ambiente de Execução	Genérico (processo)	Específico (contêineres)
Coesão	Baixa (dependente de uma hierarquia de classes)	Alta (funcionalidades auto–contidas no componente)
Evolução	Via redefinição da interface	Via adição de novas interfaces

Tab. 2.1: Diferenças entre objetos distribuídos e componentes.

2.2 Modelo de Componentes

Segundo Heineman e Councill [11], o objetivo da ESBC é o desenvolvimento de sistemas de *software* pela composição de componentes reutilizáveis. Estes componentes necessitam de padrões para interação e composição, como também de infra-estruturas e serviços padronizados. O desafio da ESBC é estabelecer modelos de componentes com tais padrões e proporcionar plataformas que implementam estes modelos de forma a permitir que componentes e infra-estruturas de componentes sejam projetados, implementados e instalados. Eles apresentam as definições a seguir:

> Um *modelo de componentes* define os padrões de interação e de composição para componentes. A *implementação de um modelo de componentes* é um conjunto de elementos de *software* dedicados que são necessários para suportar a execução de componentes aderentes ao modelo.

> Uma *infra-estrutura de componentes* assegura que um sistema ou subsistema de *software*, construído segundo um modelo de componentes, satisfaz as especificações para o componente.

Infelizmente, não existe um consenso relativo à terminologia a ser utilizada para os termos acima. Por exemplo, Bachman [13] denomina *framework* de componentes a infra-estrutura de componentes e *ambiente de execução* a implementação de um modelo de componentes. De forma semelhante à terminologia, também não existe concordância sobre o que um modelo de componentes necessita conter.

Conforme descrito por Farias [86], "um *modelo de componentes* proporciona as diretrizes para criar, implementar e instalar componentes, que podem ser compostos para formar uma aplicação. O modelo define a arquitetura básica de um componente, especificando a estrutura das suas interfaces e os mecanismos pelos quais as suas instâncias interagem com o seu ambiente, com instâncias de outros componentes e com o mundo externo".

De acordo com Bachman "um *modelo de componentes* especifica as regras de projeto que precisam ser obedecidas por componentes. Estas regras de projeto aperfeiçoam os aspectos de composição, garantem que os requisitos sejam alcançados e que os componentes possam ser facilmente instalados em ambientes de desenvolvimento e de execução (*run time*). Os *frameworks* de componentes proporcionam os serviços para suportar e impor um modelo de componentes".

Heineman e Councill [11] apresentam uma revisão detalhada da literatura [11] e, no nosso entendimento, este livro sintetiza a visão geral, bem como traduz um consenso em engenharia de *software* com relação aos elementos a serem contemplados em um modelo de componentes. Segundo a mesma referência, um modelo de componentes deve apresentar as seguintes características:

- Definir um conjunto de elementos básicos, descritos nas Seções 2.2.1 a 2.2.8. Estes elementos básicos são:

    1. padrões para especificação do comportamento do componente (especificação de interfaces e especificação de componentes);
    2. esquema de nomes padronizado;
    3. padrões para metadados;
    4. padrões para interoperabilidade;
    5. padrões para especialização;
    6. padrões para composição;
    7. padrões para suporte a evolução;

8. padrões para empacotamento e distribuição.

- Conter alguns padrões especializados para as necessidades de aplicações de domínios específicos. Por exemplo, a composição de componentes em domínios que requeiram atividades concorrentes necessitam de modelos padronizados de *threading* e de mecanismos de sincronização. Os sistemas de processamento distribuído aberto requerem padrões para a invocação de métodos remotos e de segurança. Aplicações de negócios necessitam de serviços de transação e de banco de dados. No caso de aplicações telemáticas, estas necessitam de comunicação multimídia distribuída com qualidade de serviço.

- Definir padrões para a implementação do modelo de componentes, ou seja, o conjunto de entidades de *software* para suportar a execução de componentes que agem de acordo com um modelo particular. Estes padrões, descritos na Seção 2.2.9, proporcionam um ambiente de execução (*run time*) com uma infra-estrutura para o acesso aos serviços básicos, aos serviços horizontais úteis para vários domínios e aos serviços verticais disponibilizando funcionalidades específicas para componentes de um domínio de aplicação.

Os padrões acima citados são classificados como padrões arquiteturais [82]. Não é clara a separação entre os conceitos destes padrões e dos padrões de projeto. Para ajudar na distinção entre os dois conceitos, consideramos a seguinte definição de arquitetura de *software* [87]:

> A arquitetura de *software* de um programa ou de um sistema de computação é a estrutura ou estruturas do sistema que compreendem os componentes de *software*, as propriedades destes componentes visíveis externamente e as relações entre eles.

Segundo Bass [87], o aspecto mais importante desta definição é o termo "*visível externamente*", porque ele define uma distinção natural do projeto que trata das estruturas internas do sistema. Os padrões acima são padrões arquiteturais porque descrevem os aspectos de uma arquitetura de componentes que são visíveis externamente. Outros padrões, como por exemplo o padrão de projeto observador [16], são classificados como padrões de projeto.

2.2.1 Padrões para Especificação do Comportamento de Componentes

Segundo Bertrand Meyer [88], um aspecto importante de um componente é a definição de uma especificação, para cada componente, diferente do próprio componente e que descreve para os seus usuários o que o componente faz.

De acordo com Cheesman e Daniels [83], uma *especificação de componente* é a especificação de uma unidade de *software* que descreve o comportamento de um conjunto de instâncias de um

componente instalado e define uma unidade de implementação. O *comportamento* é definido como um conjunto de interfaces.

Heineman e Councill [11] afirmam que um modelo de componentes define como o comportamento de um componente é descrito por meio de interfaces, de especificações dos aspectos funcionais e não-funcionais e de uma documentação apropriada.

Os aspectos funcionais especificam a natureza sintática e semântica (comportamental) de uma interface do componente. Eles incluem a descrição dos serviços proporcionados pelo componente, a assinatura destes serviços — nome, tipos dos atributos, operações do serviço, argumentos, a maneira como os resultados são retornados, bem como as possíveis exceções que podem ocorrer durante a execução do serviço — e o comportamento associado à operação do componente.

Os aspectos não-funcionais definem requisitos adicionais não relacionados com as funções precípuas do componente. Estes requisitos incluem garantias de performance, precisão, disponibilidade, segurança, escalabilidade, mobilidade, replicação, persistência, tolerância a falhas, transações, ciclo de vida, resolução de nomes, requisitos de armazenagem, além de atributos de qualidade que quando associados a um serviço particular são referenciados como "qualidade de serviço" (QoS).

A maior parte da especificação de um componente é a definição das suas interfaces fornecida por uma *especificação de interfaces* que é um elemento central em um modelo de componentes. O modelo de componentes pode definir um conjunto de interfaces específicas que componentes, sujeitos a este modelo, são obrigados a implementar. Geralmente, estas interfaces serão utilizadas pela implementação do modelo de forma a proporcionar para os componentes alguns serviços que estes necessitam.

O *padrão de interação* do modelo de componentes abrange as interações e todas as dependências de contexto que podem ocorrer entre componentes. Estas dependências precisam ser claramente especificadas de alguma forma, por exemplo em uma documentação do componente. Um padrão de interação especifica o tipo destas dependências. Geralmente são utilizados dois tipos básicos de interação entre componentes: cliente/servidor e publicador/subscritor. Os componentes podem atuar como clientes e servidores através de invocações de métodos. Um componente pode se subscrever em outro componente, denominado publicador, e receber notificações de eventos pré-definidos.

Interface de Componentes

Um conceito fundamental de um componente é que ele possui interfaces claramente definidas. De forma a permitir que um componente seja reutilizado e interaja com outros componentes é necessário que os componentes provedores e clientes estejam de acordo a respeito de um conjunto de interfaces. Basicamente, a habilidade para integrar e armazenar componentes e, a partir desta integração, disponibilizar estes componentes depende fundamentalmente da noção de interface do componente.

A interface de um componente define os seus pontos de acesso. Um componente não está sozinho em um ambiente e o seu propósito é proporcionar algum serviço para tal ambiente. O acesso aos serviços do componente é visível e disponibilizado a usuários da aplicação (clientes, programadores), a outros componentes, às plataformas de suporte e a outros elementos de *software*, através destes pontos. Geralmente, um componente possui múltiplas interfaces correspondendo a diferentes pontos de acesso. Tecnicamente, uma interface é um conjunto de operações "nomeadas" que podem ser invocadas pelos clientes. A semântica das operações é especificada e esta especificação tem um papel duplo: ela se presta a servidores que implementam a interface e a clientes que usam a interface. Heineman e Councill [11] definem um padrão de interface como:

> Um *padrão de interface* permite que elementos de *software* interajam com outros elementos de *software*. Padrões de interface declaram formalmente os elementos que constituem a interface (métodos, exceções, etc.).

A interface esconde a implementação do componente e, sob o ponto de vista do componente, pode consistir de dois tipos: *interface proporcionada* e *interface requerida*. Um determinado componente suporta uma *interface proporcionada* se este componente contém uma implementação de todas as operações definidas por esta interface. Um componente necessita uma *interface requerida* se este componente demanda uma interação definida nesta interface. A especificação de um componente declara todas as *interfaces proporcionadas* e todas as *interfaces requeridas* pelo componente. Uma abordagem semelhante é definida por Szyperski [15].

Algumas linguagens de descrição de interfaces têm sido desenvolvidas, como por exemplo a linguagem IDL [3] do OMG. IDL é uma linguagem neutra de definição de interface, ou seja, é uma notação independente de implementação. No entanto, expõe algumas limitações quanto à especificação de interface no contexto da ESBC.

A ênfase tradicional, em linguagens como IDL, é especificar as funcionalidades proporcionadas, ou seja, a parte sintática da especificação funcional. Uma interface proporcionada por um componente é uma lista de operações com as suas assinaturas. O compilador pode verificar se o que é proporcionado está em conformidade com o que é esperado. No entanto, o que ocorre é que ao projetar um componente o desenvolvedor sabe muito pouco sobre os outros componentes com os quais este componente possivelmente interagirá. Assim, fornecer somente a assinatura não é suficiente para que um componente saiba o que esperar de outro. É necessário fornecer também o seu comportamento. Por esta razão, os aspectos semânticos relacionados a dependências de contexto (contexto de composição e instalação), interações, comportamento e aspectos não-funcionais têm despertado crescente interesse.

Uma interface serve como um contrato entre um componente e os seus clientes. É importante enfatizar a natureza contratual das especificações de interface, onde a interface proporciona, no míni-

mo, um contrato que especifique tanto os serviços que um componente oferece para os clientes (além de proporcionar uma implementação dos serviços de acordo com esta interface), quanto os serviços que ele necessita. Visto que o componente e os seus clientes são desenvolvidos sem nenhum conhecimento mútuo, um contrato padronizado forma uma base comum para que a interação entre eles se realize com sucesso.

Contratos

Os contratos existem em diferentes níveis como, por exemplo, nos níveis sociais ou de negócios e consistem em um acordo estabelecido entre duas partes. Uma das partes, por exemplo o fornecedor, executa alguma tarefa para a outra parte, por exemplo o cliente. Cada uma das partes espera alguns benefícios do contrato e aceita algumas obrigações. O objetivo do contrato é explicitar estes benefícios e obrigações. A mesma idéia aplica-se para o componente. O contrato declara o que o cliente precisa fazer para utilizar um serviço e também declara o que o provedor tem que implementar para corresponder ao serviço prometido.

Segundo Szyperski [15], um *contrato* é uma especificação anexa a uma interface que mutuamente vincula os clientes e provedores desta interface. Os contratos podem cobrir os aspectos funcionais e não-funcionais. Um contrato complementa as operações de uma interface (IDLs) com uma especificação funcional, comumente produzida por pré-condições e pós-condições, e requisitos não-funcionais, freqüentemente chamados nível de serviço ou qualidade de serviço. Os aspectos funcionais incluem as partes sintáticas e semânticas de uma interface.

Segundo Meyer [88][89], a noção de contrato consiste no explícito entendimento entre um provedor e um cliente através de pré-condições, de pós-condições e de invariantes. As pré-condições exprimem, para o cliente, as regras que restringem o uso de um serviço (operação), por exemplo, as condições que precisam ser verdadeiras antes que um componente execute alguma função. Por outro lado, as pós-condições exprimem as garantias do resultado ao invocar a operação desde que respeitadas as pré-condições. As invariantes são asserções que descrevem as propriedades globais que precisam sempre ser verdadeiras para um objeto ou componente. Elas especificam o que precisa permanecer verdadeiro durante o ciclo de vida dos componentes.

Beugnard [90] distingue quatro níveis de contratos para componentes: básico ou sintático, de comportamento, de sincronização e de qualidade de serviço. O contrato básico corresponde à parte sintática da especificação funcional, tais como a existente em linguagens como IDL. O contrato de comportamento especifica o comportamento semântico da operação expressando invariantes, pré-condições e pós-condições. O contrato de sincronização melhora a confiabilidade em contextos distribuídos ou concorrentes. O contrato de qualidade de serviço determina a qualidade que o cliente espera do serviço e é geralmente negociável.

Cheesman e Daniels [83] definem dois tipos de contratos: de uso e de realização. O contrato de uso consiste em um contrato estabelecido entre a interface do componente e um cliente que a utiliza. O contrato de realização consiste em um contrato estabelecido entre a especificação do componente e uma implementação do componente.

Atualmente, não existe concenso quanto a utilização ou padronização de contratos para possibilitar um fundamento comum em relação à parte semântica (comportamental) das especificações funcionais e das especificações não-funcionais. Apesar de apontar alguns problemas com relação a invariantes, pré-condições e pós-condições, Szyperski [91] afirma que esta ainda é a melhor solução para a parte semântica da especificação funcional de operações, principalmente se acompanhadas de invariantes. No entanto, Szyperski enfatiza que contratos necessitam ir além do estabelecimento de cláusulas funcionais (requerida e proporcionada). As propriedades não-funcionais precisam ser incorporadas ao contrato.

Recentemente, várias propostas têm surgido com o objetivo de incorporar mecanismos orientados a contratos em linguagens. Por exemplo, a linguagem Eiffel [89] com seu suporte para *design by contract*, iContract for Java [92], Sather [93] e OCL (*Object Constraint Language*) [94] que é uma proposta de incorporação de contratos no âmbito da linguagem UML.

Dependências de Contexto

Segundo Szyperski [15], a especificação das dependências de contexto do componente é tão importante quanto a especificação das suas interfaces. Além da especificação das interfaces proporcionadas, um componente também deve especificar as suas necessidades (interfaces requeridas). Estas necessidades são denominadas dependências de contexto. Um componente pode ter dependências de contexto explicitadas em termos das interfaces de outros componentes (requeridas durante a sua execução), de algum sistema operacional específico, de outro elemento de *software*, ou mesmo necessitar de recursos como um computador com uma velocidade de processamento específica (para alcançar desempenho pré-determinado) ou ainda de dispositivos de *hardware* específicos (como câmeras e microfones, por exemplo).

Especificação de Interfaces X Especificação de Componentes

As especificações de interfaces e de componentes são conceitos separados que executam funções completamente distintas. A especificação de interface estabelece o contrato com o cliente do componente. Já a especificação do componente estabelece o contrato com o desenvolvedor ou com a pessoa que realiza os testes do componente.

Segundo Cheesman e Daniels [83], as principais diferenças são:

- a *especificação de interface* contém uma lista de operações; a *especificação do componente* contém uma lista de interfaces suportadas pelo componente;

- a *especificação de interface* representa o contrato com o cliente; a *especificação do componente* representa o contrato com o implementador do componente;

- a *especificação de interface* especifica como as operações afetam o estado do objeto que implementa esta interface e as restrições para invocar estas operações; a *especificação do componente* define a unidade de implementação e de execução do componente;

- a *especificação de interface* descreve somente os efeitos locais de suas operações; a *especificação do componente* especifica as interfaces requeridas de outros componentes.

2.2.2 Esquema de Nomes Padronizado

O modelo de componentes tem que fornecer um esquema de nomes padronizado de forma a identificar componentes e interfaces de forma homogênea. As duas principais abordagens são:

- *Identificadores Únicos* são gerados por ferramentas dedicadas, tais como compiladores, que garantem que este identificador é único. Um exemplo destes identificadores são UUIDs (*Universally Unique IDs*) do DCE (*Distributed Computing Environment*);

- *Espaços de Nomes Hierárquicos* têm garantia de serem únicos se a mais alta hierarquia for registrada em uma autoridade de nome global. A maioria dos modelos de componentes baseados em Java usam esta abordagem, embora não exista, até o momento, nenhuma autoridade de registro global para componentes.

2.2.3 Padrões para Metadados

Um modelo de componentes deve especificar de que forma um metadado — informações sobre interfaces, componentes e suas relações — é descrito e como pode ser obtido. A implementação do modelo de componentes precisa proporcionar alguns serviços que permitam a recuperação do metadado. Um metadado é proporcionado nos sistemas baseados em CORBA por repositórios de interface e de implementação; em sistemas baseados em Java, por introspecção; e, em sistemas baseados em COM, por repositórios de tipos COM.

2.2.4 Padrões para Interoperabilidade

A composição de componentes somente é possível se um modelo de componentes fornecer padrões para a interação entre componentes de diferentes fornecedores e que foram implementados, eventualmente, em diferentes linguagens de programação.

Um modelo de componentes pode suportar a interação entre componentes distribuídos. A interoperabilidade é baseada em chamadas de métodos remotos (RMCs: *Remote Method Calls*), que é equivalente ao conceito de RPCs. Em sistemas baseados em CORBA, este suporte é fornecido por meio de *stubs* e *skeletons*. Adicionalmente, para o suporte a componentes distribuídos, o modelo estabelece a definição de representações de dados comuns. Um modelo de componentes também impõe os protocolos de rede que serão utilizados para a comunicação de componentes baseados no mesmo modelo de componentes. Alguns exemplos de especificações de chamada de métodos remotos são: IIOP para sistemas baseados em CORBA; RMI para plataformas baseadas em Java; e SOAP para sistemas baseados em XML.

Um problema de interoperabilidade ocorre na interação entre componentes que foram implementados em modelos de componentes distintos e que suportam especificações incompatíveis de método remoto. Um modelo de componentes pode definir como viabilizar a interação entre implementações de diferentes modelos de componentes. Por exemplo, a especificação CORBA descreve pontes (*bridges*) que permitem acessar, a partir de ambientes CORBA, objetos COM da Microsoft e vice-versa.

2.2.5 Padrões para Especialização

O modelo de componentes provê a base para a especialização ao fornecer os padrões de interoperabilidade e de metadados para componentes e interfaces.

Heineman e Councill [11] definem *especialização de componentes* como a habilidade que um projetista de *software* tem, antes da fase de instalação ou dinamicamente, de adaptar um componente às suas necessidades.

Como os componentes são geralmente tratados como “caixa preta”, o modelo precisa especificar interfaces que suportam a especialização de componentes. Estas interfaces de especialização permitem que ferramentas de instalação e de especialização modifiquem algumas propriedades simples, ou mesmo um comportamento mais complexo, proporcionando instâncias de um determinado componente específicas às necessidades de uma aplicação. Além disso, as ferramentas de especialização, utilizando serviços de metadados, podem ser informadas sobre as interfaces de especialização definidas pelo componente.

2.2.6 Padrões para Composição

O termo composição, utilizado neste trabalho e na literatura de tecnologias de componentes, refere-se à descrição de como os sistemas são montados. Heineman e Councill [11] definem *composição de componentes* como a combinação de dois ou mais componentes que produzem um novo comportamento a partir dos componentes envolvidos. Um padrão de composição de componentes define interfaces e critérios para possibilitar a composição de componentes criando uma estrutura maior e permitindo a inserção ou a substituição de componentes, dentro da infra-estrutura de componentes existente (*frameworks* de componentes).

Em termos de composição, enfatizamos que um componente proporciona funcionalidades que combinadas com as funcionalidades supridas por outros componentes agregados a ele formam uma parte, ou uma aplicação completa. Além disso, um componente é criado considerando-se o seu reuso e, portanto, pode ser utilizado por diferentes aplicações. Segundo Heineman e Councill:

> Um *padrão de composição* define os requisitos permitindo que elementos de *software* sejam compostos para criar elementos de *software* maiores. Adicionalmente, o fornecedor do componente deve suprir uma documentação ou especificação descritiva de modo a permitir que um projetista possa utilizar o componente em uma aplicação alvo.

Bachman [13] enfatiza que os componentes são compostos de tal modo que eles possam interagir. O modelo de componentes precisa especificar padrões para os tipos de componentes e para os seus padrões de interação definindo como os componentes interagem entre si, e como eles interagem com o *framework* de componentes. Quanto à primeira forma de interação, Heineman e Councill esclarecem que, tipicamente, os componentes interagem entre si utilizando o *framework* de componentes por meio de invocações de métodos. O modelo de componentes precisa definir interfaces para prover o suporte a esta composição de forma uniforme. Já a interação com o *framework* de componentes inclui aspectos de gerenciamento de recursos, tais como ciclo de vida do componente (ativação e desativação) e os aspectos não-funcionais que o modelo fornece. De forma a suportar estas formas de interação, o modelo de componentes precisa definir contratos de realização.

Existem várias abordagens para a composição de componentes, em diferentes níveis de abstração. Os componentes podem ser conectados utilizando linguagens de programação de propósito geral (C++, Java); linguagens de *scripting* ou *glue* (JavaScript, TCL e VisualBasic); ferramentas de programação visual ou de composição; ou infra-estruturas de componentes. As linguagens e ferramentas de composição apresentam a desvantagem de que o *glue code* tem que ser completamente escrito ou graficamente especificado. Diferentemente, é alcançado o máximo reuso com *frameworks* de componentes projetados para domínios específicos, onde a interação entre as instâncias dos componentes

é pré-definida. A composição, neste contexto, é uma questão de fazer a inserção ou substituição de componentes que se sujeitam aos padrões de interação definidos pelo modelo de componentes [11].

2.2.7 Padrões para Suporte a Evolução

A evolução em um sistema altamente componentizado e bem projetado permite que mudanças (substituição de um componente por outro que contenha uma implementação com novas funcionalidades ou atualização de novas versões de uma mesma implementação) em um componente sejam realizadas facilmente e com pouco ou nenhum efeito sobre os demais componentes [95].

Cheesman e Daniels [83] afirmam que o gerenciamento de mudanças de componentes tem sido considerado como um importante desafio da abordagem orientada a componentes. A ênfase é na arquitetura do sistema, que tem que ser capaz de gerenciar todo o sistema quando ocorre uma evolução ou mudança de requisitos dos seus componentes. Uma nova versão de um componente pode atualizar a sua implementação e novas interfaces. Desta forma, as dependências entre componentes ficam restritas a interfaces individuais. Isto reduz consideravelmente o impacto quanto a mudanças, porque um determinado componente pode ser substituído por um outro com uma especificação diferente, desde que esta nova especificação inclua as mesmas interfaces ou um subconjunto igual ao que o componente substituído fornecia.

Heineman e Councill [11] expõem as mesmas necessidades apresentadas acima e afirmam que um modelo de componentes deve estabelecer regras e padrões para a evolução de componentes.

2.2.8 Padrões para Empacotamento e Distribuição

Um modelo de componentes precisa descrever a forma como os componentes são empacotados e distribuídos, tal que possam ser instalados independentemente [11]. O processo de distribuição utiliza componentes empacotados em arquivos de formatos tais como ZIP, JAR (*Java Archive*) ou DLL (*Dynamic Linking Library*), contendo as implementações e informações necessárias para instalar e personalizar as implementações de componentes, criar instâncias de componentes e interconectá-las de modo a proporcionar aplicações na forma executável. É importante enfatizar que todos os recursos necessários para a execução de um componente, incluindo a especificação das dependências de contexto, precisam ser instalados junto com o componente. Deste modo, o componente executará apropriadamente as suas funções somente quando tais recursos forem disponibilizados ao componente.

Um componente é distribuído em uma plataforma de componentes. No modelo de componentes, o padrão de distribuição especifica a estrutura e as semânticas para as descrições de distribuição e pode definir, também, o formato dos pacotes. Uma descrição de distribuição deve fornecer infor-

mações necessárias para o processo de distribuição, tais como os recursos necessários na máquina em que os componentes serão instalados, outros componentes, bem como os aspectos de configuração e especialização. Esta descrição é analisada pela infra-estrutura de componentes da máquina destino para a apropriada instalação e configuração dos componentes.

Um componente para ser instalado independentemente precisa ser completamente separado de seu ambiente de execução e de outros componentes. Conseqüentemente, o componente encapsula a sua implementação, ou seja, os dados e algoritmos imprescindíveis para executar as suas tarefas. O processo de distribuição tipicamente envolve três fases: instalação, instanciação e configuração de componentes.

2.2.9 Implementação do Modelo de Componentes

A implementação de um modelo de componentes consiste em proporcionar plataformas que forneçam o suporte a uma infra-estrutura de componentes aderentes a este modelo. Esta infra-estrutura, denominada *framework* de componentes, requer um ambiente de execução que disponibilize os serviços necessários à interação e composição de componentes.

Frameworks de Componentes

Além do reuso de componentes individuais, os *frameworks* de componentes também permitem o reuso do projeto como um todo. Szyperski [15] define *framework* de componentes como uma arquitetura com um conjunto de interfaces[1] e regras que governam a interação entre componentes. Um *framework* de componentes suporta um conjunto de componentes a ele conectados, impondo (e policiando) a estes componentes regras de interação. Portanto, *frameworks* de componentes estabelecem uma infra-estrutura de suporte que viabiliza a composição e instalação de componentes.

Segundo Bachman [13], os *frameworks* de componentes quando instalados no contêiner fornecem elementos que permitem a gerência do ciclo de vida e de outros recursos compartilhados pelos componentes, como por exemplo iniciar, suspender, reiniciar ou terminar a execução de componentes. No entanto, os *frameworks* de componentes são especializados para suportar somente uma limitada variação de tipos de componentes e as interações entre estes tipos. Uma outra característica é que os *frameworks* de componentes não têm uma existência independente dos componentes em tempo de execução, isto é, a implementação do *framework* de componente contém todos os elementos (implementação, gerência e recursos) necessários para a posterior instanciação e execução do componente.

Muitos exemplos de *frameworks* de componentes podem ser encontrados no mercado. A especificação EJB define um *framework* de servidores e contêineres para suportar o modelo de componentes

[1] Especificadas por contratos.

EJB, onde os servidores têm a responsabilidade de proporcionar os aspectos não-funcionais enquanto os contêineres gerenciam o ciclo de vida do componente.

Ambiente de Execução

A padronização do ambiente de execução, ou contêiner, para o suporte à execução de componentes é uma importante parte de um modelo de componentes [11]. Esta padronização consiste da especificação das interfaces para os serviços comuns e específicos de um determinado domínio de aplicação, tais como os serviços horizontais (ciclo de vida de componentes, persistência e segurança), e os serviços verticais (específicos de um domínio, como comunicação multimídia). Alguns destes serviços são proporcionados pelas plataformas de *middleware*.

2.3 Tecnologias Orientadas a Componentes

Nesta seção é apresentada uma visão geral das duas principais tecnologias orientadas a componentes não proprietárias: CCM e EJB. Embora estas tecnologias tenham algumas diferenças significantes, Szyperski e outros autores [15][95][11] afirmam que elas são construídas sobre os mesmos conceitos básicos do paradigma de componentes, com uma estrutura de componentes similar e vários aspectos em comum.

2.3.1 Modelo de Componentes CORBA

O Modelo de Componentes CORBA (CCM) [17] é parte da especificação CORBA 3 [96] e representa o esforço desenvolvido pelo OMG para minimizar as limitações do modelo de objetos CORBA e atender aos novos requisitos, impostos pela evolução da computação de objetos distribuídos para a computação de componentes distribuídos. Motivado pela necessidade de redução da complexidade durante as fases de desenvolvimento e instalação de aplicações orientadas a componentes e pelo sucesso do EJB, CCM proporciona um modelo de programação de mais alto nível de abstração, baseado no conceito de componentes como elementos de *software* com alto potencial de reutilização [15]. CCM é um padrão para a implementação do lado servidor de aplicações distribuídas, compostas de componentes distribuídos e heterogêneos, podendo ser utilizado também no lado cliente destas aplicações.

CCM especifica um modelo de componentes que define os padrões utilizados durante o ciclo de desenvolvimento de componentes — incluindo um modelo de suporte ao ambiente de execução — que utiliza a plataforma CORBA tradicional. CCM é um modelo neutro, ou seja, foi especificado considerando interoperabilidade. Portanto, os componentes de uma aplicação desenvolvidos segundo

o modelo CCM podem ser implementados utilizando-se diferentes linguagens de programação e sistemas operacionais.

Componentes CCM

Um componente CCM é uma extensão de um objeto CORBA, sendo especificado por meio de uma extensão da IDL do CORBA para a declaração de componentes. Os desenvolvedores de componentes CCM definem as especificações funcionais das interfaces IDL, que posteriormente são codificadas gerando as implementações de componentes. O modelo de componentes CCM define um mapeamento da IDL estendida para a IDL padrão, e é a partir desta IDL que o desenvolvedor codificará a parte funcional do componente.

Um componente CCM proporciona mecanismos denominados portas, atributos e referência de componente. Um componente CCM tem uma única referência de componente e um número arbitrário de portas e de atributos. A Figura 2.1 apresenta uma representação de um componente CCM. Nesta figura:

- *portas*: são mecanismos através dos quais clientes, componentes cooperantes e outros elementos do ambiente da aplicação podem interagir com este componente;
- *atributos*: são propriedades, normalmente utilizadas para configuração de componentes;
- *interface equivalente*: é uma interface que identifica unicamente uma instância de um componente e permite o acesso às portas do componente.

CCM define quatro tipos de portas:

- *facetas*, também conhecidas como *interfaces proporcionadas*, são as interfaces que um componente provê para interação com os seus clientes;
- *receptáculos* são os pontos de conexão que armazenam as interfaces requeridas pelo componente;
- *produtores de eventos* são os pontos de conexão que emitem eventos, de um dado tipo, para um ou mais consumidores de eventos do mesmo tipo;
- *consumidores de eventos* são os pontos de conexão capazes de consumir eventos de um dado tipo.

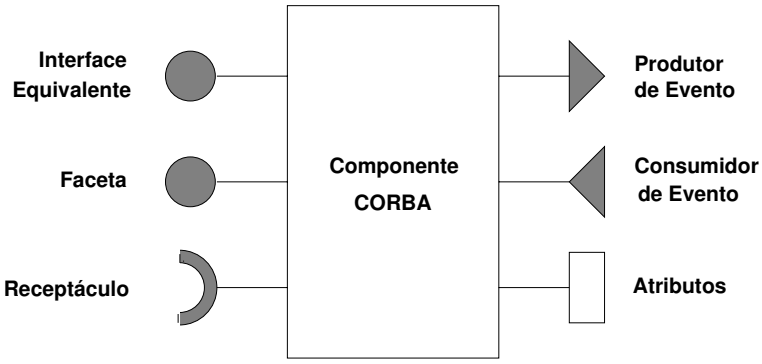

Fig. 2.1: Componente CCM.

Uma instância de um componente é identificada, principalmente, pela sua interface equivalente e, alternativamente, pelo seu conjunto de referências de facetas. O modelo CCM proporciona operações para a navegação entre as interfaces de um componente onde, a partir de uma interface conhecida de um componente (o que inclui a interface equivalente), um cliente pode obter as referências para as suas demais interfaces.

De forma a padronizar a interface de gerenciamento do ciclo de vida de um componente, CCM proporciona o elemento *home*. Este elemento gerencia o conjunto de todas as instâncias de um tipo específico de componente. As interfaces *home* proporcionam operações para gerenciar o ciclo de vida (criar, procurar e remover) de instâncias de um componente. Assim, para utilizar um componente, um cliente precisa primeiramente adquirir uma referência para a interface *home* do componente. O modelo proporciona, para os clientes, a interface *home finder* para a obtenção da referência para a interface *home* de um componente. Depois que o cliente obtém esta referência, ele pode criar ou encontrar a referência do componente desejado.

Interação entre Componentes CCM

Os componentes desenvolvidos segundo o modelo CCM interagem com entidades externas através de portas. Em adição, por meio de atributos, CCM permite que mecanismos de configuração modifiquem as configurações de um componente.

O mecanismo de interação proporcionado pelo modelo CCM é baseado em chamada de métodos remotos, portanto obedecendo ao estilo de interação operacional proposto pelo RM-ODP. CCM fornece também a interface de sinal, representada pelo modelo de comunicação publicador/subscritor, segundo o estilo de interação sinal do RM-ODP. Portanto, o modelo proporciona dois modos de interação: invocações síncronas através de facetas e notificações assíncronas através de pontos de conexão produtores e consumidores de eventos. Em adição, um componente pode definir as inter-

faces que ele necessita por meio de receptáculos que armazenam referências para as portas de outros componentes. Entretanto, este modelo não define a interface *stream*, que é extremamente importante para o desenvolvimento de aplicações multimídia distribuídas.

O Framework de Implementação de Componentes CCM

O *Framework* de Implementação de Componentes CCM (CIF: *Component Implementation Framework*) define o modelo de programação do CCM para a implementação de componentes. Segundo Marvie e Merle [12], o principal objetivo deste modelo de programação é prover o suporte para a descrição dos aspectos não-funcionais dos componentes e, a partir destes, fornecer a geração automática desta parte da implementação de forma que o desenvolvedor da aplicação codifique apenas os seus aspectos funcionais. Exemplos de aspectos não-funcionais propiciados pelo modelo de componentes CCM incluem transações, segurança e persistência.

CCM define uma linguagem declarativa, a linguagem de definição de implementação de componentes (CIDL: *Component Implementation Definition Language*), para descrever implementações e o estado persistente de componentes e *homes* de componentes. CIF utiliza as descrições em CIDL para gerar esqueletos de programação distribuída que automatizam uma parte do comportamento básico de componentes, tais como navegação entre facetas, ativação, gerenciamento de *portas*, gerenciamento de ciclo de vida do componente, gerenciamento de estado, etc. Os desenvolvedores de componentes estendem certos métodos providos por estes esqueletos durante a fase de implementação do componente. O compilador CIDL gera os *descritores de componentes* que definem as funcionalidades dos componentes bem como as restrições técnicas que serão garantidas em tempo de instalação, tais como descrições das interfaces dos componentes, políticas de transação, políticas de *threading* e os tipos de serviços que este componente necessita.

A especificação CCM define, de acordo com o modelo de persistência de estado adotado por meio da CIDL, quatro categorias de componentes CORBA: *componentes de serviço*, *componentes de sessão*, *componentes de processo* e *componentes entidade*. Um componente de serviço contém as propriedades de suporte a uma única invocação por instância. Estes componentes não possuem estado nem identidade (equivalentes aos *beans* de sessão sem estado do EJB). Um componente de sessão contém as propriedades de suporte a uma seqüência de invocações por instância. Estes componentes possuem estado transiente e identidade não persistente (equivalentes aos *beans* de sessão com estado do EJB). Um componente de processo contém as propriedades de comportamento, que podem ser transacionais. Estes componentes possuem estado persistente, comportamento transacional, são gerenciados pelo ambiente e possuem identidade persistente gerenciada pelo cliente. Um componente entidade é similar ao anterior e contém as propriedades de comportamento, que podem ser transacionais. Estes componentes possuem estado persistente e identidade visível para o cliente

(equivalentes aos *beans* de entidade do EJB).

Modelo de Programação de Contêineres CCM

Um contêiner representa o ambiente de execução para uma implementação de componente CORBA. Um contêiner provê o encapsulamento para as instâncias de um único tipo de componente. Para tal, o contêiner utiliza o adaptador portável de objeto (POA: *Portable Object Adapter*) especializado para este componente, bem como de um conjunto de interfaces para o acesso transparente a serviços CORBA necessários ao componente, tais como persistência, transação, segurança e notificação de eventos. Adicionalmente, ele também provê recursos de baixo nível, tais como memória, processador, entre outras dependências que o componente hospedado necessite. As implementações de componentes dependem do POA para interceptar e enviar as requisições dos clientes para os seus objetos servidores correspondentes. A partir da descrição do componente, a implementação do modelo de componentes CCM gera automaticamente a criação e configuração da hierarquia do POA, bem como localiza os serviços comuns definidos pelo modelo.

Um modelo de programação de contêiner define um conjunto de interfaces para facilitar o desenvolvimento de aplicações CORBA. O contêiner integra os serviços CORBA ao comportamento funcional de componentes tendo como base os descritores de componentes apresentados na seqüência. A Figura 2.2 ilustra o modelo de programação do contêiner CCM.

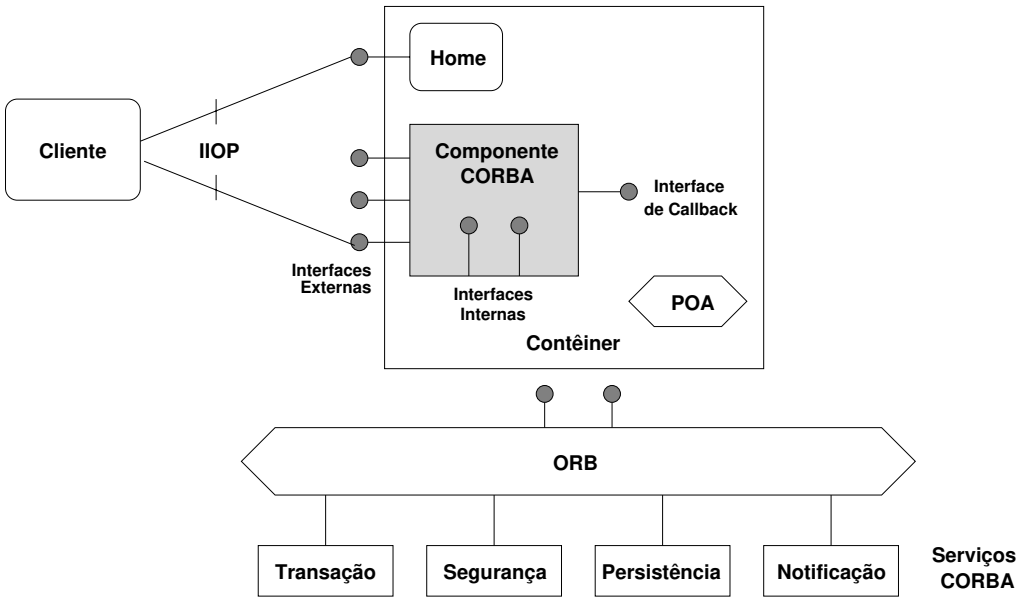

Fig. 2.2: Contêiner para componentes CORBA.

O modelo de programação de um contêiner define:

- *interfaces externas*, interfaces disponíveis para os clientes do componente hospedado. Estas interfaces estabelecem o contrato entre o desenvolvedor do componente e o cliente deste componente;
- *interfaces do contêiner*, estabelecem o contrato entre um componente e o seu contêiner por meio de dois tipos de interfaces:
  1. *interfaces internas*, definidas como as interfaces locais que são disponibilizadas para que o componente hospedado possa acessar os serviços proporcionados pelo contêiner;
  2. *interfaces callbacks*, também interfaces locais, implementadas pelo componente, que podem ser invocadas pelo contêiner para gerenciar e notificar o componente.
- *modelo de uso*, que define as interações entre o contêiner e a plataforma CORBA (POA, ORB e serviços CORBA);
- funcionalidades para a navegação entre as interfaces do componente hospedado, por meio da interface *Home*;
- conexão das facetas e receptáculos do componente;
- canais de eventos para transmitir e para receber eventos entre os componentes.

Empacotamento e Distribuição de Componentes CCM

Os componentes CCM podem ser distribuídos por meio de vários servidores e geralmente utilizam diferentes linguagens de programação, sistemas operacionais e compiladores. Em adição, estes componentes podem depender de outros componentes. Conseqüentemente, o empacotamento e a distribuição destes componentes podem tornar-se muito complexos. Com o intuito de simplificar estas etapas, a especificação do modelo de componentes CCM define técnicas que fornecem o padrão para o empacotamento e distribuição. Estas técnicas são resumidas a seguir.

A especificação CCM define a forma como os componentes e suas implementações são empacotados para, posteriormente, serem instalados em máquinas dispostas na rede. Um pacote de componentes consiste de um ou mais descritores e de um conjunto de arquivos contendo a implementação de um único componente. Estes descritores são utilizados para descrever os componentes e as suas dependências, e são escritos na linguagem OSD (*Open Software Description*). OSD é um vocabulário XML, ou seja, é um DTD (*Document Type Definition*) definido pelo consórcio W3C. Os componentes são empacotados em arquivos no formato ZIP.

O pacote de componente pode ser distribuído sozinho ou pode ser incluído em um pacote de montagem de componente e distribuído junto com outros componentes. Uma montagem especifica

uma configuração inicial. Este pacote de montagem consiste de um conjunto de pacotes de componentes (incluindo os *homes* dos componentes) interrelacionados, de um arquivo de propriedades e de um descritor de montagem. O descritor de montagem é também especificado utilizando um vocabulário XML contendo os componentes, as conexões entre eles e as informações de partição. Estas informações especificam um padrão de distribuição em que grupos de componentes devem ser distribuídos dentro de uma máquina ou processo específico, ou especificam se eles podem ser distribuídos livremente. O arquivo de propriedades define os parâmetros de configuração (atributos) de componentes e de seus *homes*.

A especificação CCM define um processo de distribuição que utiliza os pacotes de componentes de forma a instalar as implementações dos componentes, criar instâncias de componentes e interconectá-las de maneira a proporcionar uma aplicação executável.

Em adição aos descritores apresentados acima, o modelo especifica o descritor de componente, também por meio de um vocabulário XML. O compilador CIDL gera este descritor que lista as restrições técnicas a serem garantidas em tempo de instalação. Este descritor especifica as características do componente utilizadas em tempo de projeto e de instalação. Estas características incluem uma descrição da estrutura do componente em termos de interfaces suportadas, componentes herdados, eventos e portas proporcionadas e necessitadas, além dos aspectos não-funcionais de transação, segurança e persistência. Adicionalmente, em tempo de instalação, o descritor de componente é utilizado para determinar o tipo de contêiner em que o componente precisa ser instalado, bem como para fornecer informações do componente para o contêiner.

Considerações Gerais Sobre o Modelo CCM

O modelo CCM tem dois predecessores: o modelo de objetos CORBA e o modelo de componentes EJB. CCM foi modelado de acordo com a especificação EJB, ou seja, CCM integrou em sua especificação o bem sucedido modelo de componentes do EJB, adotando-o no mundo CORBA e mantendo a interoperabilidade e a neutralidade no nível de linguagem do CORBA. CCM é portanto compatível com EJB e define um mapeamento padrão entre os dois modelos de componentes. Desta forma, é facil a interoperação entre os componentes CCM e EJB. Em adição, o *framework* de comunicação do EJB suporta o protocolo IIOP da especificação CORBA. Assim, um componente CCM pode ser considerado como um componente EJB para clientes EJB e portanto pode ser instalado em um contêiner EJB, respeitando a restrição imposta pelo EJB, que exige o uso da linguagem Java como a linguagem de implementação para os componentes aderentes a este modelo. Da mesma forma, um componente EJB pode ser considerado como um componente CCM para clientes CCM e, assim, podem ser instalados em um contêiner CCM. Esta integração permite que uma aplicação seja montada utilizando uma combinação de componentes CCM e EJB.

O modelo CCM pode ser visto como um modelo adequado para o desenvolvimento de aplicações distribuídas e escaláveis. No entanto, algumas considerações são pertinentes.

Uma primeira consideração é que a especificação CCM é muito extensa, complexa e só foi disponibilizada recentemente, o que significa que as implementações do modelo e ferramentas de suporte ainda estão sendo desenvolvidas. A primeira implementação CCM, denominada OpenCCM, foi publicada recentemente com apenas uma parte do modelo. Portanto, é difícil avaliar a qualidade e performance do modelo, apesar da sua crescente aceitação como um padrão de componentes. Entretanto, devido a sua compatibilização com o modelo de componentes EJB, cuja aceitação é ampla, comprova que pelo menos esta parte do CCM pode ser considerada avaliada. As outras partes que estendem o EJB, no entanto, ainda precisam ser melhor analisadas e efetivadas na prática.

Uma segunda consideração enfatiza que o modelo CCM não é adequado às aplicações que manipulam mídias contínuas, porque não proporciona suporte à interface *stream* proposta pelo modelo de referência RM-ODP. A incorporação desta interface ao CCM foi abordada na Seção 1.4.

Finalmente, a definição estática de contêineres impede a flexibilidade das aplicações emergentes quanto a novos requisitos, por exemplo, a adequação a aspectos de qualidade de serviço impostos por novos ambientes com recursos limitados. Seria mais interessante poder utilizar contêineres adaptáveis, que permitissem a adição (estática ou dinâmica) de novos aspectos não-funcionais a serem gerenciados pelo contêiner.

2.3.2 Modelo de Componentes Enterprise Java Beans

O modelo de componentes *Enterprise Java Beans* (EJB) [18] é um modelo de componentes para o desenvolvimento e instalação de aplicações na linguagem de programação Java, pertencentes ao domínio de negócios. EJB é um modelo de componentes específico para a linguagem Java no contexto da plataforma *Java 2 Enterprise Edition* (J2EE) [79]. O modelo EJB consiste de uma especificação aberta fornecida pela Sun Microsystems a partir de 1998. Esta especificação define um modelo de componentes para o lado servidor.

Segundo a especificação, um componente EJB é denominado "*enterprise bean*". Estes componentes, localizados no lado do servidor, podem ser definidos como unidades de *software* reutilizáveis que contêm a lógica da aplicação. Os aspectos não-funcionais também são providos automaticamente pelo contêiner.

A especificação define três tipos de componentes: sessão, entidade e orientado a mensagem. O componente de sessão é um componente transiente que existe durante uma única sessão cliente/servidor. O componente entidade é persistente, com identidade única. Este componente é utilizado para modelar entidades de dados permanentes, mantidos em bancos de dados. O componente orientado a mensagem é utilizado quando a invocação assíncrona entre componentes é necessária. Este compo-

nente é um consumidor de mensagens de notificação assíncronas que são enviadas pelos clientes. Este componente não tem nenhuma identidade ou visibilidade para o cliente, é transiente e sem estado.

A implementação do modelo de componentes EJB é dividida em duas partes: o servidor EJB e o contêiner. Um componente EJB é instalado e executado em um contêiner. O contêiner gerencia o ciclo de vida do componente e é a parte central da implementação do modelo de componentes EJB e, juntamente com o servidor EJB, fornece todos os recursos necessários para que um componente distribuído possa interagir com outros componentes.

Um servidor EJB provê o ambiente de execução para um ou mais contêineres. Ele é o responsável pela gerência de recursos de baixo nível do sistema e em prover os aspectos não-funcionais alocando-os ao contêiner quando necessário.

Um contêiner provê o encapsulamento de um ou mais componentes de uma aplicação. A Figura 2.3 ilustra o contêiner no ambiente EJB. O contêiner consiste das seguintes partes:

- *Contrato de componente*: um conjunto de interfaces especificadas pelo contêiner e que os componentes devem implementar ou estender para que ele possa gerenciá-las. Exemplos são mecanismos de *callback* e interfaces com operações para criar, encontrar e destruir componentes.

- *Serviços declarativos*: são serviços que o contêiner interpõe entre clientes e componentes conforme especificado no descritor de instalação de cada componente. Semelhante ao CCM, a infra-estrutura provida pelo contêiner, em conjunto com o servidor EJB, facilita o desenvolvimento de aplicações agregando os aspectos não-funcionais a um componente tais como segurança, gerência de ciclo de vida, de estado, de transação e de persistência. O desenvolvedor do componente EJB, como no CCM, concentra-se apenas nos aspectos funcionais do componente.

- *Interfaces de serviços do contêiner*: interfaces para os serviços Java externos ao contêiner. Exemplos são as interfaces padrão da plataforma J2EE, tais como JDBC (para acesso a bancos de dados), JTA[2] (para o suporte a transações distribuídas) e JNDI[3] (provê uma forma padronizada para o acesso a diferentes serviços de nomes e de diretório distribuído).

O cliente da aplicação interage com o componente EJB através de duas interfaces geradas pelo contêiner: as interfaces *Home* e *Remote*. Quando um cliente faz uma requisição para alguma operação destas interfaces, o contêiner as intercepta, personalizando ou adicionando serviços de gerência (propriedades não-funcionais), antes de repassá-las para os componentes destino.

[2] Java Transaction API.
[3] Java Naming and Directory Interface.

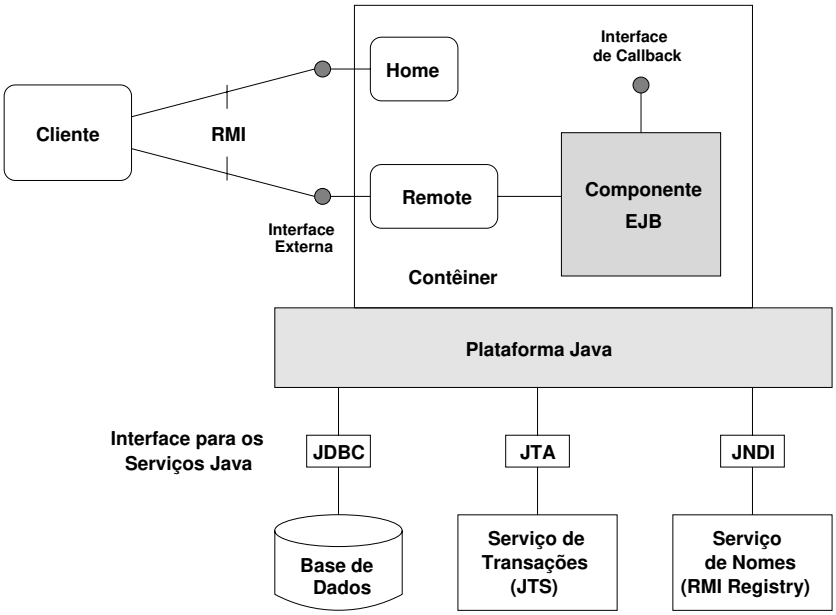

Fig. 2.3: Contêiner no ambiente EJB.

A interface *Home* atua como uma fábrica em que clientes podem criar, encontrar ou remover componentes EJB. A interface *Remote* provê, para os clientes, o acesso aos métodos da lógica da aplicação do componente EJB.

O mecanismo de interação proporcionado pelo modelo EJB é baseado em chamada de métodos remotos, portanto obedecendo ao estilo de interação operacional proposto pelo RM-ODP. Diferentemente do modelo CCM, EJB não fornece a interface de sinal e, semelhantemente ao CCM, também não fornece a interface *stream*.

Empacotamento e Distribuição de Componentes EJB

Os componentes EJB podem ser empacotados como componentes individuais, como uma coleção de componentes ou mesmo como uma aplicação completa. Estes componentes são distribuídos em um arquivo denominado *ejb-jar* no formato JAR. O arquivo *ejb-jar* contém o descritor de instalação, as classes do componente e as interfaces *Home* e *Remote*.

A especificação EJB descreve um mecanismo declarativo, o descritor de distribuição, que é um arquivo XML que contém informações e propriedades de configuração, tais como: informações sobre as classes, atributos transacionais, papéis de segurança e configurações que ditam o comportamento do componente.

Quando um componente é instalado em um contêiner, o contêiner lê o descritor de instalação onde estão especificados os atributos de transação, de segurança, de persistência (manipulada automatica-

mente ou predeterminada pelo componente) e de controle de acesso. A partir destas informações, o contêiner provê automaticamente estes aspectos não-funcionais. O componente é disponibilizado para os seus clientes ou para outros componentes na finalização desta etapa de instalação.

Considerações Gerais do Modelo EJB

Os componentes EJB são, genericamente, similares a componentes CCM. De fato, a especificação CCM pode ser vista como uma extensão do modelo de componentes EJB. No entanto, a especificação CCM define o modelo de componentes acima do modelo de objetos distribuídos CORBA, enquanto que a especificação EJB define que o modelo de componentes EJB é suportado pelo modelo de objetos de Java. Assim, os componentes segundo o modelo EJB são dependentes de linguagem de programação.

2.3.3 Comparação Entre os Modelos EJB e CCM

Nesta seção faz-se uma breve comparação entre os modelos de componentes EJB e CCM, segundo os oito elementos básicos, descritos na Seção 2.2, que caracterizam um modelo de componentes.

Padrão para Especificação do Comportamento de Componentes

Em termos da parte sintática dos aspectos funcionais, a especificação de um componente EJB é provida pela própria linguagem Java para a definição das interfaces do componente, interface *Home* e de *callback*. A especificação de um componente CCM é provida utilizando uma extensão da linguagem IDL (IDL3) para a definição das mesmas interfaces. As ferramentas de desenvolvimento que suportam CCM fazem o mapeamento de IDL3 para a IDL convencional (IDL2). Em ambos os modelos estas interfaces derivam de interfaces base que definem operações comuns a todos os componentes (por exemplo, operações de acesso à metainformação associada ao componente). Ambos os modelos não suportam a especificação do comportamento semântico dos aspectos funcionais, por exemplo por meio de pré e pós-condições.

Em termos dos aspectos não-funcionais, a especificação dos modelos CCM e EJB suportam, por meio de descritores de instalação, os serviços de segurança, transações e persistência.

Padrão para Nomes

EJB utiliza JNDI para registro e acesso a componentes. JNDI necessita um serviço de nomes externos, por exemplo o serviço de nomes do RMI (*rmiregistry*) ou o serviço de nomes do CORBA (*CosNaming*).

CCM utiliza o serviço de nomes do CORBA para registro e acesso a componentes. Tanto JNDI quanto *CosNaming* permitem associar nomes a contextos, bem como organizar contextos em hierarquias de contextos.

Padrão para Metainformação

EJB utiliza o mecanismo nativo de introspecção da linguagem Java para o acesso à metainformação associada a seus componentes.

CCM define o conceito de navegação onde, a partir da interface equivalente do componente, é possível acessar suas demais interfaces. Informações adicionais sobre as interfaces (métodos, exceções, etc.) são obtidas do Repositório de Interfaces do CORBA.

Padrão para Interoperabilidade

EJB utiliza RMI como solução de interoperabilidade. RMI pode utilizar seu protocolo de transporte nativo (*Java Remote Method Protocol* - JMRP) ou o protocolo de transporte do CORBA (*Internet Inter-ORB Protocol* - IIOP). JMRP e IIOP operam sobre redes TCP/IP apenas. Por ser uma solução puramente Java, RMI permite a passagem de qualquer objeto Java serializável em invocações de métodos definidos na interface do componente.

CCM utiliza IIOP como solução de interoperabilidade. Por ser uma solução independente de linguagem, apenas os objetos passíveis de definição em IDL podem ser passados em invocações de métodos.

Padrão para Especialização

EJB e CCM adotam o conceito de atributo (ou propriedade) para fins de especialização de componentes. Tipicamente, atributos são configurados no momento de instalação do componente. Componentes CCM definem interfaces IDL para a manipulação de atributos também em tempo de execução.

Padrão para Composição

CCM suporta composição pela interconexão de portas complementares. Estas portas, também definidas em IDL3, adotam um padrão de interação um-para-um (operações tipo *connect/ disconnect*) ou um-para-muitos (operações tipo *publish/subscribe*). Interfaces de notificação e interconexão não estão presentes no modelo EJB.

Padrão para Suporte à Evolução

A especificação do modelo EJB suporta uma única interface por componente, o que significa que a evolução do componente (ao ser substituído por outro, ou atualizado) deve ocorrer pela substituição desta interface ou pela inclusão de novos métodos à interface do componente.

A especificação do modelo CCM suporta múltiplas interfaces por componente possibilitando também evolução do componente pela adição de novas interfaces.

Padrão para Empacotamento e Instalação

Ambos os modelos utilizam arquivos tipo bibliotecas para o empacotamento das classes necessárias à instalação do componente. EJB utiliza o formato nativo da linguagem Java (JAR), enquanto CCM utiliza o formato ZIP por ser independente de linguagem.

Ambos os modelos utilizam descritores de instalação expressos em XML. CCM define quatro arquivos XML: descritor de pacote de software, descritor de componente, descritor de montagem (*assembly*) e descritor de propriedades. EJB define um único arquivo XML, equivalente ao descritor de componente e de propriedades do CCM.

2.4 Considerações Finais

Este capítulo apresentou os principais conceitos que caracterizam o paradigma de componentes de *software* e que são necessários para o entendimento dos demais capítulos.

Os modelos de componentes apresentados neste capítulo consistem, atualmente, dos dois modelos abertos mais difundidos e que são capazes de interoperar entre si e com outros padrões abertos. A separação entre os aspectos funcionais e não-funcionais, apresentados por estes modelos, mostra uma maturidade quanto ao suporte para um rápido desenvolvimento de aplicações com a incorporação de serviços tais como transação, persistência e segurança direcionados para aplicações no contexto de negócios.

Apesar da evolução das tecnologias de *middleware* com a adoção de modelos de componentes, constata-se que os requisitos impostos pelas aplicações emergentes tais como aplicações telemáticas e ubíquas não são atendidos. Tendo como base as deficiências constatadas nos atuais modelos de componentes, propõe-se no próximo capítulo um novo modelo de componentes capaz de atender aos requisitos impostos por tais aplicações.

Capítulo 3

O Modelo de Componentes CM-tel

O surgimento de novas categorias de aplicações centradas na Internet como, por exemplo, aplicações telemáticas e aplicações ubíquas, vem demandando novas tecnologias de desenvolvimento de *software*. Durante os últimos dez anos, vários padrões abertos surgiram com o intuito de facilitar o desenvolvimento de aplicações distribuídas. A evolução mais recente destes padrões foi a incorporação do paradigma de componentes, conforme discutido no Capítulo 2.

No entanto, com base nas avaliações da literatura apresentadas no capítulo anterior e apesar da maturidade dos modelos de componentes constatamos algumas deficiências com relação ao suporte fornecido por estes modelos aos requisitos impostos pelas aplicações consideradas neste trabalho. Entre estas deficiências, destacam-se:

1. Os modelos de componentes são vinculados a uma determinada tecnologia de *middleware*, por exemplo, CCM está vinculado à tecnologia CORBA, EJB à tecnologia Java e COM+ à tecnologia Windows. A utilização destes modelos implica, necessariamente, na utilização das respectivas tecnologias vinculadas.

2. Os modelos de componentes existentes são direcionados para o domínio de aplicações comerciais[1] e carecem de suporte para requisitos impostos por outros domínios, por exemplo, suporte a interfaces de fluxo contínuo (*stream*). Isto torna os modelos de componentes atuais inapropriados para a transmissão de mídias contínuas tempo-real, requisito fundamental para o desenvolvimento de aplicações que utilizam comunicação multimídia distribuída.

3. Os modelos de componentes não oferecem mecanismos que contemplem alguns aspectos não-funcionais importantes, notadamente a qualidade de serviço para fluxos contínuos, um requisito fundamental para as aplicações telemáticas interativas.

[1] Razão pela qual enfatizam o processamento de transações, segurança e a persistência de componentes.

4. Os modelos de componentes não fornecem um processo de instalação flexível para a adaptação dinâmica das aplicações às suas necessidades, requisito fundamental para o suporte às aplicações ubíquas.

5. Os modelos de componentes não permitem a instalação de componentes em dispositivos com pequeno poder computacional tais como computadores de mão (*handhelds*) e telefones celulares.

A falta de suporte aos requisitos citados restringe a utilização dos modelos de componentes existentes, bem como das suas plataformas, em muitas aplicações modernas. As deficiências constatadas e as experiências adquiridas no desenvolvimento de serviços telemáticos no âmbito do projeto do laboratório virtual REAL (*Remotely Accessible Laboratory*) (Capítulo 5), motivaram-nos a desenvolver um novo modelo de componentes de *software* e uma plataforma de suporte a este modelo, com o objetivo de contribuir para o desenvolvimento de novas aplicações disponibilizadas através da Internet, em particular aplicações telemáticas e aplicações que necessitam de requisitos relacionados à mobilidade (de código, de terminal ou de usuário) e ao acesso ubíquo. Este novo modelo, denominado CM-tel (**C***omponent* **M***odel for* **tel***ematic applications*), será apresentado neste capítulo. No Capítulo 4, apresentamos uma plataforma de suporte ao modelo baseada nas tecnologias CORBA e Java.

3.1 Visão Geral do Modelo de Componentes CM-tel

O modelo de componentes CM-tel não visa substituir os modelos de componentes existentes, mas sim complementá-los em algumas de suas principais deficiências. O desenvolvimento do modelo CM-tel foi orientado pelos seguintes princípios:

- o modelo deve ser neutro, ou seja, não vinculado a uma tecnologia específica;
- o modelo deve ser centrado em padrões abertos;
- o modelo deve oferecer mecanismos simples de especificação, instalação, configuração e composição de componentes;
- o modelo deve suportar um conjunto de aspectos funcionais e não-funcionais necessários às aplicações telemáticas e ubíquas, ausentes nos modelos de componentes existentes;
- o modelo deve permitir a utilização de qualquer metodologia de desenvolvimento de *software* baseada na linguagem UML;

- o modelo deve permitir geração automática de código em larga escala.

Dos princípios acima, o mais original é a independência de tecnologia (modelo neutro). A motivação para um modelo neutro é a capacidade de se desenvolver plataformas de suporte para o modelo sobre diferentes tecnologias tais como CORBA, Java/RMI, Java/WebService, RT-CORBA, DCOM, etc. Neste aspecto, a abordagem adotada para o desenvolvimento do modelo CM-tel é diferente da abordagem convencional, onde a tecnologia de suporte dita toda a concepção do modelo de componentes. Nossa proposta consiste em especificar um modelo de componentes em dois níveis de abstração. O primeiro nível define um modelo de componentes genérico, estável e neutro em termos de tecnologia (independente de linguagens de programação, plataformas de *middleware*, plataformas de *hardware*, sistemas operacionais, protocolos de rede, etc.). O segundo nível consiste em mapear este modelo para uma determinada tecnologia de suporte. Um dos benefícios dessa estratégia de dois níveis é a flexibilidade obtida, uma vez que a especificação do modelo de componentes bem como dos componentes e contêineres da aplicação permanecem estáveis, ao passo que a implementação destes em uma tecnologia pode evoluir ou ser substituída de forma independente.

O requisito que impõe mecanismos simples de especificação, instalação, configuração e composição de componentes tem como motivação a construção de plataformas de suporte leves para o modelo. Por exemplo, uma plataforma que implemente o modelo CM-tel pode executar em um dispositivo móvel com baixo poder de processamento, o que seria impossível para plataformas que suportam CCM ou EJB, devido à complexidade destes modelos e de suas plataformas de suporte.

O requisito de suporte a um conjunto de aspectos funcionais e não-funcionais necessários às aplicações telemáticas e ubíquas visa atender aos requisitos impostos pelas aplicações emergentes identificados na Seção 1.2. Neste contexto, CM-tel fornece soluções para estes requisitos como, por exemplo, o suporte para comunicação multimídia distribuída com facilidades de qualidade de serviço (no nível da aplicação, do serviço de gerência de fluxo contínuo e da infra-estrutura de rede), código móvel e um processo de instalação flexível que permite a adaptação dinâmica das aplicações a ambientes com características distintas.

A nossa opção pela linguagem UML deve-se a esta ser a notação padrão para modelagem visual, disponível na maioria das ferramentas de desenvolvimento atuais e amplamente utilizada pela indústria. Além disso, UML proporciona mecanismos de extensão (perfis UML) para o uso de construções em UML, que permitem adaptar a linguagem para aplicações ou tecnologias específicas.

Um modelo de componentes especifica os padrões e convenções impostos por este modelo aos desenvolvedores de componentes, definindo como componentes individuais são construídos, bem como as regras que permitem a interação e composição entre componentes [11]. Para apresentarmos tais padrões e convenções oferecidos pelo modelo CM-tel, tomamos como base as principais características que um modelo de componentes deve fornecer ao desenvolvedor de componentes, conforme

descrito na Seção 2.2. Os elementos básicos independentes de tecnologia, notadamente a composição e interação de componentes, serão descritos neste capítulo. No próximo capítulo serão fornecidas soluções tecnológicas para todos os elementos básicos no âmbito de uma plataforma de suporte ao modelo CM-tel.

Em adição, para descrevermos como componentes e contêineres aderentes ao modelo CM-tel são suportados e construídos, adotamos o conceito de modelagem segundo *níveis semânticos* [83]. Nesta abordagem, o nível semântico de um modelo determina o nível de detalhes (abstração) que o modelo expõe. Por exemplo, modelos segundo três diferentes níveis semânticos são comumente utilizados na modelagem de sistemas de *software* em UML: modelos conceituais, modelos de especificação e modelos de implementação [83].

CM-tel define três modelos de diferentes níveis semânticos, ou perspectivas, para especificar e para apresentar as regras impostas pelo modelo CM-tel para os seus componentes e contêineres: modelo de especificação, modelo de projeto e modelo de implementação física. Estes três modelos foram escolhidos com o intuito de oferecer representações para componentes e contêineres nas fases de análise, projeto e implementação encontrados na maior parte dos processos de desenvolvimento de *software*. A linguagem UML[2] é utilizada por estes três modelos como uma linguagem de modelagem para auxiliar a apresentação do modelo CM-tel. A Figura 3.1 ilustra os três modelos em uma seqüência de etapas que conduzem a especificação, projeto, implementação, empacotamento e instalação do componente e do contêiner CM-tel.

3.2 Modelo de Especificação CM-tel

O modelo de especificação CM-tel, conforme apresentado na Figura 3.1, corresponde à fase de análise dos processos de desenvolvimento de *software*. Este modelo descreve os padrões arquiteturais e convenções definidos pelo modelo de componentes CM-tel para a construção de componentes e de contêineres, bem como para a interação entre componentes em um alto nível de abstração.

3.2.1 Componentes de Software CM-tel

Os padrões arquiteturais e convenções definidos pelo modelo de especificação para componentes CM-tel descrevem a estrutura de um componente CM-tel em termos de suas interfaces, o padrão de interação entre componentes, as especificações dos aspectos funcionais e não-funcionais e a natureza contratual das especificações.

[2]Conforme definido na ferramenta Rational Rose versão 2001.

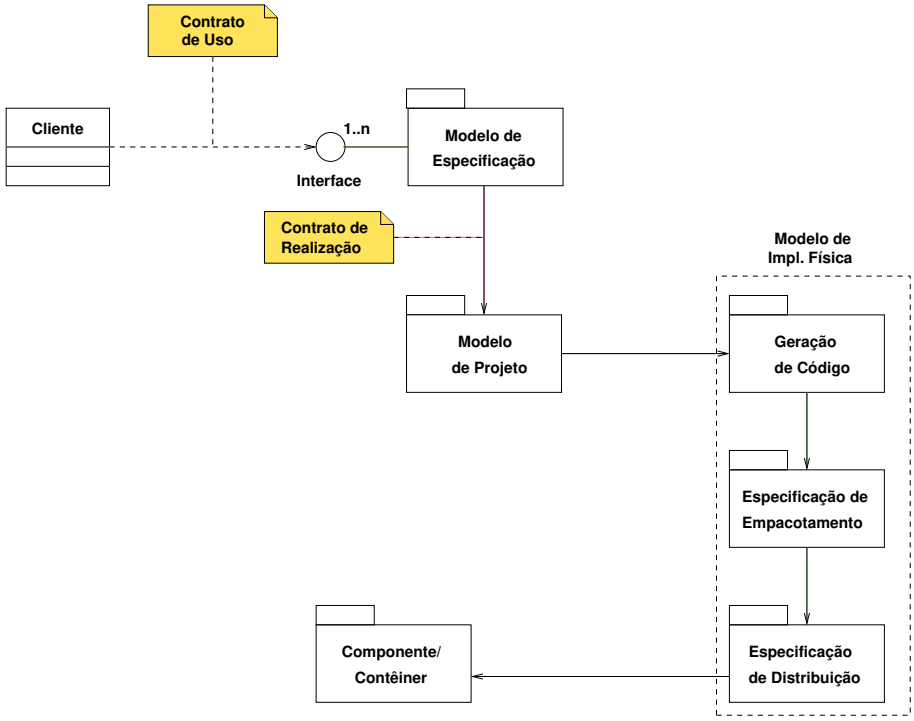

Fig. 3.1: Modelo CM-tel segundo níveis semânticos.

CM-tel mantém as principais estruturas de componentes encontradas em modelos de componentes bem estabelecidos tais como EJB e CCM, avaliados no Capítulo 2. Esta similaridade decorre do fato de que os componentes definem propriedades e métodos, bem como emitem mensagens de notificação. Além de proporcionar eventos, propriedades e métodos, ou seja, elementos da estrutura de um componente que estabelecem a forma de interação entre os componentes, a nossa proposta adiciona interfaces de fluxo contínuo (*stream*), interface esta não encontrada em outros modelos de componentes. Esta interface permite que os componentes manipulem mídia contínua (fluxos de áudio e vídeo tempo-real). Por exemplo, uma sessão multimídia pode ser estabelecida simplesmente conectando um componente consumidor de mídia a um componente produtor de mídia.

Um ponto fundamental em um modelo de componentes é a estrutura do componente. A representação da estrutura de um componente CM-tel é ilustrada na Figura 3.2.

Os componentes CM-tel são entidades de implementação que se sujeitam a um modelo de componentes, podem ser instalados de forma independente e se compor sem modificação segundo um padrão de composição[3]. Para permitir a composição, componentes CM-tel expõem um conjunto de interfaces bem definidas, através das quais os seus clientes podem acessá-los. Sob o ponto de vista do projetista do componente, o componente CM-tel é visto como um conjunto de interfaces. Para de-

[3]Esta definição, proposta por Heineman e Councill [11], foi apresentada na Seção 2.1.2.

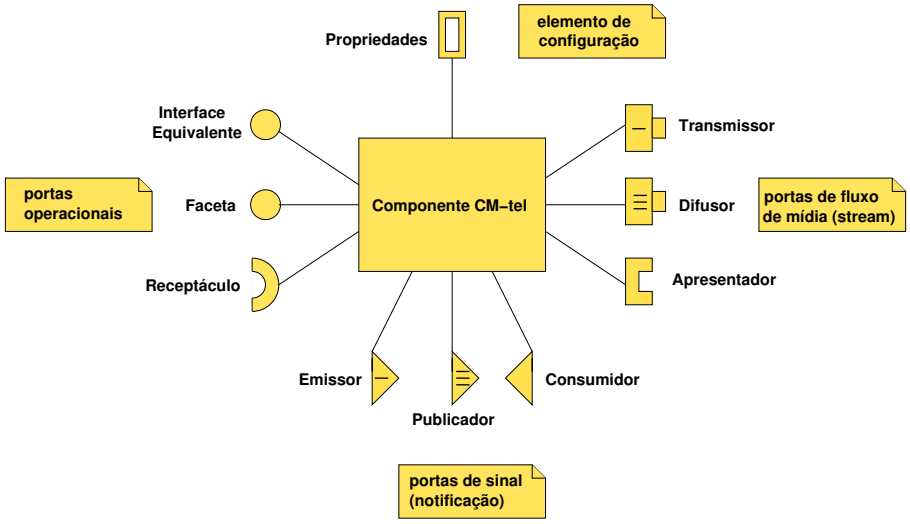

Fig. 3.2: Estrutura de um componente de software CM-tel.

senvolver uma aplicação utilizando componentes, o projetista se concentra apenas nas interfaces que os componentes expõem e na interconexão destas interfaces com interfaces de outros componentes. Deste modo, pode-se reutilizar componentes como blocos de construção. Para a construção da aplicação, os componentes são montados a partir das funcionalidades que eles provêem para formar um sistema completo. As funcionalidades oferecidas pelo componente são acessadas através destas interfaces. A interface do componente é separada da implementação das funcionalidades oferecidas pelo componente. Desta forma, pode-se usar múltiplas implementações de forma a fornecer uma evolução da implementação sem afetar a interface associada e, conseqüentemente, sem afetar outros componentes ou clientes.

O objetivo do nosso modelo é oferecer os três tipos de interfaces propostas pelo RM-ODP [7], ou seja, as interfaces operacional, de sinal e de fluxo contínuo. Estas interfaces correspondem a um contrato entre o cliente e o componente CM-tel.

Os elementos de um componente CM-tel, representados na Figura 3.2, podem ser subdivididos em três categorias:

1. Interface equivalente: interface que identifica unicamente uma instância de um componente e permite o acesso às demais interfaces do componente.

2. Elementos de interconexão ou *portas*: são interfaces operacionais (facetas e receptáculos), de sinal (pontos terminais de eventos: emissor, publicador e consumidor), e de fluxo contínuo (pontos terminais de fluxos: transmissor, difusor e apresentador), que suportam a composição de componentes mediante um processo de conexão de interfaces complementares (ligação ou

binding, no sentido do modelo RM-ODP). A Figura 3.3 ilustra a interação entre componentes através do processo de conexão de portas.

3. Elementos de configuração: são interfaces utilizadas para a configuração de componentes através de propriedades, cujo acesso ocorre sem a necessidade de conexão prévia.

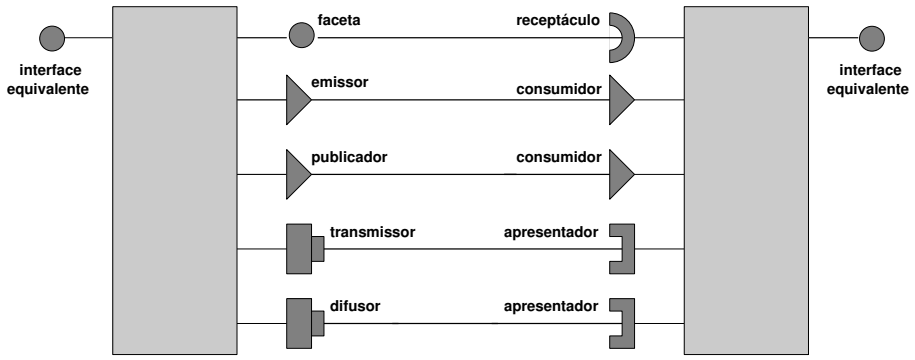

Fig. 3.3: Interação entre componentes CM-tel.

Um componente CM-tel contém uma única interface equivalente e pode conter um número arbitrário (zero ou mais) de portas e de elementos de configuração. A seguir, a interface equivalente, as interfaces dos três elementos da categoria portas e do elemento de configuração de um componente são apresentadas.

Interface Equivalente

Uma interface particular, denominada interface equivalente (também conhecida como a referência base do componente ou interface de navegação), corresponde à referência do próprio componente, isto é, a identidade do componente coincide com a identidade de sua interface equivalente. Por exemplo, um componente é registrado em um serviço de nomes ou repositório de interfaces através do registro de sua interface equivalente. A interface equivalente define as operações comuns para todos os componentes. Estas operações permitem:

- obter a referência da fábrica (*home*) do componente;
- obter a chave que identifica a instância do componente;
- efetuar a transição do componente de um estado de configuração para um estado operacional;
- obter as referências dos demais elementos estruturais do componente (operações de navegação) tais como portas e elementos de configuração (propriedades). Por exemplo, os clientes deste

componente podem verificar as interfaces proporcionadas por um determinado componente em tempo de execução (por exemplo, para fins de introspecção);

- efetuar as conexões das portas operacionais, conforme descrito na Seção 3.2.2.

A representação do elemento interface equivalente em UML é ilustrada na Figura 3.4. A classe *Component* representa um componente CM-tel e implementa uma interface, a interface equivalente do componente.

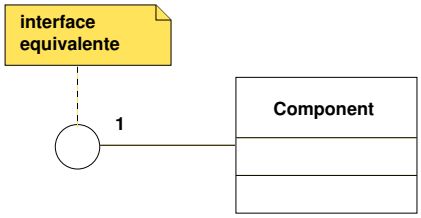

Fig. 3.4: Representação do elemento interface equivalente em UML.

Facetas e Receptáculos

Facetas são interfaces operacionais segundo o modelo RM-ODP. Através destas interfaces, os componentes conectados realizam operações de natureza síncrona no estilo RPC (chamada de procedimento remoto). Uma faceta descreve o aspecto funcional sintático de um componente e representa a interface de propósito geral (interfaces proporcionadas), ou seja, a lógica do componente. Esta lógica consiste de um conjunto de funcionalidades intrínsecas ao componente e implementadas por métodos, que fornecem parte do comportamento do componente. Um componente pode proporcionar zero ou mais facetas, cada qual expondo um aspecto funcional do componente para os seus clientes.

Tal como objetos, componentes contam com o conceito de encapsulação. Assim, a implementação de uma faceta é escondida dentro da implementação do componente e é opaca para o cliente da faceta. Portanto, as facetas não podem estar dissociadas do componente a que pertencem, ou seja, o ciclo de vida de uma faceta está relacionado ao ciclo de vida de um componente. Adicionalmente, cada faceta tem a sua própria referência.

Receptáculos provêem uma forma padrão para especificar as interfaces requeridas pelo componente durante a sua execução. Receptáculos são pontos de interconexão e descrevem a capacidade que um componente tem de aceitar uma referência (isto é, uma referência de uma faceta, uma referência de um componente, ou uma referência de um objeto) suprida por algum agente externo e utilizar as suas operações. Esta relação é denominada uma conexão. Um componente pode se conectar (isto é, obter uma das referências citadas acima) através de um receptáculo a interfaces proporcionadas por

outros componentes, de modo a formar uma composição de componentes. Durante a instalação, este padrão de composição facilita a construção de aplicações distribuídas. Em adição, ele permite que instâncias de componentes deleguem solicitações de serviço. Além disso, receptáculos são úteis para reconfiguração dinâmica, isto porque é possível alterar as conexões durante o ciclo de vida de uma instância de um componente.

Um receptáculo representa o aspecto funcional semântico (comportamental) de um componente e armazena referências para os serviços que o componente requer (interfaces requeridas), que são disponibilizados em outros componentes. Desta forma, os receptáculos são os elementos que tornam os componentes uma evolução real de objetos, onde os serviços utilizados pelo componente são explicitamente descritos. No caso de objetos, os serviços utilizados por objetos são referências escondidas na implementação do objeto. Portanto, utilizando componentes é possível saber que serviços ou interfaces têm que ser proporcionados em tempo de execução para um componente.

Um receptáculo tem as suas funcionalidades realizadas pela interface equivalente. Na realidade, um receptáculo é uma abstração que é concretamente manifestada no componente como um conjunto de operações, disponibilizadas por esta interface, para estabelecer e gerenciar conexões. A referência é associada a um receptáculo durante a fase de configuração ou execução de componentes por estas operações. Um receptáculo simples armazena uma única referência (tipicamente a referência de uma faceta definida em outro componente), enquanto que um receptáculo múltiplo armazena um número arbitrário de referências. No caso de receptáculo múltiplo, uma possibilidade de utilização é em sistemas tolerantes a falhas. Em caso de falha em determinada operação de uma conexão (faceta), a mesma operação poderá ser repetida em outra conexão. Receptáculos múltiplos podem também ser utilizados para prover ao componente uma mesma funcionalidade com diferentes níveis de qualidade de serviço (QoS) como, por exemplo, um receptáculo pode ter uma referência de um componente especificado com um determinado nível de qualidade de serviço e uma outra referência do mesmo componente especificado com um outro nível de qualidade.

A representação de portas operacionais (facetas e receptáculos) em UML é ilustrada na Figura 3.5. A classe *Facet* define uma faceta cujas operações são especificadas como métodos desta classe. A classe *Receptacle* define um receptáculo simples ou múltiplo, de acordo com o valor do atributo *multiple*. As operações de conexão para o receptáculo são disponibilizadas na interface equivalente.

Pontos Terminais de Eventos

Pontos terminais de eventos são interfaces de sinal segundo o modelo RM-ODP, onde as operações são do tipo sentido único (*oneway*). Estes pontos terminais são empregados para propagar notificações assíncronas (eventos) entre componentes, sendo usualmente utilizados para o informe de status e falhas, bem como para sincronizar o processamento entre componentes.

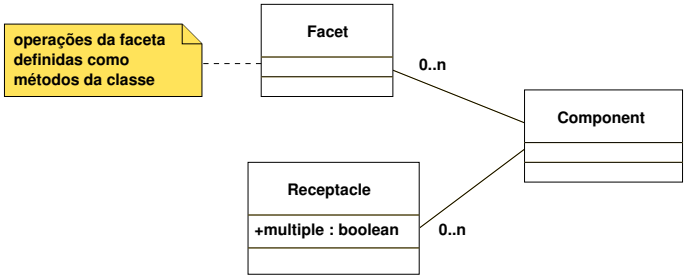

Fig. 3.5: Representação de portas operacionais em UML.

Dois tipos de pontos terminais de eventos são definidos: fontes e consumidores de eventos. Consumidores de eventos são interfaces que recebem notificações através de uma operação tipo *push*. Estas interfaces permitem a um componente consumidor receber eventos de um determinado tipo. A instância de um componente consumidor pode receber eventos de várias fontes diferentes, desde que todas forneçam eventos do mesmo tipo.

Fontes de eventos são interfaces produtoras de eventos e se subdividem em dois tipos: emissores e publicadores de eventos. Emissores de eventos admitem a conexão de uma única interface consumidora de eventos, enquanto que publicadores admitem a conexão de um número arbitrário de interfaces consumidoras ao mesmo tempo. Os eventos gerados nas interfaces produtoras são propagados a todas as interfaces consumidoras conectadas.

A representação de portas de sinal (emissores, publicadores e consumidores) em UML é ilustrada na Figura 3.6. As classes *Emitter*, *Publisher* e *Consumer* definem estes elementos. Uma classe base, *SignalPort* é utilizada para estabelecer uma relação com um determinado tipo de evento, ou seja, o evento que a porta produz ou consome. Eventos são modelados através da classe *Event*. Os parâmetros do evento são definidos como parâmetros desta classe.

Pontos Terminais de Fluxos

Pontos terminais de fluxos são interfaces tipo *stream* segundo o modelo RM-ODP. Estes pontos terminais são empregados para o estabelecimento, controle e gerência de fluxos de mídia contínua (*streams*) entre produtores e apresentadores de mídia.

O modelo CM-tel provê o suporte para a especificação de qualidade de serviço para fluxos de mídia contínua. A plataforma de suporte ao modelo de componentes CM-tel é responsável pela gerência de qualidade de serviço (negociação, monitoração e renegociação dos níveis de qualidade de serviço).

Tal como pontos terminais de eventos, pontos terminais de fluxo possuem dois tipos complementares: produtores e apresentadores de fluxo. Apresentadores de fluxo são interfaces que recebem

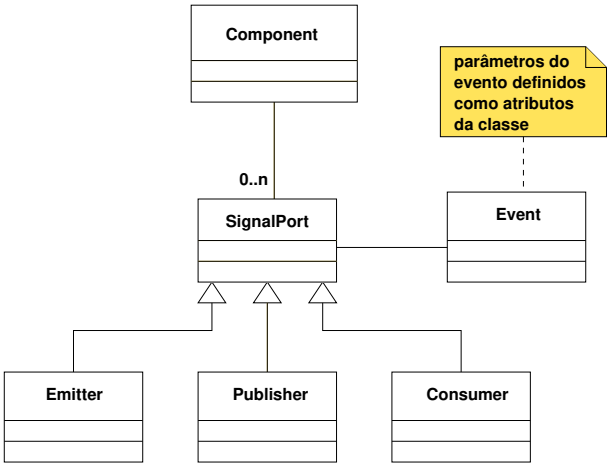

Fig. 3.6: Representação de portas de sinal em UML.

e apresentam fluxos de um único produtor de fluxo, realizando a sua apresentação em um periférico apropriado.

Produtores de fluxo são interfaces capazes de capturar e transmitir áudio e vídeo. Estes produtores se subdividem em duas categorias: transmissores e difusores de fluxo. Transmissores de fluxo estabelecem uma conexão "um-para-um" com uma interface apresentadora de fluxo, enquanto difusores de fluxo estabelecem conexões "um-para-muitos" com interfaces apresentadoras de fluxo.

A representação de portas de fluxo contínuo (transmissor, difusor e apresentador) em UML é ilustrada na Figura 3.7. As classes *Transmitter*, *Broadcaster* e *Player* definem estes elementos. Estas classes derivam de uma classe base, *StreamPort*, que define um atributo, *mediatype*, utilizado para identificar o tipo de mídia manipulado pela porta. Os tipos de mídia mais comuns são áudio ou vídeo. Entretanto, fluxos arbitrários podem ser definidos como, por exemplo, fluxo de dados de telemetria oriundos de um robô.

Elemento de Configuração

Propriedades, ou atributos, são elementos de configuração de componentes. As propriedades de componentes provêem um mecanismo padrão e definem um conjunto de "variáveis de estado" normalmente relacionadas à configuração e ao estado corrente do componente. O elemento de configuração representa a interface que proporciona o acesso às propriedades do componente. Tipicamente, os valores de propriedades são estabelecidos durante a configuração dos componentes. Por exemplo, um componente que implementa uma tela gráfica pode definir propriedades associadas com *look-and-feel* para a tela.

Propriedades podem também ser inspecionadas em tempo de execução e, se permitido, alteradas

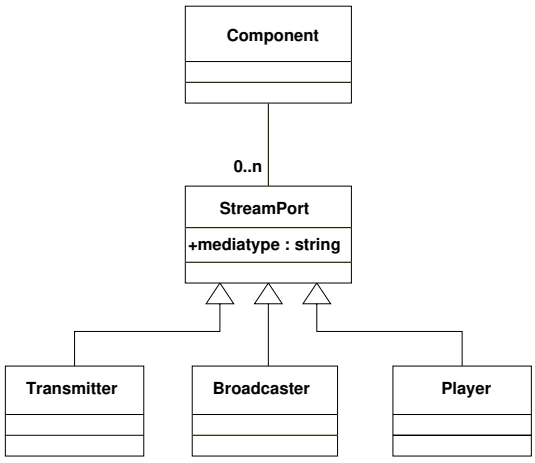

Fig. 3.7: Representação de portas de fluxo contínuo em UML.

por outro componente. Por exemplo, durante a conexão de dois pontos terminais de fluxo, os valores de propriedades relativas ao tipo de mídia e resolução podem ser alterados a fim de compatibilizar os elementos a serem conectados.

Quanto a permissão de acesso, propriedades podem ser de natureza invariante (*read-only*) ou variante (*read-write*). Propriedades são manipuladas através de uma interface específica que define operações de acesso do tipo *get* e *set* para, respectivamente, inspecionar e alterar propriedades. Propriedades invariantes possuem apenas a operação *get* associada.

A representação do elemento de configuração em UML é ilustrada na Figura 3.8. A classe *PropertySet* define um conjunto de propriedades (o elemento de configuração) para o componente. Uma propriedade é modelada pela classe *Property* que por sua vez é especializada em três tipos de propriedades:

1. simples: consiste de um único nome, tipo e valor;

2. seqüência: consiste de uma seqüência (um ou mais) de propriedades simples com o mesmo tipo;

3. estruturada: consiste de uma seqüência (um ou mais) de propriedades simples de tipos arbitrários.

As classes *Simple, Sequence* e *Struct* definem estes três tipos de propriedades. O tipo e a permissão de acesso (leitura ou leitura/escrita) são definidos na classe *Property* para todas as propriedades. Para as propriedades simples, o valor da propriedade é armazenado no atributo *value*.

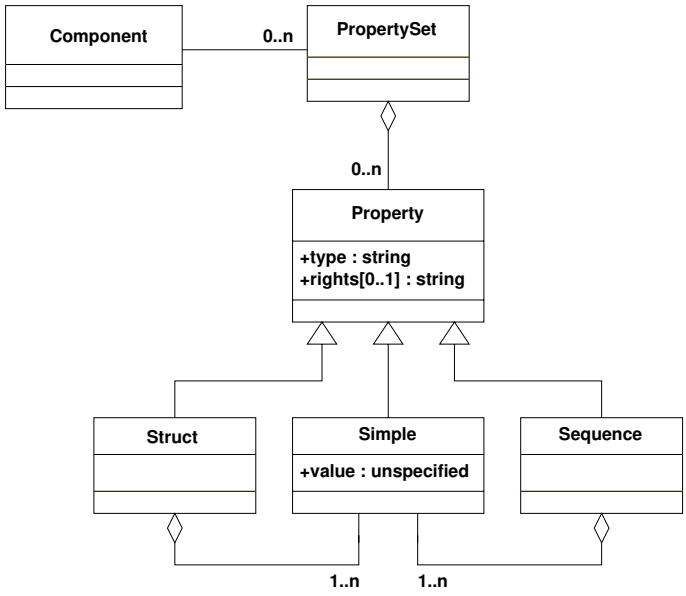

Fig. 3.8: Representação de elementos de configuração em UML.

Metamodelo para Componentes CM-tel

O modelo conceitual para componentes CM-tel em UML, ilustrado na Figura 3.9, define as classes que compõem um componente CM-tel com os respectivos atributos e relações. A figura apresentada consiste da composição das Figuras 3.4, 3.5, 3.6, 3.7 e 3.8.

Este modelo conceitual para componentes CM-tel é, na realidade, um metamodelo UML para o domínio de aplicações telemáticas e ubíquas a partir do qual modelos específicos de determinada aplicação possam ser derivados. A incorporação de restrições a um modelo UML é importante para eliminar ambigüidades do modelo, contribuindo assim para a sua precisão. Restrições podem ser expressas na notação OCL (*Object Constraint Language*) [97] conforme especificado pelo OMG. Por exemplo, a restrição que propriedades definidas como seqüência devem conter propriedades simples de mesmo tipo é expressa em OCL por uma restrição invariante da forma:

```
context Sequence
  -- todas as propriedades simples associadas a uma
  -- sequencia devem possuir tipo identico ao da sequencia
  inv: props->forAll(s:Simple | self.type = s.type) = true
```

Um metamodelo UML pode ser representado por um perfil UML. Um perfil UML é um mecanismo de extensão da linguagem UML para um domínio específio [59]. Esta extensão pode ser aplicada a elementos da linguagem tais como classes, relações, atributos, etc. Um perfil UML é também um modelo UML, sendo portanto manipulado por qualquer ferramenta de projeto que suporte esta lin-

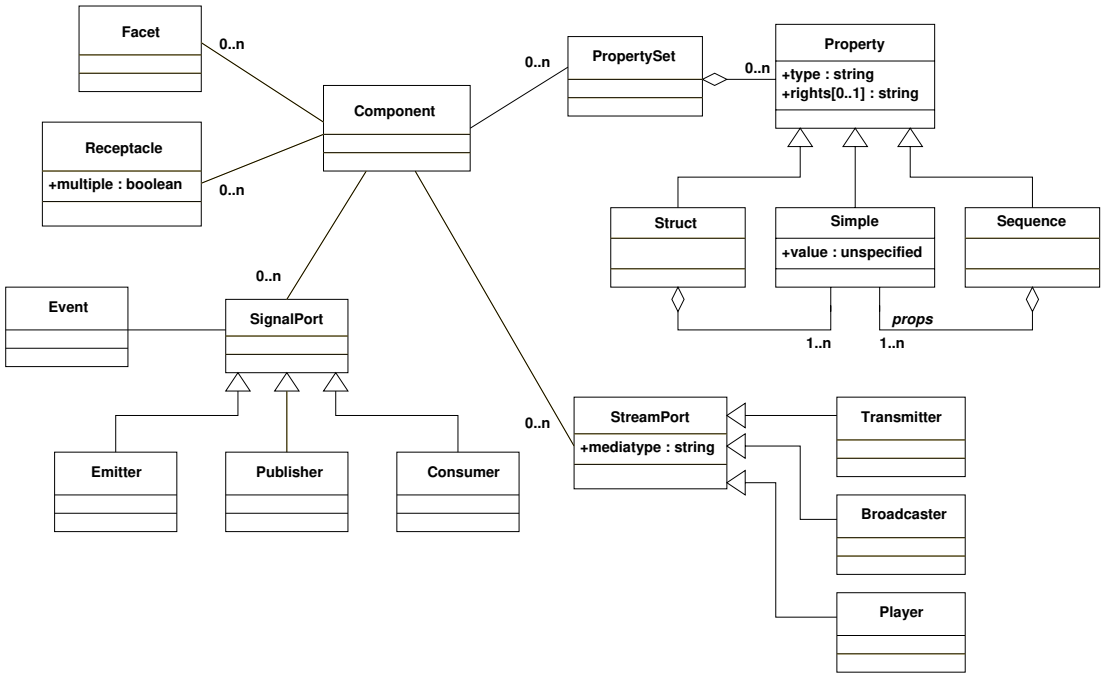

Fig. 3.9: Modelo conceitual para componentes CM-tel.

guagem. A linguagem UML prevê três mecanismos de extensões: estereótipos, valores etiquetados (*tagged values*) e restrições. Estereótipos são marcas representadas por uma palavra entre os delimitadores $<<>>$. Por exemplo, o estereótipo $<< component >>$ quando aplicado a uma classe UML qualquer da aplicação pode estabelecer uma equivalência entre esta classe e a classe *Component* do metamodelo CM-tel. Valores etiquetados são pares atributo-valor que, quando associados a um elemento da linguagem UML, podem acrescentar informação adicional a este elemento. Restrições impõem certas regras para a construção do modelo, por exemplo, exigindo que duas classes sejam ligadas por uma associação.

O perfil UML para componentes CM-tel define estereótipos para todas as classes do modelo conceitual CM-tel, exceto as classes *SignalPort*, *StreamPort* e *Property* (estas classes são empregadas apenas para evitar a replicação de associações ou atributos). O nome do estereótipo concide com o nome da classe do metamodelo em letras minúsculas. A Tabela 3.1 descreve estes estereótipos indicando o elemento da linguagem UML ao qual podem ser aplicados (no caso, *Class* - classes UML) e uma representação gráfica para classes estereotipadas.

Por exemplo, a Figura 3.10 ilustra um componente CM-tel de uma aplicação específica. O componente *PositionGenerator* define um porta publicadora de eventos denominada *PositionPort* que gera eventos do tipo *RobotPosition*. Este evento possui dois atributos do tipo inteiro, *CoordX* e *CoordY*. Os estereótipos associam as classes da aplicação com as classes correspondentes do metamodelo.

Elemento do metamodelo UML	Estereótipo	Descrição	Representação Gráfica
Class	<<component>>	Indica que a classe UML representa um componente CM−tel	 (UML padrão)
Class	<<facet>>	Indica que a classe UML representa uma porta tipo faceta	
Class	<<receptacle>>	Indica que a classe UML representa uma porta tipo receptáculo	
Class	<<emitter>>	Indica que a classe UML representa uma porta emissora de eventos	
Class	<<publisher>>	Indica que a classe UML representa uma porta publicadora de eventos	
Class	<<consumer>>	Indica que a classe UML representa uma porta consumidora de eventos	
Class	<<event>>	Indica que a classe UML representa um evento	
Class	<<transmitter>>	Indica que a classe UML representa uma porta transmissora de fluxo contínuo	
Class	<<broadcaster>>	Indica que a classe UML representa uma porta difusora de fluxo contínuo	
Class	<<player>>	Indica que a classe UML representa uma porta consumidora de fluxo contínuo	
Class	<<propertyset>>	Indica que a classe UML representa um elemento de configuração (conjunto de propriedades)	
Class	<<simple>> <<struct>> <<sequence>>	Indica que a classe UML representa uma propriedade tipo simples, estruturada ou seqüência	

Tab. 3.1: Perfil UML para componentes CM-tel.

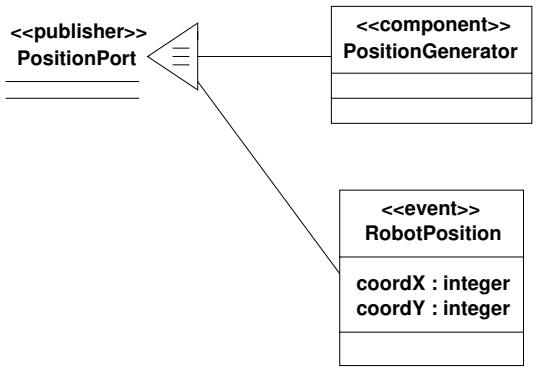

Fig. 3.10: Exemplo de componente CM-tel.

3.2.2 Padrão de Interação para Componentes CM-tel

Os componentes desenvolvidos segundo o modelo CM-tel interagem com entidades externas (tais como clientes, outros componentes ou serviços providos) através de portas. Em adição, por meio de elementos de configuração, CM-tel permite modificar as configurações de um componente.

O mecanismo básico de interação proporcionado pelo modelo CM-tel é baseado em chamada de métodos remotos, portanto obedecendo ao estilo de interação operacional proposto pelo RM-ODP. CM-tel fornece também a interface de sinal e a interface de fluxo contínuo (interface *stream* representadas pelo modelo de comunicação publicador/subscritor, segundo o estilo de interação sinal e *stream* do RM-ODP.

Uma das características do modelo CM-tel é a uniformidade na interação entre componentes. Esta forma de interação é geral para todos os elementos tipo porta e é ilustrada no diagrama UML da Figura 3.11. O modelo provê duas formas de interação entre os componentes: *ponto-ponto* e *ponto-multiponto*. Estas formas de interação podem ser utilizadas como blocos de construção para conexões *multiponto-ponto* e *multiponto-multiponto*. Em adição, elas são modeladas por relações de dependência entre portas (setas pontilhadas).

A forma de interação ponto-ponto permite um único elemento em cada extremidade da conexão. Esta forma estabelece uma relação "um-para-um", ou *unicast*. A conexão ocorre na forma de ligação implícita no sentido do modelo RM-ODP, ou seja, sem a necessidade de objetos de ligação. Conexões ponto-ponto são gerenciadas por operações tipo *connect* e *disconnect*. A operação *connect* permite conectar duas portas complementares, enquanto *disconnect* desfaz a conexão.

A forma de interação ponto-multiponto permite a conexão de um elemento tipo porta com um número arbitrário de portas complementares. Esta forma estabelece uma relação "um-para-muitos", ou *multicast*. Neste caso, um elemento de interconexão é empregado para a gerência de conexões e para a difusão de informação entre as portas conectadas (o objeto de ligação segundo o modelo

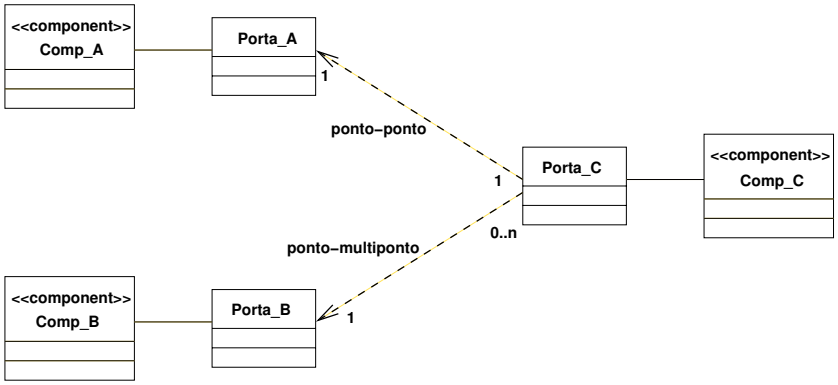

Fig. 3.11: Padrão de interação do modelo CM-tel.

RM-ODP). Conexões ponto-multiponto são gerenciadas por operações tipo *subscribe* e *unsubscribe*. A operação *subscribe* permite acrescentar uma porta à conexão, retornando um identificador para este ramo da conexão. A operação *unsubscribe* desfaz o ramo da conexão identificado pelo parâmetro de entrada. Estas interações, baseadas no padrão de projeto Observador [16], representam os modelos de comunicação publicador/subscritor e produtor/consumidor.

A razão de se empregar no modelo a separação entre as formas ponto-ponto e ponto-multiponto, dado que a primeira forma é um caso particular da segunda forma, se deve ao impacto na implementação. Especificamente, na infra-estrutura a ser gerada pelo modelo, sendo esta mais simplificada para aplicações que empregam a primeira forma. A diferença consiste na geração ou não do elemento de interconexão. Quando a conexão ponto-ponto é empregada, a plataforma de suporte não interpõe o elemento de interconexão entre as portas conectadas, o que simplifica a infra-estrutura gerada que é uma das metas do modelo.

Para facilitar a descrição do padrão de interação CM-tel no suporte às formas de interconexão e operações para cada uma das portas da estrutura de um componente, apresentadas nas seções subseqüentes, utilizamos diagramas UML.

Padrão de Interação entre Portas Operacionais

Componentes CM-tel definem suas funcionalidades intrínsecas por meio de facetas e suas funcionalidades requeridas por meio de receptáculos. Componentes CM-tel podem estabelecer conexões entre um receptáculo e uma ou mais interfaces, tipicamente facetas. Logo que estas conexões são estabelecidas, os componentes podem interagir invocando as operações disponibilizadas nas referências obtidas dos receptáculos.

A Figura 3.12 apresenta a especificação para as duas formas de interação do modelo CM-tel referentes a portas operacionais: faceta-receptáculo simples, correspondente à forma de interação

ponto-ponto e faceta-receptáculo múltiplo, correspondente à forma de interação ponto-multiponto. Estas formas de interação são estabelecidas tanto em tempo de configuração de componentes, quanto em tempo de execução.

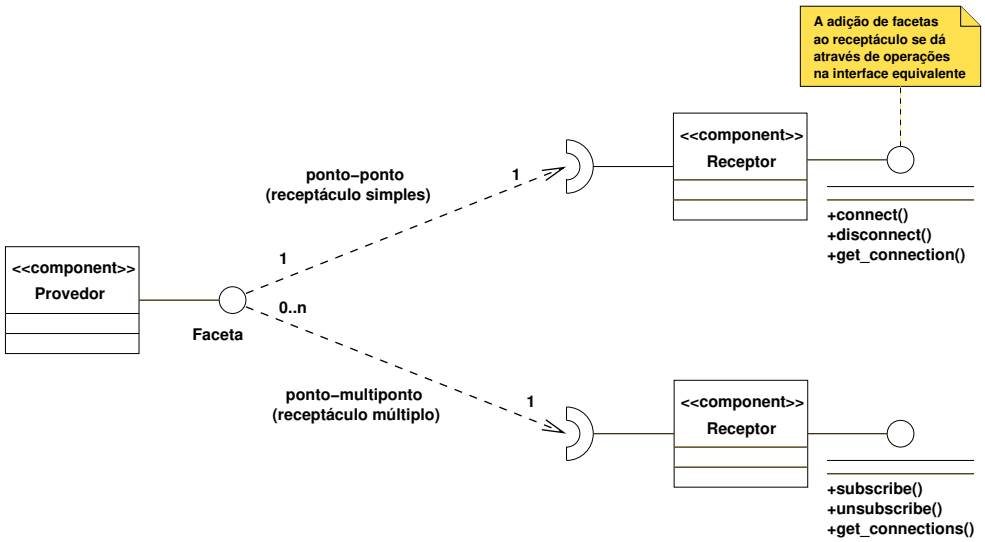

Fig. 3.12: Padrão de interação entre portas operacionais.

As operações de gerenciamento de conexões são obtidas a partir da interface equivalente. Receptáculos simples acrescentam à interface equivalente as operações *connect*, *disconnect* e *get_connection*. A operação *connect* permite armazenar no receptáculo a referência de uma faceta, estabelecendo assim a conexão faceta-receptáculo. A operação *disconnect* remove a referência, desfazendo a conexão. A operação *get_connection* retorna a referência armazenada no receptáculo. Receptáculos múltiplos acrescentam à interface equivalente as operações *subscribe*, *unsubscribe* e *get_connections*. Estas operações são similares às anteriores, exceto que permitem a conexão de múltiplas facetas ao receptáculo.

Padrão de Interação entre Portas de Sinal

O modelo CM-tel emprega, através da porta de sinal, a notificação de eventos do tipo *push*, onde produtores de eventos tomam a iniciativa de notificar os consumidores por ocasião da ocorrência de um determinado tipo de evento (por exemplo, a mudança do valor de uma propriedade do componente). Estas interações fracamente acopladas representam o modelo de comunicação do tipo publicador/subscritor e são muito utilizadas em aplicações distribuídas [98] e em sistemas de tempo real dirigidos a eventos [99][100]. Por exemplo, eventos podem ser utilizados para sincronização (término da missão do robô, por exemplo) e exceções diversas (violação de segurança, término do tempo

reservado a uma sessão de acesso, etc.).

A Figura 3.13 apresenta a especificação para as duas formas de interação do modelo CM-tel referentes à porta de sinal: emissor-consumidor, correspondente à forma ponto-ponto e publicador-consumidor, correspondente à forma ponto-multiponto.

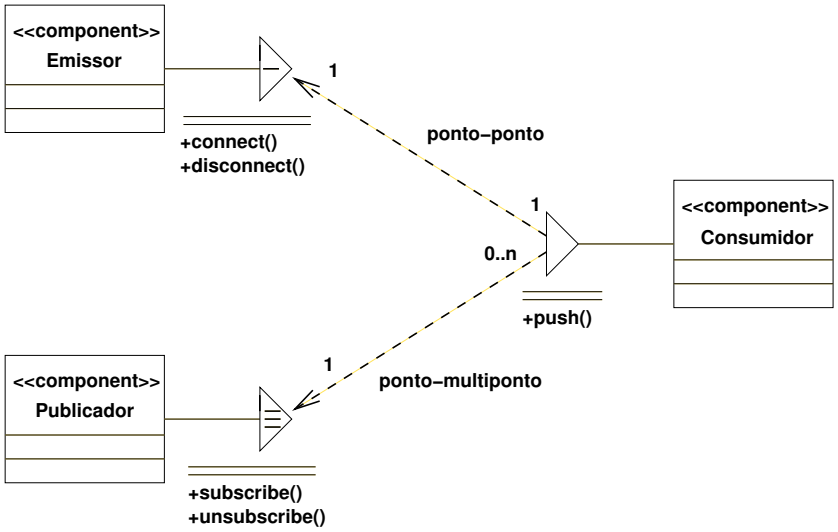

Fig. 3.13: Padrão de interação entre portas de sinal.

No esquema emissor-consumidor, uma forma de interação "ponto-ponto" é estabelecida, conectando diretamente um único componente produtor de eventos a um único componente consumidor destes eventos. Os eventos gerados pelo componente produtor são transmitidos para o componente consumidor deste tipo de evento. No esquema publicador-consumidor, uma forma de interação "ponto-multiponto" é estabelecida, conectando um único componente produtor de eventos a um ou mais componentes consumidores destes eventos. Estas formas de conexão são estabelecidas tanto em tempo de configuração de componentes, quanto em tempo de execução.

CM-tel fornece duas portas produtoras de eventos (*Emitter* e *Publisher*) e uma porta consumidora de eventos (*Consumer*), apresentadas na Figura 3.13. A interface *Emitter* exporta duas operações: *connect* e *disconnect*. A operação *connect* permite conectar um componente consumidor de eventos a um componente produtor de eventos. A operação *disconnect* permite desconectar estes componentes.

A interface *Publisher* exporta também duas operações: *subscribe* e *unsubscribe*. A operação *subscribe* permite conectar um componente consumidor a um componente publicador de eventos, habilitando o consumidor para receber eventos gerados pelo publicador. A operação *unsubscribe* permite desconectar estes componentes.

A interface *Consumer* exporta uma única operação, *push*. Esta operação é utilizada para a notificação de eventos gerados por interfaces produtoras de eventos.

Padrão de Interação entre Portas de Fluxo Contínuo

O modelo CM-tel fornece, por meio da porta de fluxo contínuo, o suporte para o estabelecimento, controle e gerência de fluxos de mídia contínua provendo interfaces capazes de capturar, transferir e apresentar fluxos de áudio e vídeo com facilidades de qualidade de serviço.

Os componentes CM-tel podem estabelecer uma sessão multimídia pela conexão de um ou mais componentes consumidores de mídia a um componente produtor do mesmo tipo de mídia. Estas formas de interação são extremamente úteis em aplicações distribuídas que empregam comunicação multimídia [22] tais como teleconferência, TV interativa, teleimersão e laboratórios virtuais.

A Figura 3.14 apresenta a especificação para as duas formas de interação do modelo CM-tel referentes a interfaces de fluxo contínuo: transmissor-apresentador e difusor-apresentador. Estas formas de interação são estabelecidas tanto em tempo de configuração de componentes, como em tempo de execução.

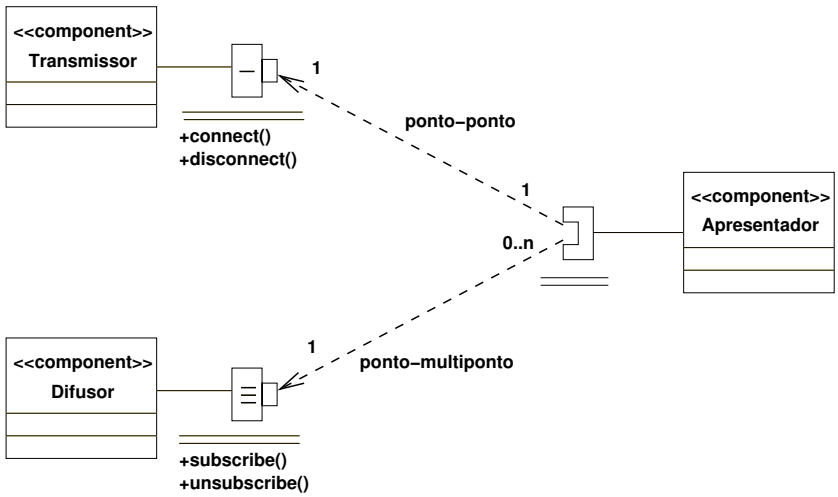

Fig. 3.14: Padrão de interação entre portas de fluxo contínuo.

CM-tel fornece duas portas emissoras de fluxo de mídia contínua (*Transmitter* e *Broadcaster*) e uma porta apresentadora de fluxo (*Player*). A interface *Transmitter* exporta duas operações: *connect* e *disconnect*. A operação *connect* permite conectar um componente apresentador a um componente transmissor de fluxo. A operação *disconnect* permite desconectar estes componentes.

A interface *Broadcaster* exporta duas operações: *subscribe* e *unsubscribe*. A operação *subscribe* permite conectar um ou múltiplos componentes apresentadores a um componente difusor de fluxo, habilitando o apresentador a receber os fluxos de mídia gerados pelo difusor. A operação *unsubscribe* permite desconectar estes componentes. A interface *Player* é utilizada tanto na forma de interação transmissor-apresentador, como na forma difusor-apresentador.

Conforme mencionado anteriormente, CM-tel permite a especificação de qualidade de serviço para as interfaces de fluxo contínuo.

Modelo de Especificação para Componentes CM-tel

A Figura 3.15 complementa as interfaces de um componente CM-tel (Figura 3.9) com as suas respectivas operações. Estas operações são definidas pelo modelo CM-tel conforme apresentado pelos padrões de interação (no caso das portas operacionais, de sinal e de fluxo contínuo) e pelas funções previstas para os elementos de configuração e interface equivalente. E, posteriormente são fornecidas pelo processo de geração automática de código. As operações das facetas são as únicas que devem ser providas pelo desenvolvedor da lógica da aplicação.

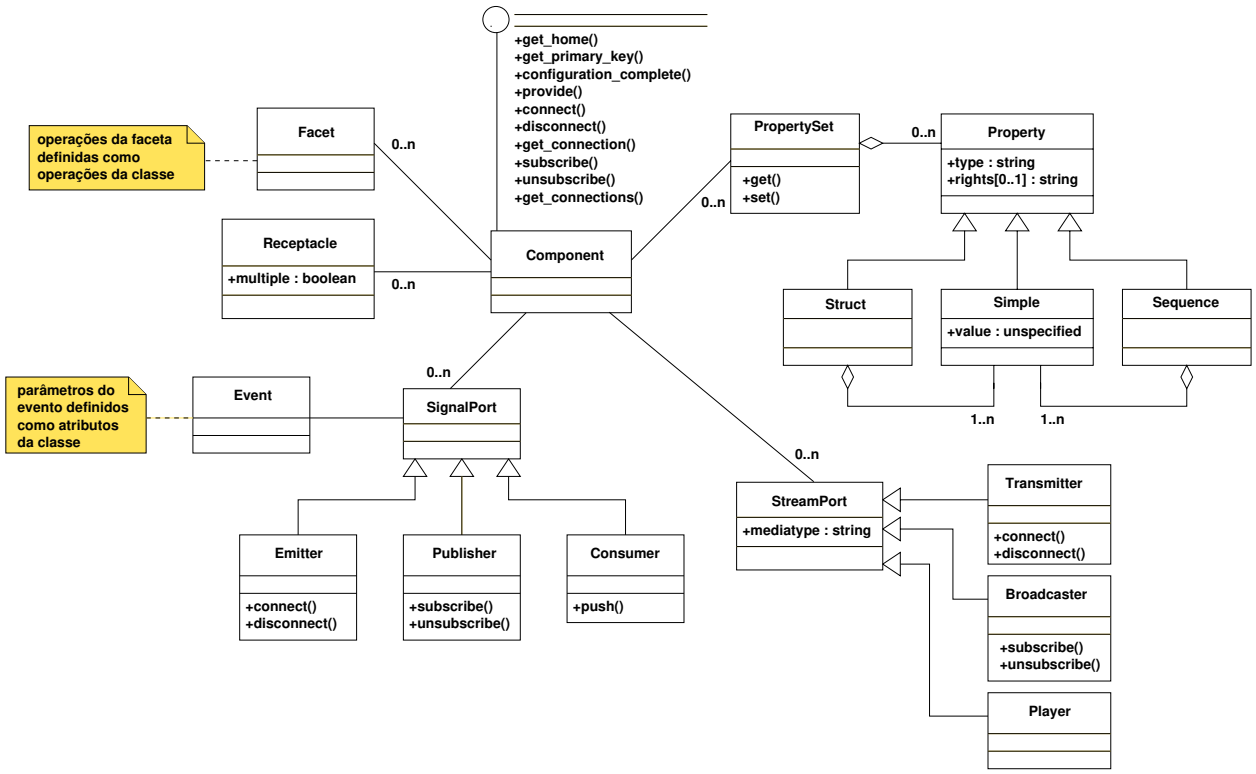

Fig. 3.15: Representação completa de um componente CM-tel.

Especificação de Contratos no Modelo CM-tel

A especificação de um componente por seus aspectos funcionais e não-funcionais deve fornecer elementos que possibilitam tanto a implementação do componente quanto a sua utilização. Com o

objetivo de balisar o uso e a implementação de componentes, o modelo CM-tel acrescenta à especificação do componente dois tipos de contratos: contrato de uso e contrato de realização (definidos na Seção 2.2.1). A Figura 3.16 apresenta o modelo de especificação com os tipos de contratos CM-tel.

Fig. 3.16: Modelo de especificação de contratos CM-tel [1].

Especificação de Contrato de Uso

A especificação de um contrato de uso descreve a relação entre a interface de um componente e um cliente, representando os aspectos funcionais sintáticos e os aspectos funcionais semânticos. Este contrato é especificado para o tempo de execução e é relevante para quem utiliza o componente. No nível de especificação, os aspectos funcionais sintáticos são definidos pelas operações presentes nas facetas. Os aspectos funcionais semânticos, que correspodem à especificação da parte comportamental do componente, podem ser definidos por:

- receptáculos, que especificam as interfaces requeridas pelo componente para a sua execução;
- restrições globais (invariantes) impostas ao componente;
- pré e pós-condições, em que, para cada operação um contrato lista as restrições impostas que precisam ser satisfeitas pelo cliente (pré-condição) e as garantias do resultado (pós-condição), desde que sejam respeitadas as pré-condições no momento da invocação.

CM-tel permite incorporar invariantes, pré e pós-condições associadas a cada operação das facetas de um componente. A linguagem OCL é capaz de expressar invariantes, pré e pós-condições em modelos UML.

Especificação de Contrato de Realização

A especificação de um contrato de realização define como a especificação de um componente será realizada (implementada). Este contrato determina o comportamento do componente que se sujeita às regras impostas pelo modelo CM-tel. O modelo CM-tel define um modelo de projeto que baliza a construção de componentes. Este modelo de projeto, descrito na Seção 3.3.1, fornece toda a infra-estrutura básica necessária para a comunicação e interação entre componentes. Na realidade, este modelo de projeto é o contrato de realização para componentes CM-tel.

3.2.3 Contêineres CM-tel

Um contêiner CM-tel representa o ambiente de execução para um conjunto de implementações de componentes e inclui mecanismos para facilitar o desenvolvimento de aplicações, para integrar os aspectos funcionais providos pelo componente com os requeridos pelos aspectos não-funcionais técnicos e, também para controlar o acesso aos componentes. Em adição, CM-tel provê mecanismos de interceptação, que é uma forma de tornar os contêineres mais flexíveis em tempo de execução permitindo a integração de um comportamento especializado na cadeia de invocação de uma operação.

O modelo de especificação para um contêiner CM-tel, de forma semelhante ao modelo de especificação de componentes, descreve os padrões arquiteturais e convenções definidos pelo modelo para a construção de um ambiente de execução. Estes padrões e convenções contemplam a estrutura deste ambiente CM-tel em termos de suas interfaces, das funcionalidades disponíveis por estas interfaces, bem como a infra-estrutura fornecida pelo modelo CM-tel para o suporte às especificações dos aspectos funcionais e não-funcionais proporcionados pela natureza contratual das especificações. Esta infra-estrutura será apresentada no modelo de projeto para contêineres na Seção 3.3.3 e, na Seção 4.2.1 serão apresentadas as adições relativas ao suporte aos aspectos tecnológicos.

CM-tel mantém as principais estruturas de um contêiner encontradas em modelos de componentes bem estabelecidos tais como EJB e CCM, avaliados no Capítulo 2. Esta similaridade decorre do fato de que os contêineres hospedam componentes CM-tel e provêem alocação dos recursos requeridos aos componentes hospedados neste ambiente. Por exemplo, recursos necessários para que um componente distribuído possa interagir com outros componentes, recursos para prover os aspectos não-funcionais, além de recursos básicos como processamento, comunicação, memória, armazenagem e outras dependências que o componente hospedado necessite. Em adição, a nossa proposta possibilita que um contêiner proporcione o suporte a algumas extensões como, por exemplo, a adição de novos elementos ao contêiner, tais como facilidades de qualidade de serviço para portas de fluxo contínuo e suporte à mobilidade de código.

O modelo conceitual para contêineres CM-tel é apresentado na Figura 3.17. A classe *Container*

define um contêiner CM-tel. Esta classe fornece o suporte necessário aos componentes (relação *supports*), bem como a infra-estrutura necessária para que estes componentes interajam com outros componentes. A relação *depends* identifica a dependência que um componente tem de outros componentes (dependência de contexto).

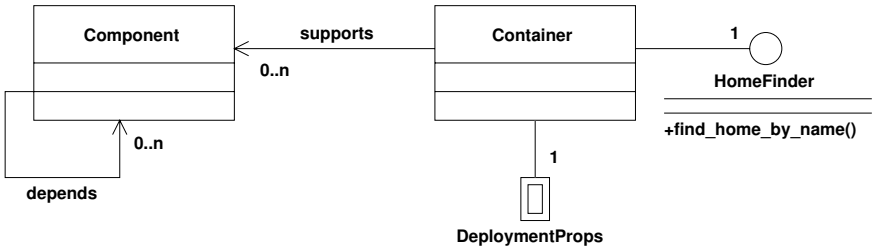

Fig. 3.17: Modelo conceitual para contêineres CM-tel.

Contêineres CM-tel proporcionam a interface *HomeFinder*. Esta interface possibilita a obtenção das referências das fábricas dos componentes e de outros elementos que venham a ser incorporados ao contêiner. Após a obtenção da referência da fábrica desejada, o cliente pode criar ou encontrar a referência do componente desejado. Quando o cliente faz uma requisição para alguma operação destas interfaces, o contêiner pode interceptá-las por meio de interceptadores, personalizando ou adicionando serviços de gerência (propriedades não-funcionais), antes de repassá-las para os componentes de destino. Uma instância de um componente é gerenciada, em tempo de execução, por uma única fábrica. A interface *HomeFinder* define a operação *find_home_by_name*, que retorna a referência da fábrica dado o seu nome.

Adicionalmente, os contêineres CM-tel proporcionam a interface de configuração *DeploymentProps*, conforme apresentado na Figura 3.17. Esta interface provê o acesso às propriedades de um contêiner que foram definidas na sua instalação. Isto permite que uma aplicação altere a configuração do contêiner dinâmicamente. Por exemplo, a aplicação pode alterar o valor da qualidade de serviço de um contêiner fazendo com que componentes que definem portas de fluxo contínuo sejam notificados da alteração (e, eventualmente, efetuem operações de adaptação ao novo nível de QoS).

Extensão de Contêineres CM-tel

O modelo CM-tel possibilita o suporte a algumas extensões de domínio específico ao seu modelo conceitual básico como, por exemplo, a adição de um novo elemento ao contêiner: agente móvel para prover mobilidade de código. Os contêineres CM-tel hospedam e gerenciam agentes móveis, além de componentes, bem como provêem alocação de recursos necessários aos agentes hospedados. Com este suporte, um agente móvel pode, por exemplo, sensoriar o ambiente onde foi instalado e

executar algumas ações; por exemplo, adaptar dinâmicamente certas funcionalidades do contêiner ao ambiente.

A representação completa do contêiner CM-tel com a extensão para incorporar agentes móveis ao modelo é ilustrada na Figura 3.18. A classe *Agent* é utilizada para definir um agente móvel CM-tel.

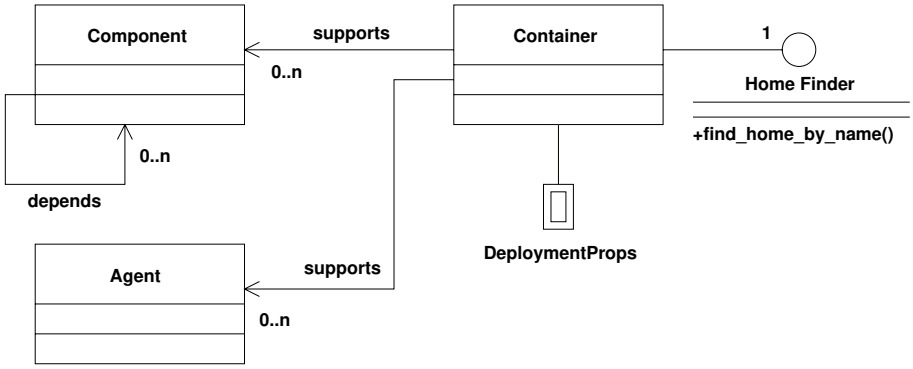

Fig. 3.18: Representação completa de contêiner CM-tel estendido.

De forma semelhante a componentes, a interface *HomeFinder* possibilita a obtenção das referências das fábricas de agentes móveis hospedados no contêiner.

A união dos metamodelos UML estabelecidos para componentes e para contêineres CM-tel (figuras 3.9 e 3.18) forma o metamodelo para aplicações telemáticas e ubíquas.

Perfil UML para Contêineres CM-tel

A Tabela 3.2 define um perfil UML associado ao modelo conceitual (metamodelo) para contêineres CM-tel. Os estereótipos *container* e *agent* permitem relacionar classes do modelo da aplicação com as classes *Container* e *Agent*, respectivamente.

Analogamente aos metamodelos, a união dos perfis UML estabelecidos para componentes e para contêineres CM-tel forma um perfil UML empregado na modelagem UML de aplicações telemáticas e ubíquas.

3.3 Modelo de Projeto CM-tel

O modelo de projeto CM-tel, conforme apresentado na Figura 3.1, corresponde à fase de projeto dos processos de desenvolvimento de *software*. Ele refina o modelo de especificação apresentado para os componentes e contêineres (Seções 3.2.1 a 3.2.3), bem como define padrões arquiteturais e de projeto de infra-estrutura que provêem o suporte ao modelo de componentes. Em adição, o modelo de projeto detalha como é realizado cada um dos elementos da estrutura de um componente e de um

Elemento do metamodelo UML	Estereótipo	Descrição	Representação Gráfica
Class	<<container>>	Indica que a classe UML representa um contêiner CM−tel	
Class	<<agent>>	Indica que a classe UML representa um agente hospedado em contêineres CM−tel	

Tab. 3.2: Perfil UML para contêineres CM-tel.

contêiner CM-tel, ou seja como são estruturados internamente. Neste contexto, os elementos neste nível semântico constituem um conjunto de artefatos que expõem as suas relações e comportamentos internos necessários à construção de componentes e de contêineres. Estes artefatos constituem a base para a maior parte do código gerado para o desenvolvedor da aplicação. Portanto, as operações e os seus respectivos comportamentos internos, descritos a seguir, não são implementados pelo desenvolvedor, mas sim proporcionados pelas plataformas que suportam o modelo CM-tel. A única exceção refere-se às operações dependentes da lógica da aplicação, que devem ser implementadas pelo desenvolvedor.

Nas seções subseqüentes, utilizaremos o termo *Impl* para indicar o artefato de programação que fornece o comportamento de componentes e de contêineres. Por exemplo, *EmitterImpl* refere-se ao artefato que implementa a interface da porta produtora de eventos *Emitter*.

3.3.1 Componentes CM-tel

O modelo de projeto para componentes refina o modelo de especificação descrito anteriormente detalhando a estrutura interna fornecida pelo modelo CM-tel para a construção de componentes, bem como para as operações das interfaces correspondentes aos elementos desta estrutura. Esta estrutura interna contempla um conjunto de artefatos necessários à construção destes componentes. Este conjunto de artefatos forma a infra-estrutura de componentes de *software* denominada *framework* de componentes e proporciona um ambiente contextual, isto é, um "mundo do componente" fornecendo o suporte necessário para a instanciação, composição, especialização e execução de componentes nos seus respectivos contêineres. Note que, dependendo dos elementos definidos na especificação de um componente é gerado um *framework* de componentes contendo os artefatos correspondentes. O *framework* de componentes, no contexto do paradigma de desenvolvimento orientado a componentes, foi definido na Seção 2.2.9 como uma infra-estrutura que viabiliza a composição e instalação de componentes.

A Figura 3.19 apresenta o modelo de projeto para as portas e elementos de configuração de um

componente CM-tel. Esta é a forma geral, imposta pelo modelo, de como cada um dos elementos da estrutura de um componente CM-tel é realizado. Esta forma consiste de uma fábrica para a instanciação da classe que implementa a interface do elemento especificado pelo componente. Esta classe pode necessitar de serviços específicos. Por exemplo, as portas de sinal necessitam de um serviço de notificação, enquanto que as portas de fluxo contínuo necessitam do serviço de controle e gerência de fluxos de mídia contínua. A classe que implementa a interface do elemento pode ainda interagir com parte da aplicação. Por exemplo, uma porta produtora de eventos pode expor uma interface de notificação, onde eventos gerados pela aplicação são notificados (esta notificação gera a emissão de eventos por parte da porta do componente para as portas conectadas de outros componentes). Esta interface de notificação é interna, ou seja, visível apenas aos objetos instanciados no mesmo espaço de endereçamento do componente.

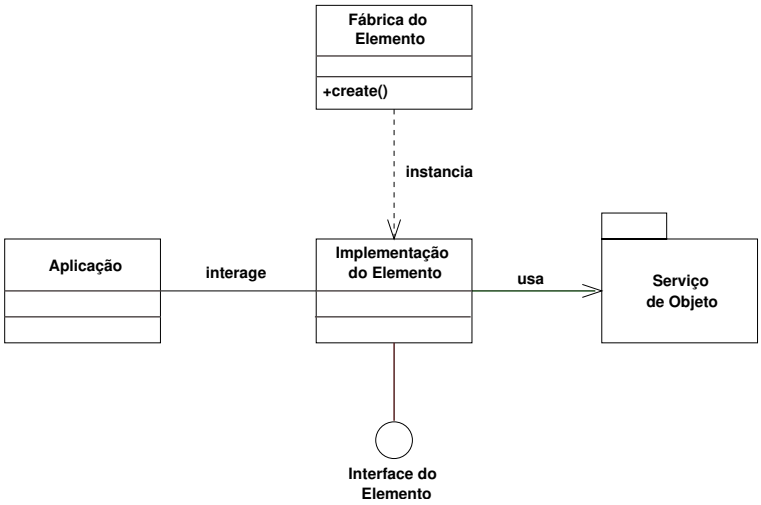

Fig. 3.19: Modelo de projeto para portas e elementos de configuração CM-tel.

Outro aspecto importante é o processo de instanciação de um componente CM-tel, ilustrado na Figura 3.20. Os componentes são instanciados pela fábrica de componentes por meio da operação *create*, que serve como uma interface para que os clientes possam gerenciar as instâncias do respectivo componente. O processo de instanciação mostra que a implementação da operação *create* instancia, inicialmente, a classe que implementa a interface equivalente do componente. A seguir, para cada porta e elemento de configuração definidos pelo componente, a fábrica de componentes interage com a respectiva fábrica de cada elemento a fim de instanciá-lo. Uma vez instanciado o elemento, este é registrado pela fábrica de cada elemento na classe que implementa a interface equivalente, por meio da operação interna *register_element*. Este registro é necessário para que a interface equivalente possa retornar a referência do elemento do componente através de suas operações de navegação.

Como descrito anteriormente, a provisão da infra-estrutura para que um determinado tipo de com-

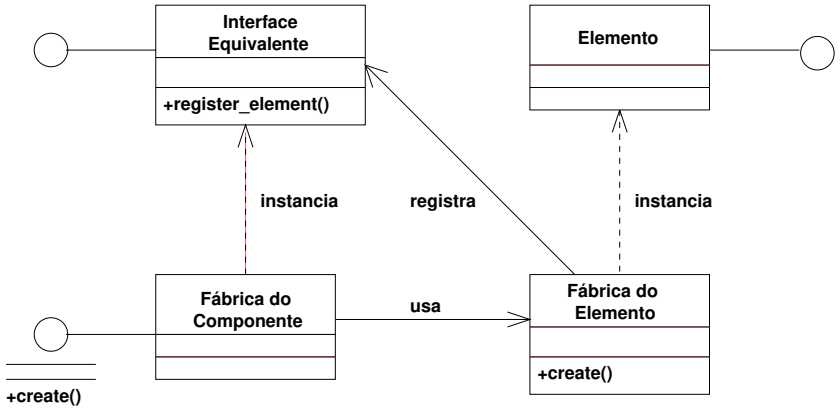

Fig. 3.20: Instanciação de componentes CM-tel.

ponente possa ser instanciado e para que possa interagir com o contêiner e com outros componentes, segundo as regras de interação definidas anteriormente, é fornecida pelo *framework* de componentes CM-tel.

No lado do componente, um *framework* de componentes CM-tel consiste das seguintes classes:

- fábrica do componente;
- fábrica de cada elemento especificado no componente;
- classes que implementam a interface de cada elemento especificado no componente;
- infra-estrutura para acesso remoto às interfaces do componente (*skeletons* das interfaces);
- infra-estrutura de acesso aos serviços de objetos (*stubs* das interfaces dos serviços).

No lado do cliente do componente, um *framework* de componentes CM-tel consiste das seguintes classes:

- infra-estrutura para acesso remoto às interfaces do componente (*stubs* das interfaces);
- um conjunto de classes utilitárias para a interação com o componente:
  1. classes para navegação (localizadoras de componentes no contêiner);
  2. classes para a manipulação de propriedades definidas pelos elementos de configuração do componente.

A seguir é descrito o modelo de projeto para os elementos que compõem o *framework* de componentes.

Modelo de Projeto para Interface Equivalente

O modelo de projeto para a interface equivalente é apresentado na Figura 3.21. As operações definidas na interface equivalente devem ser realizadas na classe que a implementa (*ComponentImpl*). Dependendo dos elementos que são especificados no componente é gerado um conjunto diferente de operações. Por exemplo, caso a especificação de um componente contenha um receptáculo, então as operações específicas para o tipo de receptáculo (simples ou múltiplo) são automaticamente geradas na classe *ComponentImpl*; caso contrário, elas não são geradas.

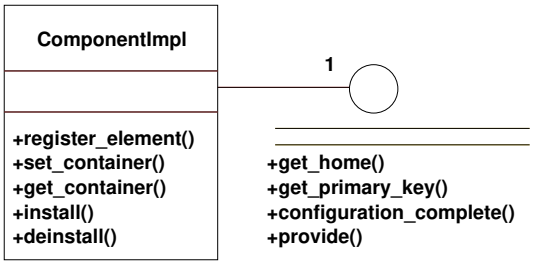

Fig. 3.21: Projeto da interface equivalente para componentes CM-tel.

A classe *ComponentImpl* define as seguintes operações:

1. A operação *get_home* retorna a referência da fábrica que criou o componente, sendo definida em UML como (ObjRef denota a referência para um objeto remoto):

   ```
   get_home() : ObjRef
   ```

   Esta operação não lança nenhuma exceção.

2. A operação *get_primary_key* retorna a chave que identifica o componente, sendo definida em UML como:

   ```
   get_primary_key() : String
   ```

   A operação lança a exceção *NoKeyAvailable*, caso não exista uma chave que identifique esta instância do componente.

3. A operação *configuration_complete* coloca o componente no estado operacional (apto para interagir com outros componentes), sendo definida em UML como:

   ```
   configuration_complete() : void
   ```

   A operação lança a exceção *InvalidConfiguration*, caso a transição do componente do estado de configuração para o estado operacional seja impossível no momento.

4. A operação *provide* é uma operação de navegação, a partir da qual os demais elementos estruturais do componente são acessados, sendo definida em UML como:

   ```
   provide(name: string) : ObjRef
   ```

   O parâmetro *name* fornece o nome do elemento do componente, cuja referência é retornada pela operação. A operação lança a exceção *ElementNotFound* caso o elemento especificado no parâmetro de entrada não exista.

5. Operações para receptáculos, definidas na Seção 3.3.1.

A classe *ComponentImpl* implementa três operações de *callback* invocadas pela fábrica do componente:

1. A operação *set_container* informa ao componente a referência de seu ambiente de execução (contêiner), sendo definida em UML como:

   ```
   set_container(cont : Container) : void
   ```

2. A operação *install* informa ao componente que o mesmo foi instalado no ambiente de execução, sendo definida em UML como:

   ```
   install() : void
   ```

3. A operação *deinstall* informa ao componente que o mesmo está prestes a ser removido de seu ambiente de execução, sendo definida em UML como:

   ```
   deinstall() : void
   ```

A classe *ComponentImpl* implementa ainda duas operações internas:

1. A operação *register_element* registra o elemento instanciado na interface equivalente, sendo definida em UML como:

   ```
   register_element(name : string, element : ObjRef) : void
   ```

   A operação recebe como parâmetros o nome do elemento e a referência de sua interface, e lança a exceção *InvalidName*, caso o nome ou referência do elemento sejam inválidos (por exemplo, nulos).

2. A operação *get_container* retorna a referência do ambiente de execução do componente, sendo definida em UML como:

   ```
   get_container() : Container
   ```

Modelo de Projeto para Fábrica de Componentes

A Fábrica de Componentes (*ComponentHome*) consiste em um objeto que expõe uma interface que define operações de gerência do ciclo de vida de um componente CM-tel tais como operações para criar, encontrar e destruir instâncias dos componentes hospedados no contêiner. Este elemento gerencia o conjunto de todas as instâncias de um tipo específico de componente.

A representação da fábrica de componentes CM-tel é ilustrada na Figura 3.22.

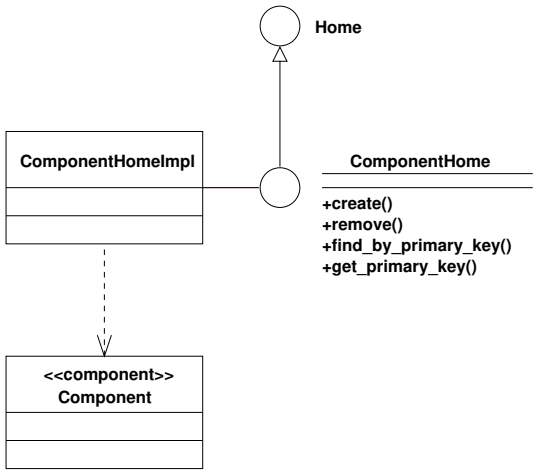

Fig. 3.22: Fábrica de componentes CM-tel (*ComponentHome*).

A interface *ComponentHome* deriva de uma interface base, *Home*. A interface *Home* consiste da base a partir da qual são derivadas as fábricas dos elementos do contêiner como, por exemplo, a fábrica de componentes *ComponentHome* e a fábrica de agentes *AgentHome*. A interface *Home* não define operações. A classe *ComponentHomeImpl* implementa a interface *ComponentHome* e exporta quatro operações:

1. A operação *create* permite criar uma nova instância de um componente que será gerenciada pela sua fábrica de componentes. Esta instância é identificada pela chave primária que é passada como parâmetro da operação. As chaves primárias servem para identificar as instâncias de um componente. Esta operação retorna a interface equivalente do componente e possui a seguinte definição em UML:

   ```
   create(key : string) : ObjRef
   ```

   Esta operação lança as exceções: *InvalidKey*, caso a chave primária passada como parâmetro da operação seja inválida e *DuplicateKeyValue*, caso já exista uma instância de componente associada à chave fornecida.

2. A operação *remove* permite destruir uma instância de um componente. Esta instância é identificada pela chave primária que é passada como parâmetro da operação. A operação *remove* possui a seguinte definição em UML:

   ```
   remove(key : string) : void
   ```

   Esta operação lança as exceções: *InvalidKey*, caso a chave primária passada como parâmetro da operação seja inválida e *UnknownKeyValue*, a chave caso não exista.

3. A operação *find_by_primary_key* retorna a referência base (interface equivalente) da instância do componente identificada pela chave primária que é passada como parâmetro da operação. A operação *find_by_primary_key* possui a seguinte definição em UML:

   ```
   find_by_primary_key(key : string) : ObjRef
   ```

   Esta operação lança as exceções: *InvalidKey*, caso a chave primária passada como parâmetro da operação seja inválida e *UnknownKeyValue*, caso não exista uma instância de componente associada a chave fornecida.

4. A operação *get_primary_key* retorna o valor da chave primária associada com a instância de um componente. A operação *get_primary_key* possui a seguinte definição em UML:

   ```
   get_primary_key(comp : ObjRef) : String
   ```

   Esta operação não lança nenhuma exceção.

Modelo de Projeto para Fábrica de Elementos

A fábrica de elementos consiste de um objeto que expõe a operação *create*. Esta operação permite criar instâncias de cada um dos elementos especificados por um componente CM-tel. A operação recebe como parâmetro a referência da interface equivalente do componente e retorna a referência do elemento criado. A operação *create* possui a seguinte definição em UML:

```
create(inteq: ObjRef) : ObjRef
```

A operação *create* não lança nenhuma exceção.

Modelo de Projeto para Portas Operacionais

O modelo de projeto para as portas operacionais tipo faceta é apresentado na Figura 3.23. As operações definidas na faceta são dependentes da lógica da aplicação e devem ser realizadas na classe que implementa a faceta (*FacetImpl*).

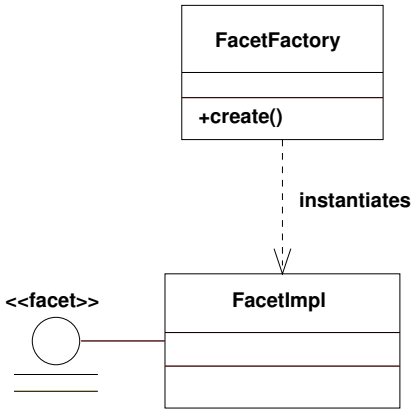

Fig. 3.23: Projeto do elemento faceta.

Quanto aos receptáculos, as suas operações são realizadas na interface equivalente do componente conforme ilustra a Figura 3.24. O primeiro parâmetro destas operações identifica o receptáculo para o qual a operação é direcionada.

1. Para receptáculos simples, as operações *connect*, *disconnect* e *get_connection* possuem a seguinte definição em UML (ObjRef denota uma referência de uma interface remota):

   ```
   connect(rec : Receptacle, peer : ObjRef) : void
   disconnect(rec : Receptacle) : void
   get_connection(rec : Receptacle) : ObjRef
   ```

   A operação *connect* lança a exceção *AlreadyConnected*, caso o receptáculo já esteja conectado a uma interface remota no momento da operação. As operações *disconnect* e *get_connection* lançam a exceção *NoConnection*, caso o receptáculo não esteja conectado no momento da operação.

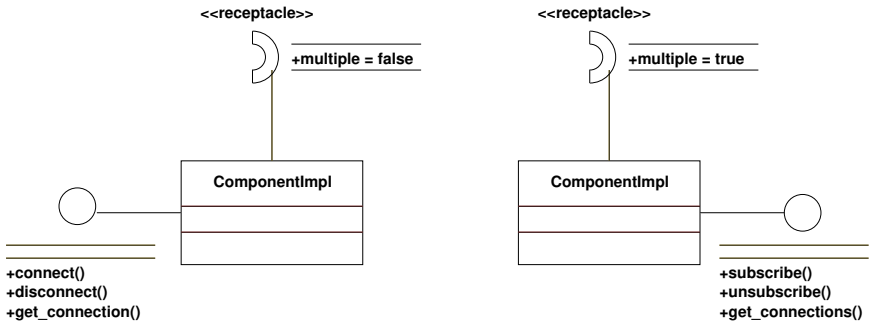

Fig. 3.24: Projeto do elemento receptáculo.

2. Para receptáculos múltiplos as operações *subscribe*, *unsubscribe* e *get_connections* possuem a seguinte definição em UML:

```
subscribe(rec : Receptacle, peer : ObjRef) : Cookie
unsubscribe(rec : Receptacle, conn : Cookie) : void
get_connections(rec : Receptacle) : Cookie[]
```

A operação *subscribe* retorna um identificador de conexão (*cookie*), que é utilizado na operação *unsubscribe* para identificar a conexão a ser desfeita.

A operação *subscribe* lança a exceção *AlreadyConnected*, caso o receptáculo já esteja conectado à referência passada como parâmetro da operação. A operação *unsubscribe* lança a exceção *InvalidConnection*, caso o identificador de conexão não corresponda a uma conexão existente no momento da operação. A operação *get_connections* não lança nenhuma exceção.

Modelo de Projeto para Portas de Sinal

O modelo de projeto para as portas de sinal é apresentado na Figura 3.25. As classes *EmitterImpl* e *PublisherImpl* implementam as portas emissoras e publicadoras de eventos. O evento é propagado por estas classes para todas as portas consumidoras conectadas. A classe *ConsumerImpl* implementa a porta consumidora de eventos.

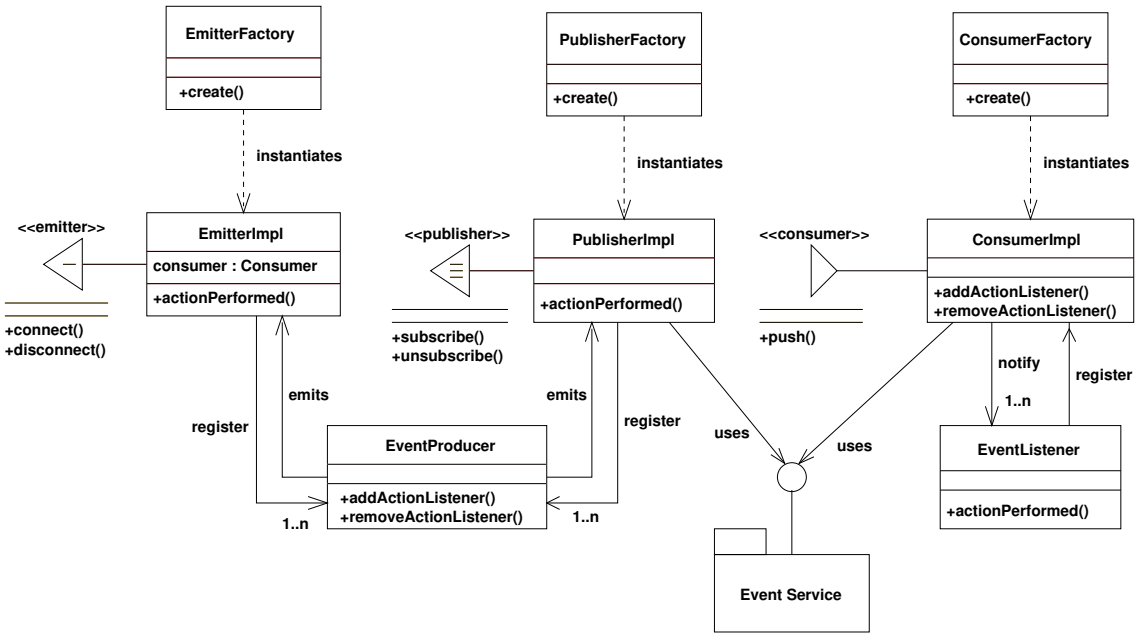

Fig. 3.25: Projeto das portas de sinal.

1. A classe *EmitterImpl* armazena a referência da porta consumidora conectada passada como parâmetro da operação *connect* (esta referência é invalidada pela operação *disconnect*). Eventos gerados pela aplicação são propagados pela própria classe *EmitterImpl*, por meio da invocação da operação *push* disponibilizada na porta consumidora.

   As operações *connect* e *disconnect* possuem a seguinte definição em UML:

   ```
   connect(peer : Consumer) : void
   disconnect() : void
   ```

   A operação *connect* lança a exceção *AlreadyConnected* caso a porta emissora já esteja conectada a um consumidor de eventos no momento da operação. A operação *disconnect* lança a exceção *NoConnection* caso a porta emissora não esteja conectada no momento da operação. A sintaxe de pré e pós-condições em OCL para estas operações é dada abaixo:

   ```
   context EmitterImpl::connect(peer : Consumer)
   pre: peer <> 'null' and self.consumer = 'null'
   post: self.consumer = peer

   context EmitterImpl::disconnect()
   pre: self.consumer <> 'null'
   post: self.consumer = 'null'
   ```

2. No caso de portas publicadoras de eventos a propagação ocorre por um serviço de eventos que opera no estilo *push*. Este serviço deve prover interfaces de gerenciamento para registro de produtores e consumidores de eventos e para submissão de eventos para distribuição. A operação *subscribe* disponibilizada pela porta publicadora registra a porta consumidora passada como argumento no serviço de eventos e retorna um identificador da conexão (*cookie*). A operação *unsubscribe* recebe como parâmetro um identificador de conexão e desregistra a respectiva porta consumidora. Eventos gerados pela aplicação são submetidos ao serviço de eventos para difusão, através da interface de difusão do serviço, para todas as portas conectadas .

   As operações *subscribe* e *unsubscribe* possuem a seguinte definição em UML:

   ```
   subscribe(peer : Consumer) : Cookie
   unsubscribe(conn : Cookie) : void
   ```

   A operação *subscribe* lança a exceção *AlreadyConnected* caso a porta emissora já esteja conectada ao consumidor de eventos passado como parâmetro da operação. A operação *unsubscribe* lança a exceção *InvalidConnection* caso o identificador de conexão não corresponda a uma conexão existente no momento da operação.

3. As portas consumidoras de eventos recebem eventos a partir de um serviço de eventos que opera no estilo *push*. A interface *Consumer* define a operação *push*. A implementação desta operação recebe eventos gerados pelas portas produtoras de eventos conectadas.

   A operação *push* possui a seguinte definição em UML:

   ```
   push(evt : Event) : void
   ```

   A operação lança a exceção *Disconnected* caso a porta consumidora esteja desconectada no momento da operação.

Modelo de Projeto para Portas Stream

O modelo de projeto para as portas de fluxo contínuo (*stream*) é apresentado na Figura 3.26.

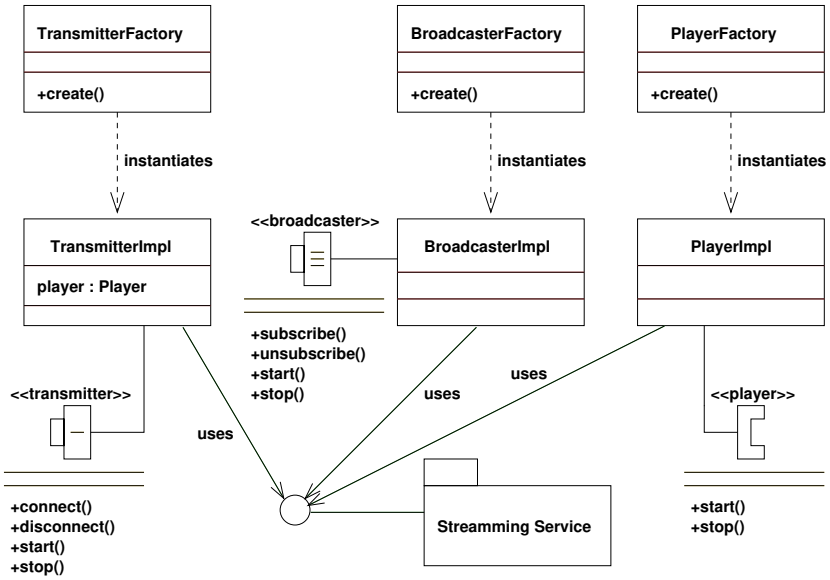

Fig. 3.26: Projeto dos elementos de fluxo contínuo.

As classes *TransmitterImpl*, *BroadcasterImpl* e *PlayerImpl* implementam as portas transmissoras, difusoras e apresentadoras de fluxo contínuo, respectivamente. Estas classes utilizam um serviço de controle e gerência dos fluxos de áudio e vídeo.

1. As operações *connect* e *disconnect* do elemento *Transmitter* possuem a seguinte definição em UML:

   ```
   connect(peer : Player) : void
   disconnect() : void
   ```

A operação *connect* lança a exceção *AlreadyConnected*, caso a porta transmissora já esteja conectada a um apresentador de mídia no momento da operação. A operação *disconnect* lança a exceção *NoConnection*, caso a porta emissora não esteja conectada no momento da operação.

2. As operações *subscribe* e *unsubscribe* do elemento *Broadcaster* possuem a seguinte definição em UML:

```
subscribe(peer : Player) : Cookie
unsubscribe(conn : Cookie) : void
```

A operação *subscribe* lança a exceção *AlreadyConnected* caso a porta difusora já esteja conectada ao apresentador de mídia passado como parâmetro da operação. A operação *unsubscribe* lança a exceção *InvalidConnection* caso o identificador de conexão não corresponda a uma conexão existente no momento da operação.

3. As operações de controle de fluxos multimídia: *start* e *stop* presentes em todos os elementos de fluxo contínuo possuem a seguinte definição em UML:

```
start() : void
stop() : void
```

Estas operações não lançam exceções.

Modelo de Projeto para Elementos de Configuração

O modelo de projeto para as portas de configuração é apresentado na Figura 3.27.

A classe *PropertySetImpl* implementa a interface do elemento de configuração. Esta classe utiliza um serviço de propriedades (*PropertyService*) para a manipulação remota das propriedades de um componente por meio de duas operações. A operação *set* permite criar e alterar propriedades, enquanto a operação *get* permite acessar propriedades. Estas operações são definidas em UML como:

```
set(prop : Property) : void
get(name : string) : Property
```

As operações lançam a exceção *PropertyException* em caso de falha na criação ou alteração da propriedade (por exemplo, alterar o valor de uma propriedade invariante), ou em caso de acesso a uma propriedade inexistente (operação *get*).

A operação interna *getProperty* obtém uma propriedade específica de um componente. A operação *create* da fábrica *PropertySetFactory* permite criar um elemento de configuração vazio ou com um conjunto inicial de propriedades.

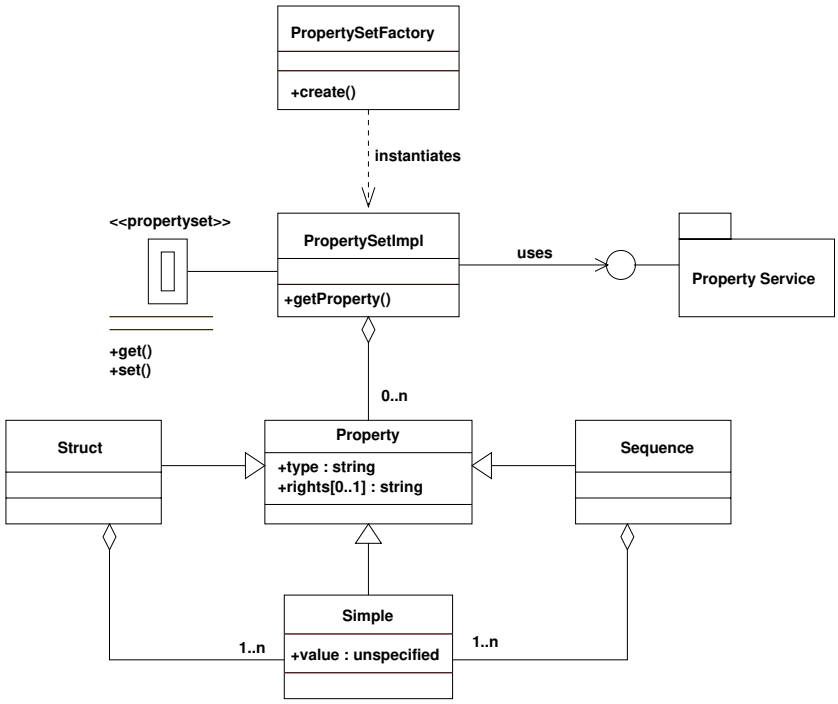

Fig. 3.27: Projeto do elemento de configuração.

3.3.2 Especificação de Componentes CM-tel em XML

No modelo de projeto, os componentes e contêineres CM-tel são descritos por meio de documentos XML (*Extensible Markup Language*) [29]. A linguagem XML é completamente independente de tecnologia em termos de linguagem de programação, sistema operacional e plataforma de *middleware*. CM-tel define *XML Schemas* [101][102] de forma que componentes e contêineres CM-tel possam ter as suas especificações em XML validadas de acordo com as regras impostas pelo modelo CM-tel.

A especificação em XML de componentes CM-tel deve ser coerente com a especificação do modelo conceitual para componentes (Figura 3.9), ou seja, todas as informações presentes em uma especificação devem também estar presentes na outra. Esta equivalência de especificações somente é garantida se o *XML Schema* for diretamente derivado do modelo conceitual em XML.

Carlson [103][27] propôs um perfil UML para a geração de *XML Schemas* a partir de diagramas de classe UML. Este perfil define como mapear elementos de modelagem UML (classes, atributos, relações e estereótipos) em elementos compatíveis com a especificação de *XML Schemas*. Resumidamente, classes UML são mapeadas em elementos do *XML Schema*; atributos UML com o estereótipo *XSDAttribute* são mapeados em atributos dos elementos XML; relações de herança em UML são mapeados em extensões de elementos XML; e relações UML são mapeadas em seqüências de elementos

XML. Ferramentas de modelagem com suporte para XML como HyperModel[4] são capazes de gerar *XML Schemas* a partir de especificações UML aderentes ao perfil UML acima citado.

Para a geração de *XML Schemas*, o diagrama UML do modelo de especificação para componentes CM-tel, (Figura 3.9), foi transformado manualmente em um diagrama equivalente e aderente ao perfil UML de Carlson. Este novo diagrama é apresentado na Figura 3.28. O modelo de projeto preserva a estrutura do modelo de especificação (modelo conceitual), daí a semelhança entre os modelos do ponto de vista estrutural. Cada classe definida no modelo de especificação é transformada em uma classe no perfil UML (e, portanto, em um elemento no *XML Schema*). Classes e atributos ausentes no modelo de especificação foram acrescentados no modelo de projeto com a finalidade de preservar todas as informações do modelo de especificação. Estas adições compensam basicamente a ausência de estereótipos e operações nas classes do modelo de projeto, restrições estas impostas pela ausência destes elementos UML em *XML Schemas*.

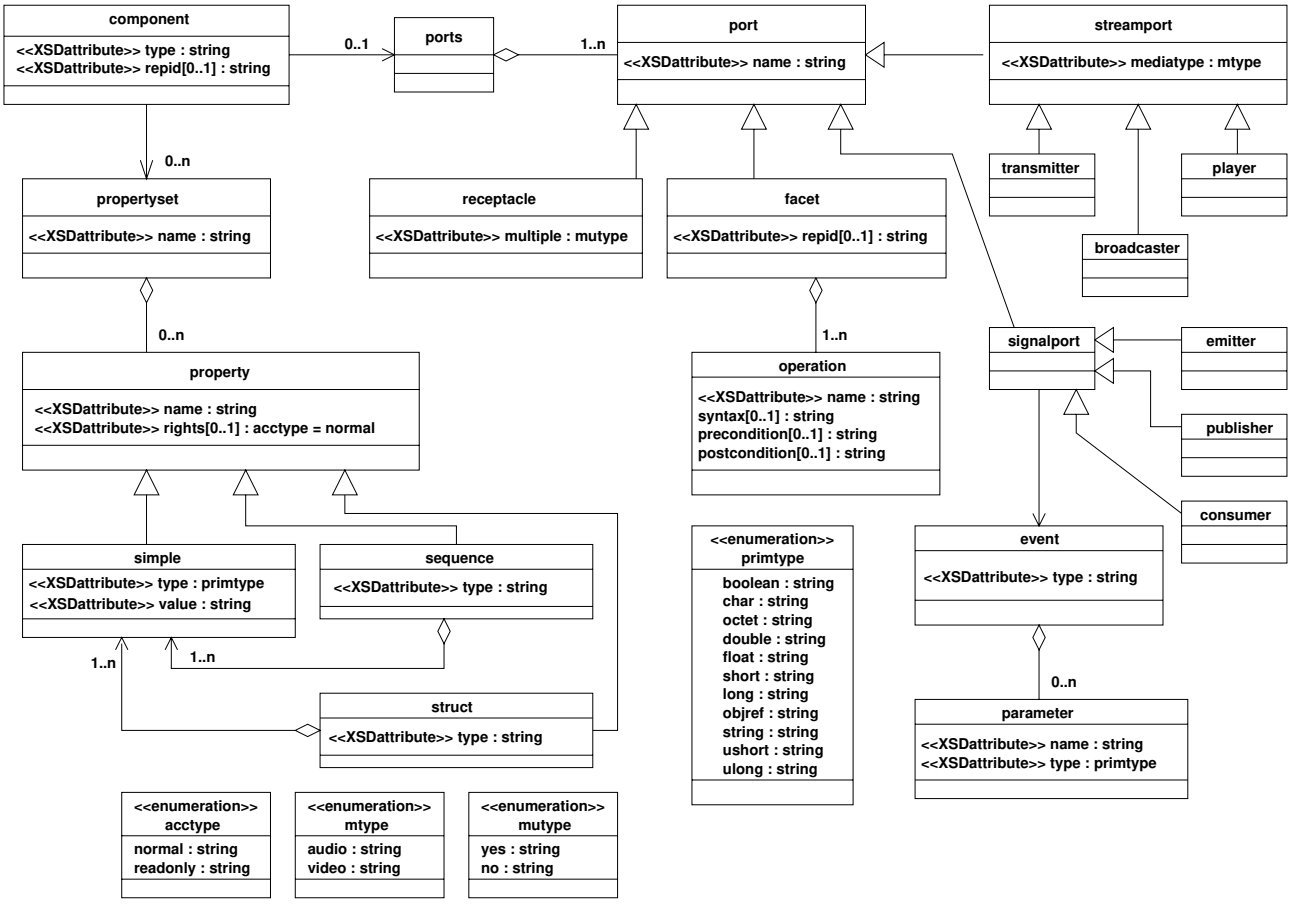

Fig. 3.28: Representação em UML de um *XML Schema* para a especificação de componentes CM-tel.

[4] `http://www.xmlmodeling.com/products/hyperModel/`

A partir do diagrama UML da Figura 3.28 foi gerado um *XML Schema*. A geração é automática e emprega o auxílio de ferramentas de modelagem com suporte para XML. No nosso caso, foi utilizada a ferramenta HyperModel para a geração do *XML Schema*. O trecho abaixo ilustra um fragmento do *XML Schema* que descreve um componente CM-tel em XML como um agregado de portas e elementos de configuração.

```
<xs:element name="component" type="componentType"/>
 <xs:complexType name="componentType">
    <xs:sequence>
       <xs:element ref="ports" minOccurs="0" maxOccurs="1"/>
       <xs:element ref="propertyset" minOccurs="0"
           maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required"/>
    <xs:attribute name="repid" type="xs:string" use="required"/>
 </xs:complexType>
```

A seguir são fornecidas, por meio de exemplos, as estruturas de documentos XML válidas para representar componentes CM-tel.

Especificação XML para um componente CM-tel

Um componente mínimo CM-tel é ilustrado abaixo. Este componente possui apenas a interface equivalente, ou seja, não foi especificado nenhuma porta ou elementos de configuração. A sua especificação XML requer o tipo (classe) do componente e, opcionalmente, a sua identificação em um repositório de componentes. Estas informações são supridas pelos atributos *type* e *repid* (opcional), respectivamente.

```
<?xml version="1.0" ?>
<component xmlns:xsi="http://www.w3.org/2001/XMLShema-instance"
           xsi:noNamespaceSchemaLocation="comp.xsd"
    type=" C_MinimalComp"
    repid="IDL: C_MinimalComp:1.0">
</component>
```

Especificação XML para Portas Operacionais

As portas operacionais são identificadas pelas marcações XML *facet* e *receptacle* correspondentes às portas operacionais faceta e receptáculo, respectivamente. As portas são identificadas pelo atributo *name*.

No caso de receptáculos, o atributo *multiple* define o tipo da conexão simples ou múltipla proporcionada pelo receptáculo. Os valores possíveis para o atributo *multiple* são definidos no *XML Schema* e correspondem aos atributos da classe *mutype*, "yes" ou "no", ilustrados na Figura 3.28.

O exemplo abaixo ilustra uma porta correspondente a um receptáculo simples:

```
<ports>
   <receptacle
       name="RS_ControlRef"
       multiple="no"/>
</ports>
```

No caso de facetas, o atributo *name* identifica a faceta e o atributo opcional *repid* define a sua identificação em um repositório de componentes. As operações das facetas são definidas pela marcação XML *operation*, correspondente ao conjunto de funcionalidades intrínsecas ao componente. Estas funcionalidades são descritas pelas marcações *syntax*, *precondition* e *postcondition*. A marcação *syntax* permite inserir a sintaxe da operação em uma notação arbitrária. As marcações *precondition* e *postcondition* permitem descrever, em notação textual, as pré e pós-condições associadas à operação.

O exemplo abaixo ilustra uma porta correspondente a uma faceta com uma operação e suas pré e pós-condições. A sintaxe da operação neste exemplo é expressa em CORBA/IDL.

```
<ports>
  <facet
   name = "FA_BasicControl">
     <operation name = "RB_MoveToTarget">
        <syntax>void RB_MoveToTarget(in long x, in long y);</syntax>
        <precondition>
             pre: x >= 0 and y >= 0
        </precondition>
        <postcondition>
             post: -- Robot on the new position
        </postcondition>
     </operation>
   </facet>
 </ports>
```

Especificação XML para as Portas de Sinal

As portas de sinal são identificadas pelas marcações XML *emitter, publisher* e *consumer* correspondentes às portas emissoras, publicadoras e consumidoras de eventos, respectivamente. As portas são identificadas pelo atributo *name* e definem o tipo de evento que são capazes de processar. Os

eventos são definidos através da marcação *event*, sendo o tipo de evento suprido através do atributo *type*. Os parâmetros do evento são fornecidos pelas marcações *parameter* que possuem dois atributos: *name* (nome do parâmetro) e *type* (tipo de parâmetro). O atributo *type* admite um conjunto de valores representando tipos primitivos (*boolean, long, string, etc.*). Os tipos válidos são especificados no *XML Schema* e correspondem aos atributos da classe *primtype* na Figura 3.28.

O exemplo abaixo ilustra a especificação XML de uma porta emissora de eventos de nome *EM_PositionSource* que emite eventos do tipo *RobotPosition*. O evento define dois parâmetros do tipo *long* representando as coordenadas de posição. As portas publicadoras e consumidoras de eventos são definidas da mesma forma.

```
<ports>
  <emitter name="EM_PositionSource">
     <event type="RobotPosition">
        <parameter name="X-position" type="long"/>
        <parameter name="Y-position" type="long"/>
     </event>
  </emitter>
</ports>
```

Especificação XML para as Portas Stream

As portas de fluxo (*stream*) são identificadas pelas marcações XML *transmitter, broadcaster* e *player*, correspondentes às portas transmissoras, difusoras e consumidoras de mídia contínua, respectivamente. As portas são identificadas pelo atributo *name* e definem o tipo de mídia contínua que manipulam, por meio do atributo *mediatype*. Os valores possíveis para o atributo *mediatype* são definidos no *XML Schema* e correspondem aos atributos da classe *mtype* na Figura 3.28 (“audio” ou “video”).

O exemplo abaixo ilustra uma porta transmissora de vídeo de nome *TR_VideoCamera*. As portas difusoras e consumidoras de fluxo são definidas da mesma forma.

```
<ports>
  <transmitter name="TR_VideoCamera" mediatype="video"/>
</ports>
```

Especificação XML para Elementos de Configuração

Os elementos de configuração são definidos pela marcação XML *propertyset*. O atributo *name* identifica o elemento de configuração. As propriedades são definidas por três marcações XML:

- *simple*: define propriedades simples, na forma de pares atributo-valor. A marcação admite cinco atributos:

  1. *name*: o nome do atributo;
  2. *type*: o tipo do valor, conforme estipulado pelos atributos da classe *primtype* na Figura 3.28;
  3. *value*: o valor do atributo;
  4. *rights* (opcional): direitos de manipulação, conforme estipulado pelos atributos da classe *acctype* na Figura 3.28, ou seja "normal", significando direito de leitura/escrita (*default*) ou "readonly" (leitura apenas);
  5. *description* (opcional): uma descrição textual da propriedade.

- *sequence*: uma seqüência de propriedades simples de mesmo tipo. A marcação *sequence* possui quatro atributos:

  1. *name*: o nome da seqüência;
  2. *type*: o tipo dos elementos que compõem a seqüência;
  3. *rights* (opcional): os direitos de manipulação opcional;
  4. *description* (opcional): uma descrição textual.

- *struct*: uma seqüência de propriedades simples de tipos arbitrários. Os atributos são os mesmos de propriedades tipo *sequence*. Neste caso, o atributo *type* atribui um novo tipo para a estrutura.

O exemplo abaixo define um elemento de configuração.

```
<propertyset name="PS_Props1">
  <simple name="background" type="string" value="white"
      rights = "normal"
      description = "Enables to set color atributes"/>

  <sequence name="foreground" type="long" description="RGB color">
     <simple name="R" type="long" value="0"/>
     <simple name="G" type="long" value="65536"/>
     <simple name="B" type="long" value="0"/>
  </sequence>
</propertyset>
```

3.3.3 Contêineres CM-tel

O modelo de projeto para contêineres refina o modelo de especificação para contêineres CM-tel, detalhando a infra-estrutura interna fornecida pelo modelo CM-tel para a construção de contêineres, as operações das interfaces correspondentes aos elementos desta estrutura, bem como descreve o que proporciona de suporte aos aspectos não-funcionais. A Figura 3.29 ilustra os principais elementos de projeto agregados pelo contêiner, no contexto do modelo de projetos.

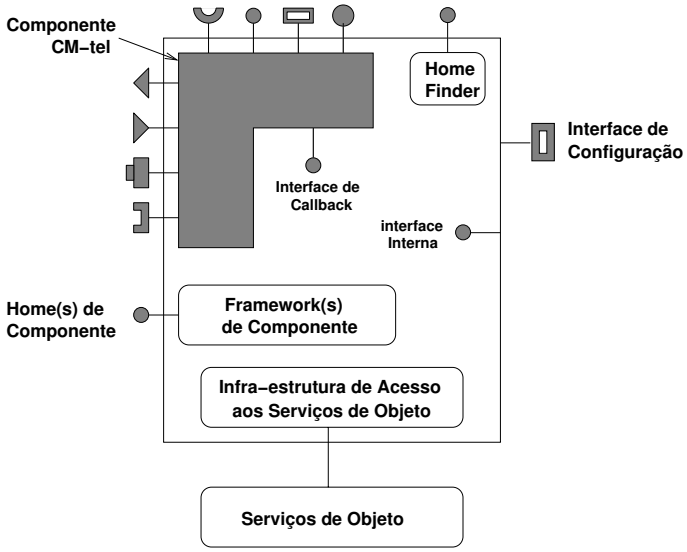

Fig. 3.29: Modelo de projeto para contêineres CM-tel.

Os elementos estruturais do contêiner são:

- Interfaces externas do componente: objetos que expõem interfaces externas dos componentes hospedados, disponíveis para os clientes destes componentes. Estas interfaces estabelecem o contrato de uso entre o desenvolvedor dos componentes e o cliente destes componentes;
- Interface interna: interface local que é disponibilizada para que o componente hospedado possa acessar os serviços proporcionados pelo contêiner;
- Interface callback: interfaces local que é implementada pelo componente, com operações que podem ser invocadas pela sua fábrica no contêiner (por exemplo, quando ocorre a remoção ou destruição de um componente), conforme descrito na Seção 3.3.1;
- Interface de Configuração: contém um conjunto de propriedades para a sua especialização;
- Infra-estrutura de Acesso aos Serviços de Objeto: são serviços e funcionalidades (por exemplo, interceptadores) que o contêiner interpõe entre clientes e componentes;

- Interface Localizador de Fábrica (*HomeFinder*).

Modelo de Projeto para Localizador de Fábricas CM-tel

A interface do Localizador de Fábrica (*HomeFinder*) é implementada pela classe *HomeFinderImpl*. Esta classe oferece duas operações (Figura 3.30):

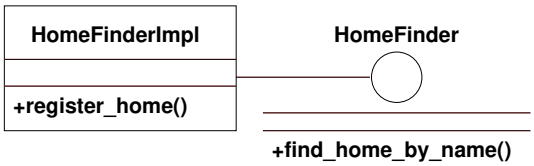

Fig. 3.30: Projeto para o localizador de fábricas.

1. A operação *find_home_by_name*, permite obter a referência de uma fábrica cujo nome é passado como parâmetro. Esta operação possui a seguinte definição em UML:

   ```
   find_home_by_name(name : string) : ObjRef
   ```

   Esta operação lança a exceção *HomeNotFound*, caso o nome passado como parâmetro da operação não seja encontrado.

2. A operação interna *register_home*, permite registrar uma referência de uma fábrica na interface *HomeFinder*. Esta operação recebe como parâmetro o nome da fábrica de um componente e a sua referência. Esta operação possui a seguinte definição em UML:

   ```
   register_home(name : string, ref : ObjRef) : void
   ```

   Esta operação lança as exceções *InvalidKey*, caso o nome da fábrica passado como parâmetro da operação seja inválido e *DuplicateKeyValue*, caso já exista uma referência associada ao nome fornecido.

Modelo de Projeto para Extensão do Contêiner CM-tel

A extensão do contêiner consiste na adição de dois novos elementos ao contêiner: agentes móveis e o *framework* de agentes. A Figura 3.31 ilustra a adição destes novos elementos agregados pelo contêiner, no contexto do modelo de projeto.

Semelhante ao *framework* de componentes, o *framework* de agentes móveis contempla um conjunto de artefatos de *software* que suporta a comunicação, mobilidade, instanciação e execução de

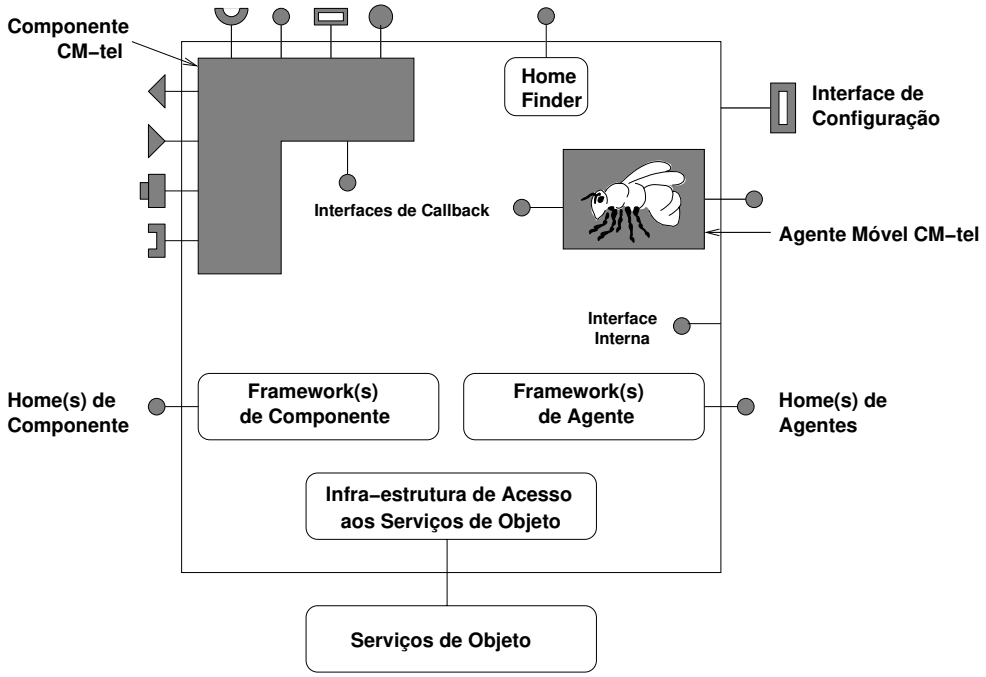

Fig. 3.31: Projeto para extensão de contêineres CM-tel.

agentes móveis em contêineres. Ainda de forma semelhante ao *framework* de componentes, os *frameworks* de agentes proporcionam um ambiente contextual, isto é, um "mundo do agente", com serviços de suporte que disponibilizam os recursos que um agente CM-tel necessite. O modelo de projeto de um agente móvel CM-tel é apresentado na seqüência desta seção.

No lado do contêiner, o *framework* de agentes móveis CM-tel consiste das seguintes classes:

- fábrica do agente (*AgentHome*);
- infra-estrutura de acesso remoto à interface da fábrica de agentes móveis (*skeletons* da interface);
- infra-estrutura de acesso aos serviços de objetos (*stubs* da interface do serviço de suporte a agentes móveis).

No lado do cliente, o *framework* de agentes móveis CM-tel consiste da seguinte classe:

- infra-estrutura de acesso remoto à interface da fábrica de agentes móveis (*stubs* da interface).

Um cliente utilizando uma interação com um componente ou agente móvel CM-tel precisa, primeiramente, adquirir uma referência para a fábrica do componente ou do agente móvel, respectivamente, *ComponentHome* ou *AgentHome*, por meio da interface *HomeFinder*. Enfatiza-se que um determinado tipo de componente ou agente móvel é especificado de forma independente da sua fábrica;

porém, em tempo de execução, uma instância de um componente é criada e gerenciada por um único objeto que implementa a interface *ComponentHome*. De forma semelhante, em tempo de execução, uma instância de um agente móvel é criada e gerenciada por um único objeto que implementa a interface *AgentHome*.

Em adição, o contêiner proporciona aos agentes móveis, como extensões, alguns mecanismos adicionais que consistem de interfaces utilizadas pelo desenvolvedor do agente móvel e que estabelecem o contrato entre este agente hospedado e o seu contêiner:

1. interface interna: interface local que é disponibilizada para que o agente hospedado possa acessar os serviços proporcionados pelo contêiner, descrita na seqüência;

2. interfaces externas do agente móvel: objetos que expõem interfaces externas dos agentes hospedados, disponíveis para os seus clientes. Estas interfaces estabelecem o contrato de uso entre o desenvolvedor dos agentes e os seus clientes;

3. interface callback: interface cujas operações de *callback* são implementadas pelos agentes móveis e que podem ser invocadas pela sua fábrica de agentes no contêiner, conforme descrito na seqüência;

4. infra-estrutura de acesso aos serviços de objeto: são serviços e funcionalidades que o contêiner interpõe entre clientes e agentes móveis. Esta infra-estrutura facilita o desenvolvimento de aplicações telemáticas e ubíquas ao agregar, além dos aspectos não-funcionais disponibilizados pelo contêiner CM-tel para componentes, os aspectos não-funcionais relativos a agentes móveis tais como o serviço responsável pela gerência e mobilidade destes agentes.

Modelo de Projeto para Interfaces Internas do Contêiner

O contêiner define quatro operações em sua interface interna para serem invocados pelos componentes e agentes móveis hospedados. Estas operações são:

1. A operação *get_registered_name* permite obter o nome com que o *HomeFinder* representando o contêiner foi registrado. Esta operação possui a seguinte definição em UML:

   ```
   get_registered_name() : String
   ```

2. A operação *get_home_finder* permite obter a referência do *HomeFinder*. Esta operação possui a seguinte definição em UML:

   ```
   get_home_finder() : HomeFinder
   ```

3. A operação *get_deployment_property* retorna o valor de uma propriedade de distibuição específica. Esta operação possui a seguinte definição em UML:

   ```
   get_deployment_property(name : String) : String
   ```

4. A operação *get_deployment_properties* retorna o conjunto de propriedades armazenadas na estrutura de propriedades de instalação. Esta operação possui a seguinte definição em UML:

   ```
   get_deployment_properties() : PropertySet
   ```

   As quatro operações acima descritas não lançam nenhuma exceção.

Instanciação dos Elementos do Contêiner CM-tel

O processo de instanciação dos elementos de um contêiner CM-tel é apresentado na Figura 3.32. A classe que implementa o contêiner instancia, inicialmente, a classe que implementa a interface Localizador de Fábricas (*HomeFinder*), especificada para este contêiner. A classe que implementa a interface de configuração do contêiner também é instanciada. A seguir, instancia as fábricas de componentes (*ComponentHomes*) para os tipos de componentes, especificados pelo contêiner. Da mesma forma, são instanciadas as fábricas de agentes móveis para os tipos de agentes móveis. Uma vez instanciada, cada fábrica é registrada na classe que implementa a interface *HomeFinder* por meio da operação interna *register_home*.

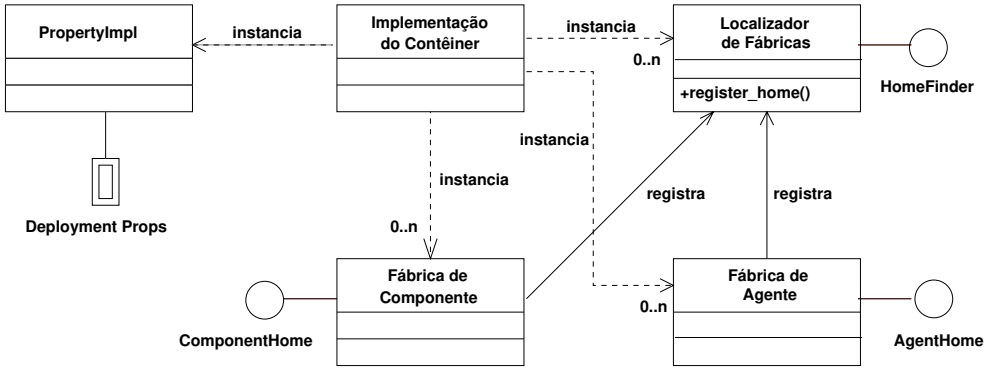

Fig. 3.32: Instanciação dos elementos do contêiner.

Fábrica de Agentes CM-tel (AgentHome)

A fábrica de agentes (*AgentHome*), ou agência, exporta uma interface que define operações de gerência do ciclo de vida de um agente móvel CM-tel tais como operações para criar, destruir, suspender, recomeçar, listar todos os agentes móveis hospedados em uma agência e migrar agentes

móveis para uma outra agência. A Figura 3.33 ilustra a interface *AgentHome*. O contêiner agrega a fábrica de agentes e os agentes móveis hospedados na agência. Contêineres CM-tel podem gerenciar os agentes móveis que hospeda através de agências.

De maneira semelhante à interface *ComponentHome*, a interface *AgentHome* é derivada da interface base *Home*, descrita na Seção 3.3.1. A classe *AgentHomeImpl* implementa a interface *AgentHome* e exporta operações para:

- instanciação de agentes móveis;
- migração de agentes para outra agência;
- acesso aos agentes gerenciados;
- controle de execução de agentes (suspensão e retomada de execução);
- destruição de agentes.

Em adição, o modelo proposto provê funcionalidades para a comunicação entre o contêiner e os agentes móveis (interfaces *Callback*). Estas interfaces são interfaces locais, implementadas pelo agente móvel, que podem ser invocadas pelo contêiner (por exemplo, para informar o agente móvel que o mesmo será destruído).

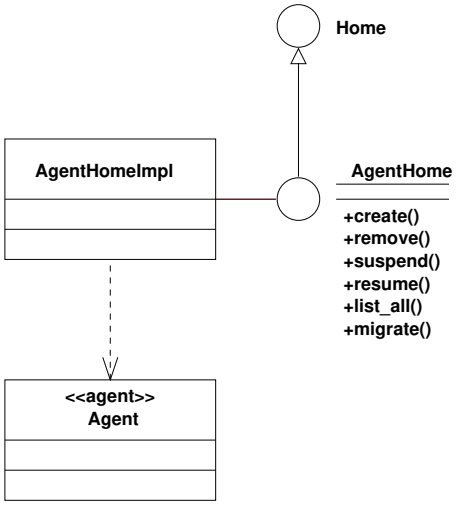

Fig. 3.33: Fábrica de agentes móveis (*AgentHome*).

A fábrica de agentes móveis define as seguintes operações:

1. A operação *create* permite criar, na agência, uma nova instância de um agente móvel. A operação recebe como parâmetro um nome para o agente móvel e retorna uma referência para o agente móvel criado. Esta operação possui a seguinte definição em UML:

```
create(name : string) : ObjRef
```

Esta operação lança a exceção *InvalidName*, caso o nome passado como parâmetro da operação viole as regras de atribuição de nomes de agentes.

2. A operação *remove* permite destruir uma instância de um agente móvel identificada pelo seu nome. Esta operação possui a seguinte definição em UML:

```
remove(name : string) : void
```

Esta operação lança a exceção *AgentNotFound*, caso não exista um agente móvel com o nome passado como parâmetro da operação.

3. A operação *suspend* permite suspender a execução de uma instância de um agente móvel identificado pelo seu nome. Esta operação possui a seguinte definição em UML:

```
suspend(name : string) : void
```

Esta operação lança a exceção *SuspendFailed*, caso não tenha sido possível suspender o agente móvel passado como parâmetro da operação.

4. A operação *resume* permite reiniciar a execução de uma instância de um agente móvel. Esta operação possui a seguinte definição em UML:

```
resume(name : string) : void
```

Esta operação lança a exceção *ResumeFailed*, caso não tenha sido possível reiniciar o agente móvel passado como parâmetro da operação.

5. A operação *list_all* permite obter uma lista de todos os agentes móveis. Esta operação possui a seguinte definição em UML:

```
list_all() : AgentList
```

Esta operação não lança nenhuma exceção.

6. A operação *migrate* permite solicitar a migração de um agente para uma outra agência. Esta operação possui a seguinte definição em UML:

```
migrate(nameagent : string, nameagency : string) : void
```

Esta operação lança a exceção *ArgumentInvalid*, caso o nome do agente móvel e/ou agência passados como parâmetro não sejam válidos.

As operações citadas acima são operações básicas para a gerência de agentes móveis e são providas por todas as implementações de sistemas de agentes móveis.

Modelo de Projeto de um Agente Móvel CM-tel

A Figura 3.34 ilustra as interfaces externas e as operações internas e de *callback*, que todo agente móvel deve disponibilizar para a sua agência. As operações definidas para o agente móvel CM-tel devem ser realizadas na classe que o implementa (*AgentImpl*).

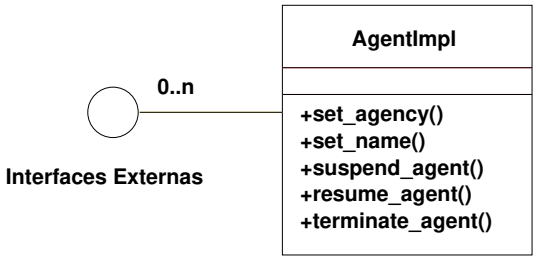

Fig. 3.34: Projeto para agentes móveis no modelo CM-tel.

As interfaces externas do agente móvel são disponibilizadas para os clientes do agente hospedado. A definição e implementação destas interfaces é de responsabilidade da aplicação.

A classe *AgentImpl* define as seguintes operações de *callback* implementadas pelo agente móvel, que podem ser invocadas pela fábrica de agentes:

1. A operação *set_agency* informa ao agente móvel a agência em que o mesmo foi criado e possui a seguinte definição em UML:

   ```
   set_agency(agency : AgentHome) : void
   ```

   O parâmetro *agency* fornece o nome da agência. A operação não lança nenhuma exceção.

2. A operação *set_name* atribui o nome definido na criação do agente móvel (operação *create*) e possui a seguinte definição em UML:

   ```
   set_name(name : String) : void
   ```

   O parâmetro *name* fornece o nome do agente móvel. A operação não lança nenhuma exceção.

3. A operação *suspend_agent* notifica ao agente móvel que o mesmo será suspenso e possui a seguinte definição em UML:

   ```
   suspend_agent() : void
   ```

   Esta operação lança as exceções *SuspendFailed*, caso não tenha sido possível suspender o agente móvel; e, *AgentIsSuspended*, caso o agente móvel já esteja suspenso.

4. A operação *resume_agent* notifica ao agente móvel que o mesmo teve a sua execução iniciada e possui a seguinte definição em UML:

   ```
   resume_agent() : void
   ```

   Esta operação lança as exceções *ResumeFailed*, caso não tenha sido possível iniciar a execução do agente móvel; e, *AgentIsRunning*, caso o agente móvel já esteja executando.

5. A operação *terminate_agent* notifica ao agente móvel que o mesmo teve a sua execução terminada e possui a seguinte definição em UML:

   ```
   terminate_agent() : void
   ```

   Esta operação lança a exceção *TerminateFailed*, caso não tenha sido possível terminar a execução deste agente.

Suporte aos Aspectos Não-funcionais

Plataformas aderentes ao modelo CM-tel devem prover suporte para os seguintes requisitos não-funcionais:

1. Facilidades de qualidade de serviço (QoS) para interfaces de fluxo contínuo. Qualidade de serviço deve ser tratada nos seguintes níveis:

   - componentes da aplicação: onde são especificados os parâmetros de QoS tais como taxa de amostragem de vídeo, formato de vídeo e tamanho da janela de apresentação de vídeo;
   - serviço de *streamming*: onde QoS é negociada entre portas produtoras e consumidoras de fluxo contínuo no momento da interconexão das portas;
   - rede: onde parâmetros de QoS são negociados com a camada de rede;
   - *hardware*: onde dispositivos como câmeras de vídeo e microfone são gerenciados.

2. Mobilidade de código. O suporte à mobilidade de código demanda por parte do contêiner serviços tais como localização, migração de agentes móveis, gerência de mobilidade e serviços relacionados a segurança. Estes serviços são importantes para o desenvolvimento de aplicações ubíquas com capacidade de adaptação ao ambiente (por exemplo, aplicações que empregam terminais sem fio).

3. Segurança. O nível de segurança propiciado pelo contêiner deve ser compatível com as tecnologias empregadas pelas plataformas de suporte ao modelo. Por exemplo, uma plataforma implementada com a tecnologia CORBA pode empregar o serviço de segurança do OMG (CORBAsecurity) para prover interação segura entre componentes.

4. Serviços de Objeto. O contêiner provê a infra-estrutura, que define como os componentes e agentes móveis de uma aplicação utilizam os serviços de objeto de forma totalmente transparente.

3.3.4 Especificação de Contêineres CM-tel em XML

Tal como componentes, os contêineres CM-tel são especificados no nível de projeto em XML. O processo de geração do *XML Schema* para validação de documentos XML especificando contêineres CM-tel é idêntico ao processo descrito na Seção 3.3.2 para componentes CM-tel. A representação UML do *XML Schema* para contêineres é apresentada na Figura 3.35. Esta representação foi gerada manualmente a partir do modelo de especificação apresentado na Figura 3.18. O *XML Schema* foi gerado automaticamente pela ferramenta HyperModel a partir do diagrama da Figura 3.35. O texto abaixo ilustra um fragmento do *XML Schema* que descreve o contêiner como um agregado de componentes e agentes.

```
<xs:element name="container" type="containerType"/>
 <xs:complexType name="containerType">
    <xs:sequence>
       <xs:element ref="components" minOccurs="0" maxOccurs="1"/>
       <xs:element ref="agents" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required"/>
 </xs:complexType>
```

A seguir são fornecidas, por meio de exemplos, as estruturas de documentos XML válidos para representar contêineres CM-tel.

Especificação XML para um contêiner mínimo CM-tel

Um contêiner mínimo CM-tel é ilustrado abaixo. Este contêiner possui apenas a especificação XML do tipo (classe) do contêiner. Esta informação é suprida pelo atributo *type*. O tipo representa a identificação deste contêiner. A especificação mínima de um contêiner não contém nenhum elemento a ser instanciado neste contêiner (tipo de componente ou de agência). Este contêiner pode ser criado, a partir desta especificação, como um ambiente de execução vazio.

```
<?xml version="1.0" ?>
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="cont.xsd"
    type = "CT_videoPlayer">
</container>
```

Fig. 3.35: Representação em UML de um *XML Schema* para a especificação de contêineres CM-tel.

Especificação XML para Componentes Hospedados no Contêiner

Os componentes que serão hospedados no contêiner são identificados pela marcação XML *components*. O elemento *componenttype* identifica o tipo do componente a ser instanciado e o elemento *dependency* é utilizado para especificar as dependências de contexto de execução deste componente em relação a outros. O suporte a estas dependências de contexto deve ser proporcionado pelo contêiner.

O exemplo abaixo ilustra a especificação XML de dois tipos de componentes (*C_videoPlayer* e *C_handoffMgr*). O componente *C_videoPlayer* especifica a sua dependência em relação ao componente *C_videoTransmitter*. Neste exemplo, o componente *C_handoffMgr* não possui nenhuma dependência de contexto.

```
<!-- componentes que serão instanciados neste contêiner -->
<components>
   <componenttype type = "C_videoPlayer">
      <dependency type = "C_videoTransmitter"/>
   </componenttype>
   <componenttype type = "C_handoffMgr"/>
</components>
```

Especificação XML para os Agentes Hospedados no Contêiner

Os agentes móveis que serão hospedados no contêiner são identificados pela marcação XML *agents*, sendo cada agente identificado pela marcação *agenttype*. Para cada tipo de agente é instanciada uma fábrica de agentes (agência) identificada pelo tipo do agente (atributo *type*). Estas agências proporcionam a gerência do ciclo de vida de agentes móveis.

O exemplo abaixo ilustra a especificação XML da agência identificada pelo nome global *MobilityMgmt*.

```
<!-- agentes neste contêiner -->
<agents>
  <agenttype name = "MobilityMgmt"/>
</agents>
```

3.4 Arquitetura de Software CM-tel

O modelo de projeto deve identificar uma arquitetura de *software* que suporte a instalação e execução de componentes e de agentes móveis CM-tel. Nós utilizamos o termo arquitetura de *software* para representar uma infra-estrutura de componentes de *software* associada a um conjunto de regras de projeto CM-tel.

Segundo Szyperski [15], uma arquitetura de componentes consiste de um conjunto de decisões de plataforma; um conjunto de *frameworks* de componentes; e, de um projeto de interoperação para estes *frameworks*.

O modelo CM-tel define uma arquitetura de *software* da plataforma independente de tecnologia e consistente com a definição de Szyperski, incorporando a este mais um elemento que é o *framework* de agentes. No padrão arquitetural CM-tel, os *frameworks* de componentes e de agentes são estruturados em camadas horizontais (*tiers*). Neste contexto, uma arquitetura com N camadas horizontais é denominada arquitetura *N-tier*. Diferentes camadas ocupam-se de diferentes graus de interação, com diferentes graus de especialidade. Os *frameworks* de componentes e de agentes CM-tel, localizados em determinada camada, integram os *frameworks* de componentes e de agentes (ou simplesmente, componentes e agentes móveis) localizados em camadas mais específicas. Para esta integração e interoperação, o modelo CM-tel disponibiliza recursos na camada mais genérica.

A Figura 3.36 ilustra uma arquitetura *3-tier*. A primeira camada consiste de um conjunto de componentes e de agentes móveis; a segunda consiste de *frameworks* de componentes e de agentes móveis responsáveis pela infra-estrutura para a execução de componentes e agentes móveis; e, a terceira consiste de um *framework* de integração, que fornece a infra-estrutura básica para a comunicação, interação e interconexão entre componentes e agentes móveis. Enfatizamos que este padrão

arquitetural é recorrente. Por exemplo, pode-se integrar diferentes arquiteturas *3-tier* por uma camada horizontal mais geral, obtendo-se uma arquitetura *4-tier*. O suporte de execução consiste no sistema de gerência de recursos como, por exemplo, sistema operacional ou máquina virtual Java.

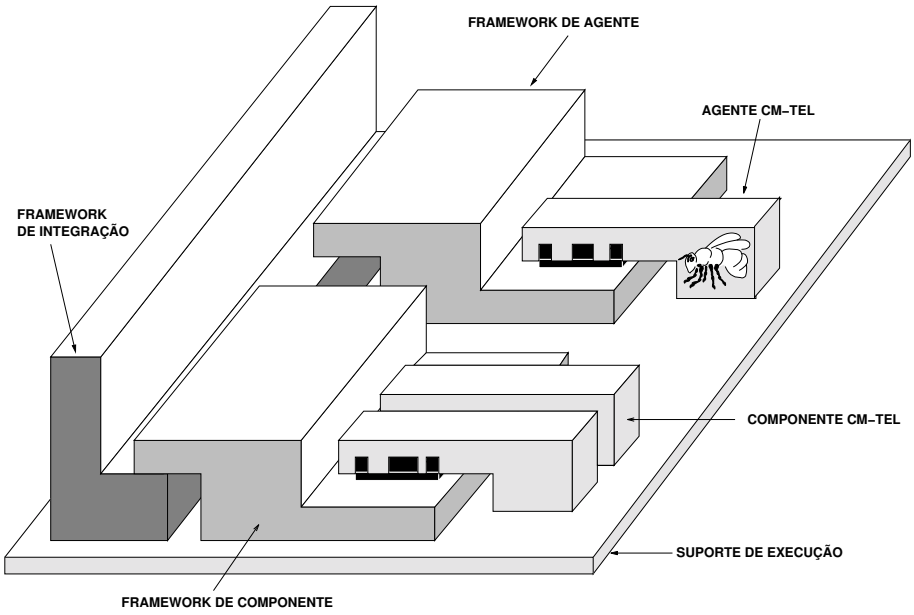

Fig. 3.36: Arquitetura da plataforma de suporte ao modelo CM-tel (arquitetura 3-tier).

3.5 Modelo de Implementação Física CM-tel

O modelo de implementação física CM-tel, conforme apresentado na Figura 3.1, é um modelo de referência totalmente independente de tecnologia para o desenvolvimento de plataformas de suporte ao desenvolvimento de *software*. Este modelo prescreve as linhas gerais para a geração de código, empacotamento e distribuição de componentes, agentes móveis e contêineres. Este modelo provê os elementos presentes na Figura 3.36.

O modelo de implementação física CM-tel, considera a geração de código como uma transformação executada por um processador de transformações, de acordo com uma definição de transformação expressa em uma linguagem apropriada. A definição de transformação descreve como um modelo em uma linguagem origem é transformado para gerar outro modelo em uma linguagem destino. O processo de transformação CM-tel é ilustrado pela Figura 3.37.

No modelo de implementação física CM-tel, ilustrado na Figura 3.38, as transformações ocorrem em duas fases: transformações especificação-projeto e transformações projeto-implementação física. A primeira fase consiste em transformar uma representação de componentes e contêineres do nível

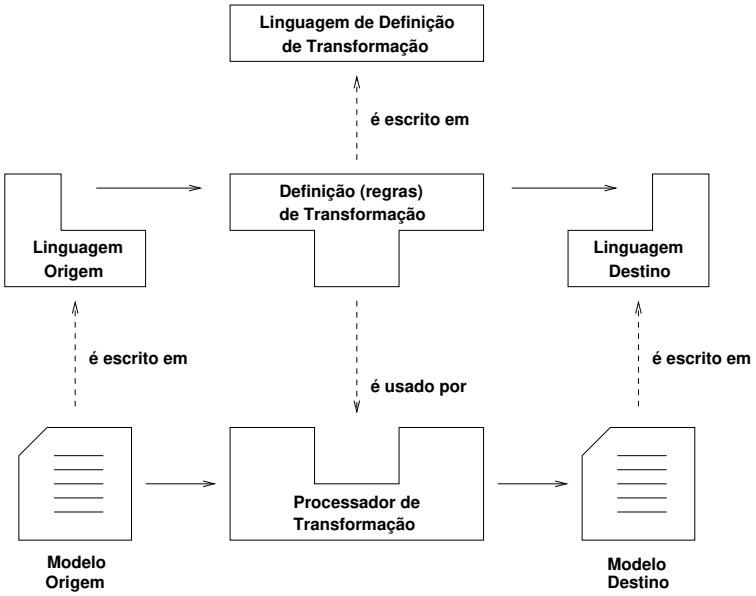

Fig. 3.37: Processo de transformação CM-tel.

de especificação para o nível de projeto, enquanto que a segunda fase transforma a especificação no nível de projeto para o nível de implementação física (código). O modelo CM-tel emprega um conjunto de gabaritos para cada fase de transformação. Estes gabaritos consistem de artefatos de *software* que representam elementos do modelo CM-tel e, como são a base para o processo de transformação, definem a estrutura do documento fonte a ser gerado. Um processador de transformações é responsável pela execução das transformações. Uma máquina de construção de código é responsável pela compilação do código fonte gerado e pelo empacotamento dos arquivos executáveis em um formato apropriado para a instalação do componente ou contêiner em um ambiente.

Este modelo, ao prover estas transformações, torna o desenvolvimento de aplicações telemáticas e ubíquas mais rápido e eficiente, além de reduzir significativamente a complexidade destes desenvolvimentos. A chave deste processo é desobrigar os programadores da aplicação de conhecer as complexidades e os aspectos técnicos da distribuição associados à tecnologia escolhida. Em adição, melhora a qualidade ao utilizar gabaritos com regras de projeto e com padrões de código experimentados, testados e menos propensos a erros. Como resultado, tem-se uma geração de sistemas de *software* robustos e de fácil evolução. Outra vantagem é a facilidade para a evolução (extensão) destes gabaritos de código, por exemplo, para tirar proveito das novas características introduzidas em uma determinada tecnologia. As duas transformações são descritas a seguir.

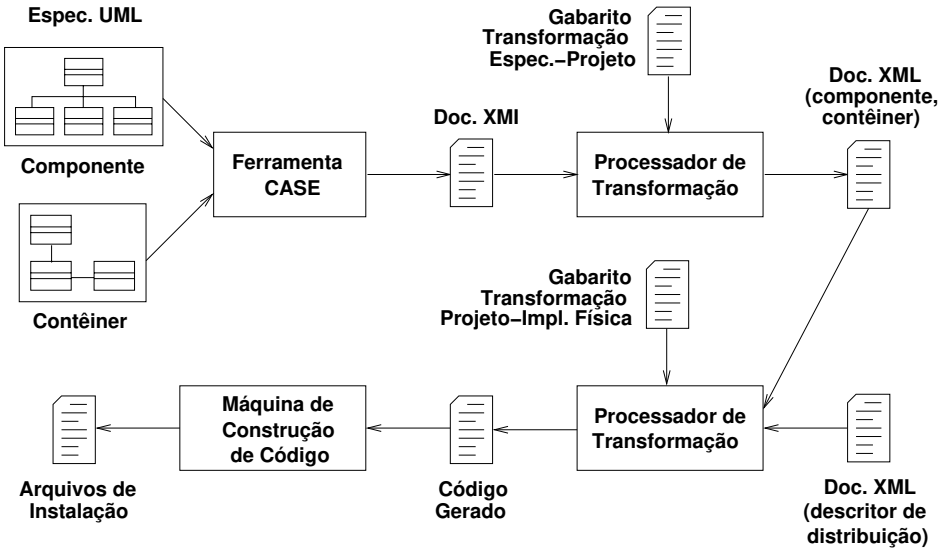

Fig. 3.38: Modelo de implementação física CM-tel.

Transformações Especificação-Projeto

As transformações especificação-projeto têm como entrada uma representação de componentes e contêineres no nível de especificação (UML) e como saída a representação destes artefatos de *software* no nível de projeto (XML). Em termos de notação, temos uma transformação UML-XML.

Os artefatos em UML são transformados em XMI com o auxílio de uma ferramenta CASE (*Computer-Aided Software Engineering*). O metamodelo XMI do OMG é utilizado para a descrição de artefatos UML em XML. Os gabaritos desta fase especificam as regras de transformação XMI-XML, e consistem fundamentalmente do mapeamento de marcações XMI em marcações XML (definidos pelo modelo CM-tel), para cada elemento de componente e contêiner.

É importante notar que estas transformações são independentes da tecnologia alvo, escolhida para suportar o modelo CM-tel.

Transformações Projeto-Implementação Física

As transformações projeto-implementação física têm como entrada uma representação XML de componentes, contêineres e, adicionalmente descritores de distribuição (descritos na Seção 3.5.2), e como saída a implementação destes artefatos de *software* (*frameworks* de componentes e agentes, contêineres e arquivos de instalação) em formatos do código fonte escolhido e regras de construção. Temos, portanto, uma transformação de especificações XML para formatos pertinentes à tecnologia específica. Os gabaritos desta fase especificam as regras de transformação projeto-implementação e são a base para a geração automática de código.

3.5.1 Empacotamento

A especificação de empacotamento CM-tel define como os arquivos produzidos pela máquina de geração de código serão compilados, ligados e empacotados. Esta especificação se dá por um conjunto de regras de construção compatíveis com uma máquina de construção. Esta máquina é análoga à ferramenta *make* disponibilizada em vários sistemas operacionais. A máquina de construção recebe como entrada os arquivos fonte gerados pelas transformações projeto-implementação física, processando-os de acordo com regras de construção. Tais regras especificam o compilador e empacotador a serem utilizados, os parâmetros passados na execução destes aplicativos, os diretórios de destino, etc.

3.5.2 Distribuição

Os contêineres são instalados e configurados em um conjunto de computadores de acordo com especificações de alto nível em XML proporcionadas pelos descritores de distribuição. Isto é necessário porque o contêiner fornece os aspectos técnicos de uma forma flexível e genérica para que sejam aplicáveis para todos os componentes e agentes hospedados. Os descritores especificam as características e os atributos necessários para personalizar o contêiner e seus componentes durante a sua instalação. Entre outros, ele pode conter parâmetros de instalação (por exemplo, localização dos arquivos em formato JAR para a instalação de componentes), políticas de segurança, propriedades iniciais de componentes e de contêineres, um modelo particular de *threading*, etc. Em adição, por meio das informações proporcionadas pelo descritor, um contêiner integra os aspectos funcionais proporcionados pelo componente com os aspectos técnicos requeridos. Por exemplo, a utilização de determinado apresentador de vídeo (JMF[5] ou *QuickTime*) disponível na infra-estrutura de serviços de objeto.

Um atributo importante para serviços telemáticos e independente de tecnologia consiste nos parâmetros de qualidade de serviço (QoS) para as interfaces de fluxo contínuo de um componente. Por ser um aspecto não-funcional, os parâmetros de QoS são especificados pelo descritor de distribuição, que por sua vez dita a configuração específica para um determinado contêiner. Outro atributo de configuração é o nome atribuído ao localizador de fábrica (*HomeFinder*) do contêiner instanciado. Finalmente, parâmetros específicos a serem disponibilizados para uma aplicação (na forma nome-valor) também podem ser especificados no descritor de distribuição.

Um *XML Schema* foi definido para a especificação de descritores de distribuição para contêineres CM-tel. Este *XML Schema* é apresentado por um diagrama UML na Figura 3.39. Os parâmetros de qualidade de serviço, dados pelo elemento *QoS*, foram definidos de acordo com a especificação

[5] Java Media Framework [104].

AVStreams [23] e apresentados no trecho abaixo do *XML Schema* que define descritores de distribuição CM-tel.

```
 <xs:element name="QoS" type="QoSType"/>
  <xs:complexType name="QoSType">
   <xs:attribute name="video_framerate" type="xs:unsignedByte" default="10"/>
   <xs:attribute name="video_colorDepth" type="xs:unsignedByte" default="24"/>
   <xs:attribute name="video_colorModel" type="colorTypes" default="1"/>
   <xs:attribute name="video_resolution" default="160X120">
   <xs:attribute name="video_quality" type="xs:float" default="0.5"/>
      <xs:simpleType>
        <xs:restriction base="xs:string">
           <xs:pattern value="\d{2,4}X\d{2,4}"/>
        </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="audio_sampleRate" type="rateTypes" default="8000"/>
   <xs:attribute name="audio_sampleSize" type="sizeTypes" default="8"/>
   <xs:attribute name="audio_numChannels" type="channelTypes" default="1"/>
   <xs:attribute name="audio_quantisation" type="quantType" default="1"/>
  </xs:complexType>
```

O exemplo abaixo ilustra a especificação em XML de um descritor de distribuição. Neste exemplo, o contêiner "CT_videoPanel" tem a sua interface *HomeFinder* identificada pelo nome "VideoPanelContainer" e os arquivos de instalação serão gerados com o nome "vp". O descritor de distribuição estipula cinco parâmetros de qualidade de serviço para as portas produtoras de vídeo dos componentes instalados no contêiner.

```
<?xml version="1.0" ?>
<!-- descritor de distribuição -->
<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/CCMBuilder/schema/depl.xsd"
  class = "CT_videoPanel"
  file = "vp">
<HomeFinder
  HomeFinderName = "VideoPanelContainer"/>
<StreamPorts>
   <QoS
      video_framerate = "25"
      video_colourDepth = "24"
      video_colourModel = "1"
      video_resolution = "160X120"
```

```
      video_quality ="0.5"/>
</StreamPorts>
</deployment>
```

A máquina de transformações, que automatiza a geração de código, processa um descritor de distribuição e gera um ou mais arquivos necessários à instalação do contêiner especificado. Por exemplo, os formatos típicos para tais arquivos são HTML para contêineres instalados como *applets* Java em navegadores Web, *shell scripts* para contêineres instalados como processos de sistema operacional ou *midlets* para contêineres instalados em dispositivos móveis com baixo poder de processamento.

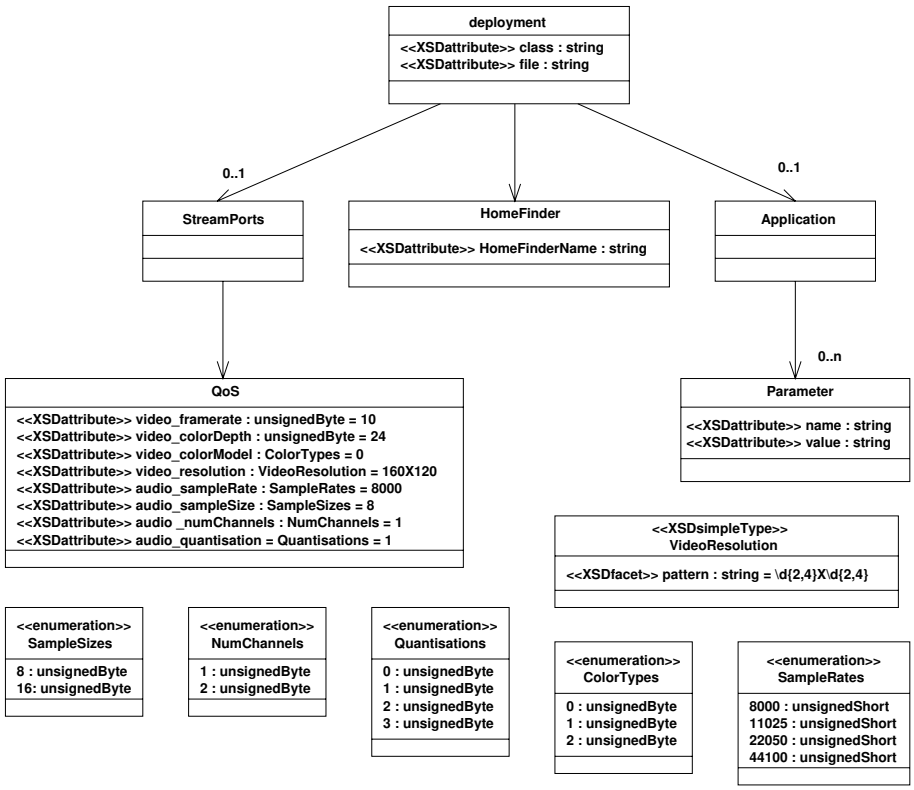

Fig. 3.39: Representação em UML de *XML Schema* para a especificação de distribuição de componentes CM-tel.

3.6 Considerações Finais

O modelo de componentes CM-tel proposto neste capítulo fornece soluções aos requisitos impostos pelas aplicações telemáticas e ubíquas, requisitos estes não encontrados nos modelos de componentes existentes. CM-tel é um modelo de componentes neutro para o desenvolvimento destas

aplicações emergentes com suporte nativo à comunicação multimídia distribuída com facilidades de qualidade de serviço. Em adição, contêineres CM-tel fornecem suporte à adaptação dinâmica das aplicações e ubiqüidade por meio de código móvel e de agentes móveis. Essencialmente, a nossa proposta consiste em especificar um modelo de componentes em dois níveis de abstração. O primeiro nível, apresentado neste capítulo, compreende um modelo neutro em termos de tecnologia (independente de plataformas, linguagens de programação, sistemas operacionais, protocolos de rede, etc). Neste nível, o componente é especificado em UML ou XML. No segundo nível, plataformas de suporte ao modelo são construídas de acordo com o modelo de implementação física baseado em transformações de documentos XML. A partir da especificação do componente, tais transformações geram código para suporte ao componente em uma determinada tecnologia. Algumas extensões relativas à tecnologia escolhida são necessárias aos contêineres e aos descritores de distribuição. Uma das plataformas de suporte ao modelo CM-tel empregando a tecnologia CORBA é apresentada no próximo capítulo.

Em uma avaliação inicial do modelo CM-tel constatam-se alguns pontos de relevância. Em particular, podemos destacar que é possível a dois mapeamentos diferentes do modelo interoperarem, caso as tecnologias alvo sejam capazes de interoperar. Por exemplo, um mapeamento para a tecnologia CORBA é capaz de interoperar com um mapeamento para a tecnologia Java RMI, satisfeitas as restrições impostas pela operação de RMI sobre IIOP.

As regras de um mapeamento podem ainda especificar uma extensão de um modelo de componentes existente. Por exemplo, uma extensão do modelo de componentes EJB pode adicionar a este modelo interfaces de fluxo contínuo de áudio e vídeo, visando o desenvolvimento de aplicações multimídia distribuídas.

Por ser um modelo simplificado, plataformas CM-tel podem ser implementadas sobre infra-estruturas mínimas tais como aquelas oferecidas por *handhelds* e telefones celulares (o que seria impossível para modelos "pesados" como EJB ou CCM).

Seguindo os passos utilizados durante a concepção do modelo de componentes CM-tel, é possível projetar novos modelos de componentes de domínios específicos, visando atender domínios de aplicações específicos.

Finalmente, constata-se que o modelo proposto apresenta soluções para as deficiências apresentadas no início deste capítulo. Entre estas soluções destacam-se:

1. O modelo CM-tel é desvinculado de tecnologia específica;

2. O modelo CM-tel não se restringe a aplicações comerciais graças a presença neste modelo da interface de fluxo contínuo (*stream*);

3. Os requisitos não-funcionais de qualidade de serviço, de mobilidade de código, de segurança

e de serviços de objetos são automaticamente incorporados às plataformas que implementam o modelo CM-tel, conforme apresentado nos Capítulos 4 e 5;

4. O modelo CM-tel utiliza agentes móveis, de forma integrada com componentes, como base para adaptação dinâmica de aplicações (adaptação dinâmica também não é contemplada pelos modelos de componentes existentes);

5. O modelo CM-tel é leve o suficiente para a instalação em dispositivos móveis sem fio.

Capítulo 4

Plataforma CCM-tel

O modelo de componentes CM-tel é um modelo neutro, desvinculado de qualquer tecnologia, conforme descrito no Capítulo 3. A principal vantagem proporcionada por este modelo é possibilitar a construção de plataformas de suporte ao modelo CM-tel em diferentes tecnologias alvo (CORBA, Java RMI, COM+, EJB, entre outras), conforme ilustrado na Figura 4.1. Entretanto, CM-tel exige a definição de mapeamentos do modelo para cada uma destas tecnologias. Estes mapeamentos acrescentam certas particularidades da tecnologia às especificações do modelo CM-tel. Para cada plataforma a ser construída existe um conjunto de mapeamentos específicos que incorporam as características da tecnologia escolhida.

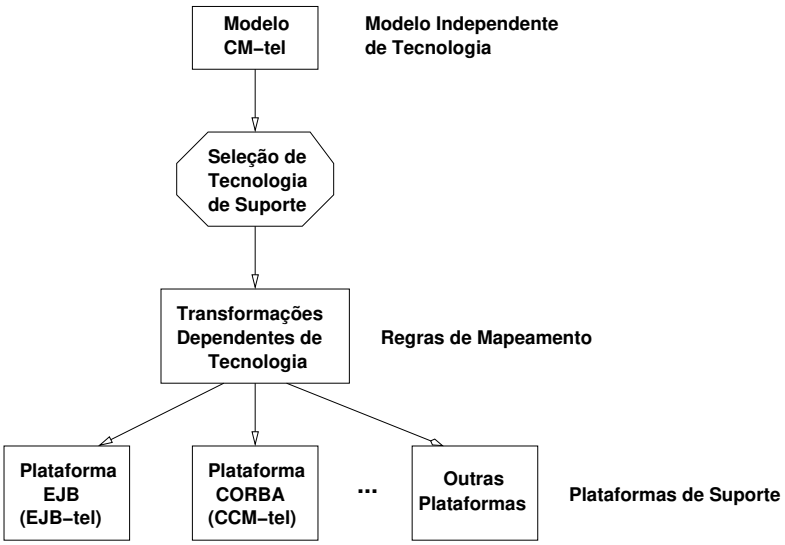

Fig. 4.1: Plataformas de suporte ao modelo CM-tel.

Neste capítulo, apresentamos os aspectos relacionados à construção de uma plataforma de suporte

ao modelo CM-tel. Nas próximas seções, abordaremos os detalhes de implementação das soluções propostas para atender aos requisitos deste modelo relacionados ao seu mapeamento para as tecnologias escolhidas (CORBA e Java), bem como infra-estruturas de *middleware* necessárias. Esta plataforma, denominada CCM-tel, tem como objetivo suportar o desenvolvimento e a distribuição de aplicações utilizando o modelo de componentes proposto.

4.1 A Opção pelas Tecnologias CORBA/Java

Ao examinarmos as possíveis tecnologias a serem utilizadas na implementação da plataforma de suporte ao modelo CM-tel, consideramos aquelas cujas especificações contemplavam aspectos compatíveis com os princípios apresentados por este modelo e que pudessem ser implementadas com relativa facilidade e eficiência. A escolha da tecnologia CORBA, da linguagem de programação e da plataforma Java como um primeiro protótipo para avaliar a adequação da nossa proposta, baseou-se nos pontos descritos a seguir:

- a tecnologia CORBA é uma tecnologia que tem uma especificação aberta e neutra (independente de linguagem de programação, sistema operacional e plataformas de *hardware*), o que é coerente com o modelo CM-tel;
- uma implementação do padrão CORBA está incorporada na plataforma Java;
- CORBA disponibiliza interceptadores portáveis [3] um aspecto importante para prover os requisitos não-funcionais tais como qualidade de serviço, mobilidade de código e segurança;
- disponibilidade de implementações de serviços especificados pelo OMG para a arquitetura CORBA (nomes, segurança, transações, etc);
- a maturidade dos protocolos associados ao padrão CORBA possibilita a interoperabilidade (IIOP) e a portabilidade de implementações (POA) entre ORBs;
- a linguagem Java é independente de plataforma e disponível em uma vasta gama de dispositivos, tais como terminais móveis e navegadores Internet;
- a linguagem Java suporta carregamento de classes sob demanda e serialização de objetos, requisitos importantes para a instalação flexível de aplicações e mobilidade de código.

Como motivações adicionais da adequação das tecnologias CORBA/Java para o desenvolvimento da plataforma CCM-tel citamos:

- disponibilidade da implementação em Java das seguintes especificações de serviços CORBA: propriedades, eventos, gerência de fluxos contínuos (A/V Streams) e agentes móveis (MAF - *Mobile Agent Facility*) [50];
- a experiência acumulada no desenvolvimento de aplicações distribuídas em Java que utilizaram as implementações dos serviços CORBA citados acima, tais como ferramenta de educação a distância [105], aplicações multimídia [106] e sessão de acesso TINA [33];
- projeto e implementação da versão inicial do laboratório de acesso remoto REAL [39] que utiliza estas tecnologias.

4.2 Um Mapeamento do Modelo CM-tel para CORBA

Mapeamentos do modelo CM-tel para uma dada tecnologia alvo produzem plataformas de suporte específicas para esta tecnologia. Estes mapeamentos possuem a vantagem de preservar a estrutura e a especificação (UML e XML) dos componentes e dos contêineres CM-tel. As particularidades da tecnologia alvo ficam circunscritas a dois elementos CM-tel: contêineres e descritores de distribuição. Entretanto, no caso de contêineres estas particularidades são geradas automaticamente pela plataforma, enquanto que para descritores de distribuição alguns novos elementos são definidos e necessitam ser especificados de forma declarativa em XML. Esta propriedade traz um grande benefício em termos de desenvolvimento de aplicações, pois permite a especificação dos componentes e contêineres da aplicação de forma independente de tecnologia, o que não ocorre com os modelos de componentes existentes como CCM e EJB.

Esta seção apresenta um mapeamento do modelo CM-tel para CORBA e para a linguagem de programação Java, disponibilizado em uma plataforma denominada CCM-tel (mapeamento "CORBA CM-tel"). Este mapeamento compreende uma arquitetura de contêineres e extensões para descritores de distribuição relacionados às tecnologias CORBA e Java. Em adição, define as transformações do modelo de implementação física CM-tel, que constituem a base para a geração automática de código para estas tecnologias. Estas definições e transformações são descritas na seqüência.

4.2.1 Arquitetura de Contêineres CCM-tel

O mapeamento CCM-tel define os elementos específicos da tecnologia CORBA para a arquitetura dos contêineres CM-tel, descritos na Seção 3.3.3. Estes elementos são: ORB, o adaptador de objeto portável (POA), interceptadores portáveis e os serviços e facilidades CORBA. A Figura 4.2 ilustra estes elementos na arquitetura do contêiner CCM-tel. A seguir é apresentada uma breve descrição destes elementos:

- O intermediador de requisições (*ORB - Object Request Broker*), que é uma infra-estrutura de interconexão aderente à especificação CORBA e tem como papel principal prover um canal de comunicação entre objetos distribuídos, de maneira transparente em relação à distribuição física e à heterogeneidade do sistema. O ORB provê o suporte à comunicação entre clientes e servidores mediando o transporte através da rede de requisições e respostas.

- Adaptador de Objetos Portáveis (POA), que é um padrão definido na especificação CORBA de forma a facilitar a portabilidade de implementações de objetos servidores entre diferentes ORBs. POA consiste de um conjunto de funções para a gerência de objetos serventes. Por meio do POA, os contêineres CCM-tel podem estabelecer um conjunto de políticas que governam aspectos tais como a ativação de objetos (por exemplo, ativação sob demanda, compartilhamento de *threads* e persistência), bem como a criação de referências de objetos. Desta forma, POA possibilita uma maior flexibilidade de configuração no comportamento dos componentes gerenciados pelo contêiner.

- Interceptadores portáveis CORBA, constituem um mecanismo natural para realizar extensões à plataforma, porque tornam-se parte do *ORB* após a sua inicialização. Este mecanismo possibilita a interceptação do fluxo de comunicação do ORB, em pontos bem definidos, podendo alterar a forma como as requisições são conduzidas. Os interceptadores realizam a interceptação das requisições dos clientes e das respostas durante as interações entre componentes, inspecionam as informações obtidas e incluem comportamentos ou serviços internos adicionais ao contêiner para a sua manipulação. Entre as possíveis extensões realizadas por interceptadores para suportar novos aspectos não-funcionais destacam-se as funções de gerência de recursos, de segurança e de transações.

- Serviços e facilidades CORBA que incluem os serviços básicos para suporte aos componentes e agentes móveis: serviço de nomes, serviço de gerência de fluxos de áudio/vídeo (A/V Streams), serviço de propriedades, serviço de notificação de eventos e a facilidade de agentes móveis (MAF).

Os novos elementos do contêiner CCM-tel não necessitam ser especificados ou implementados. Estes elementos são fornecidos pela ferramenta de geração de código da plataforma e agregados aos elementos básicos do contêiner CM-tel. Assim sendo, o *XML Schema* para validação de documentos XML especificando contêineres CCM-tel é o mesmo *XML Schema* para contêineres CM-tel (Figura 3.35).

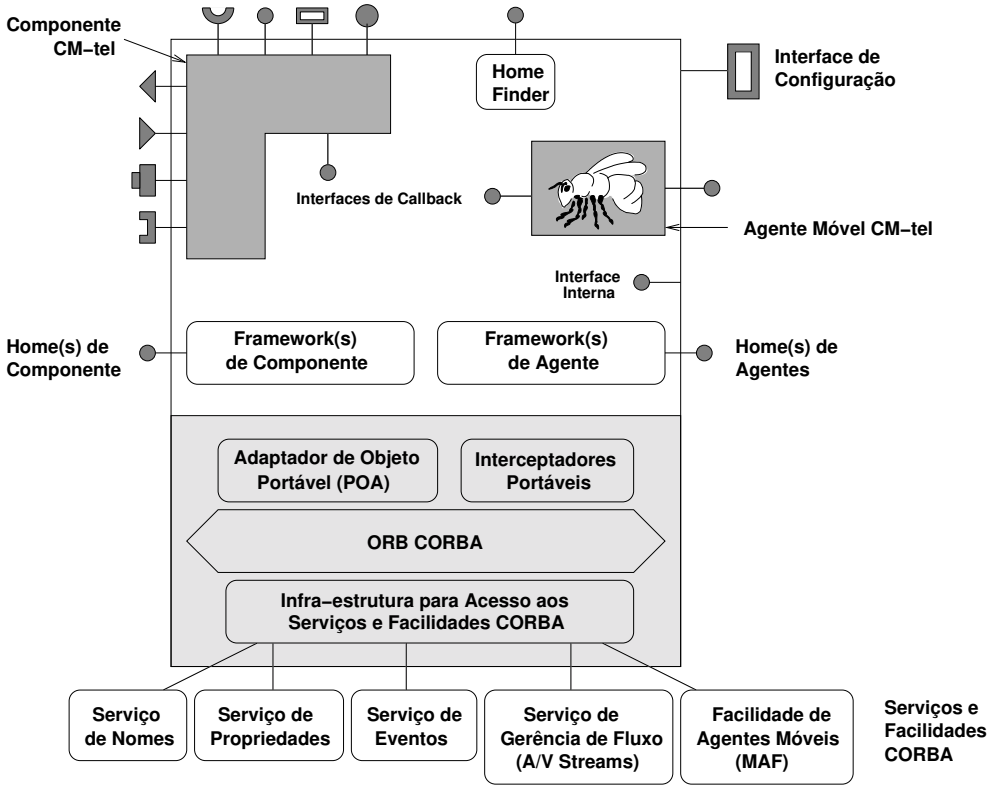

Fig. 4.2: Arquitetura de um contêiner CCM-tel.

4.2.2 Extensões para Descritores de Distribuição CCM-tel

O mapeamento CCM-tel define novos elementos de extensão específicos da tecnologia escolhida para os descritores de distribuição CCM-tel, conforme descrito na Seção 3.5.2. A partir destes descritores de distribuição, os contêineres CCM-tel são distribuídos e configurados. O processo de distribuição compreende a fase de instalação, instanciação e posterior configuração do contêiner. A etapa de configuração permite ainda um outro nível de configuração da aplicação, que consiste em adaptar o contêiner para um determinado ambiente de execução pela especificação de políticas para determinados requisitos. O descritor de distribuição especifica em XML as políticas que determinam a forma como o contêiner irá gerenciar certos requisitos não-funcionais como, por exemplo, políticas de QoS para fluxos de mídia contínua. Estes elementos são atributos especificados em XML para incorporar os seguintes parâmetros:

- Parâmetros relacionados ao ORB. Estes parâmetros permitem especificar uma implementação de ORB, suprir parâmetros de inicialização do ORB (por exemplo, endereço de rede do serviço de nomes), definir políticas do POA e agregar interceptadores portáveis ao ORB.

- Parâmetros de instalação de *applets* Java. Especificam a classe que implementa o contêiner na forma de *applet* Java, as dimensões do painel ocupado pelo *applet*, localização do servidor HTTP e diretório neste servidor onde se encontram as classes necessárias à execução do contêiner (*frameworks* de componentes e agentes, serviços CORBA, etc.).

- Parâmetros relacionados às portas de fluxo contínuo. Especificam os dispositivos de captura, apresentação e difusão de mídia contínua. Por exemplo, pode-se especificar capturadores e apresentadores baseados em Java Media Framework (JMF) ou QuickTime, os parâmetros de qualidade de serviço para as portas de fluxo e o identificador de contrato com a rede[1] que permite priorizar o tráfego de mídia contínua em redes DiffServ.

- Parâmetros de localização (URLs) de recursos. Especificam a localização de servidores de nomes e servidores HTTP capazes de suprir as classes necessárias à execução dos agentes móveis no contêiner.

O processo de geração do *XML Schema* para validação de documentos XML especificando a distribuição de contêineres CCM-tel estende a representação UML do *XML Schema* definido pelo modelo de distribuição CM-tel (Figura 3.39). A representação UML do *XML Schema* para descritores de distribuição CCM-tel é apresentada nas Figuras 4.3 e 4.4. Os elementos sombreados nestas figuras são as extensões dependentes de tecnologia. A partir deste diagrama UML foi gerado pela ferramenta *HyperModel* o *XML Schema* correspondente.

Cabe ressaltar que um mecanismo flexível de distribuição, aliado à capacidade de manipular código ou agentes móveis, viabiliza a utilização de contêineres adaptáveis que permitem tanto a adição dos aspectos da tecnologia escolhida, como a adição estática ou dinâmica de suporte a novos aspectos não-funcionais a serem gerenciadas por ele. Esta característica proporciona uma maior flexibilidade às aplicações emergentes quanto a novos requisitos; por exemplo, a adequação a aspectos de qualidade de serviço impostos por ambientes com recursos limitados.

O exemplo a seguir ilustra uma especificação completa em XML para um descritor de distribuição CCM-tel, incorporando a declaração dos novos elementos aos anteriormente apresentados na Seção 3.5.2.

```
<?xml version="1.0" ?>
<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="depl.xsd"
  class = "videoPlayerServer"
  file = "vp">

<HomeFinder
```

[1] SLA - Service Level Agreement.

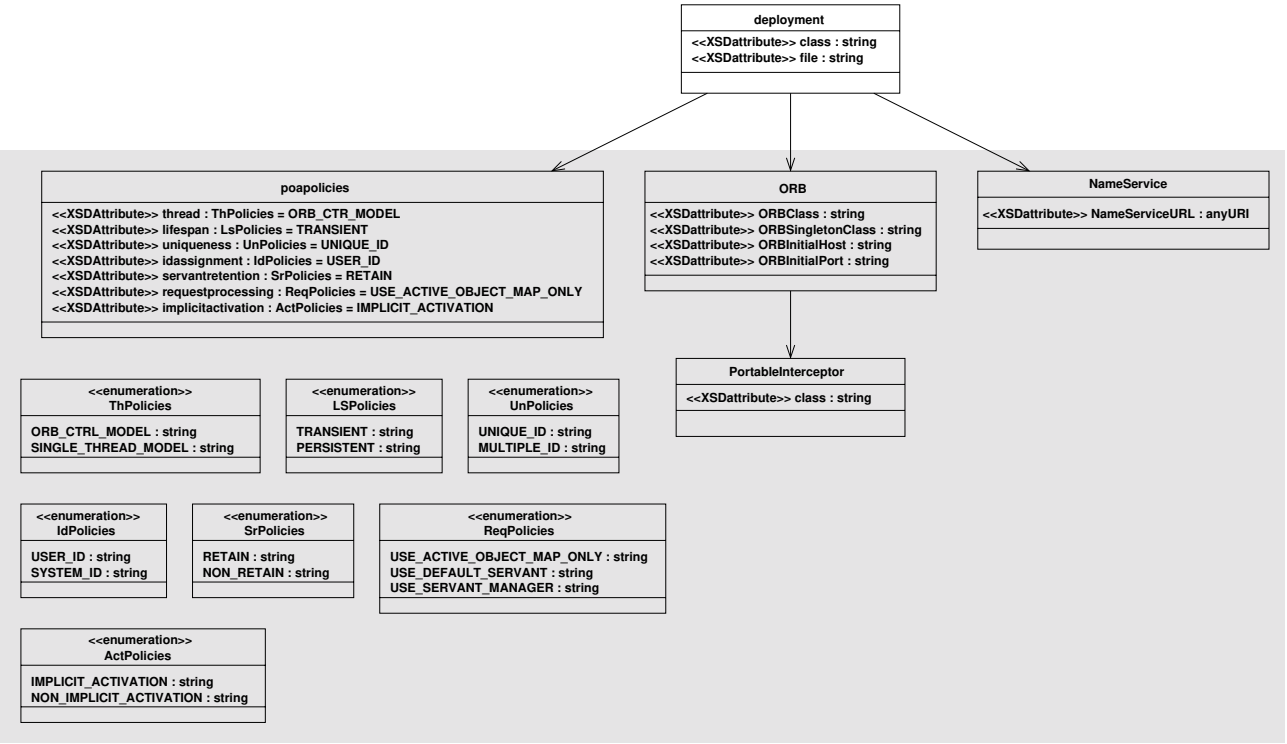

Fig. 4.3: Representação em UML de XML Schema para a especificação de distribuição CCM-tel (parâmetros do ORB).

```
  HomeFinderName = "videoPlayerAgent"/>
<!-- ORB -->
<ORB
    org.omg.CORBA.ORBInitialHost = "formoso.p4.cenpra.gov.br"
    org.omg.CORBA.ORBInitialPort = "6688">
  <PortableInterceptor class = "IQoS"/>
</ORB>
<!-- Políticas do POA -->
<poapolicies
      thread = "ORB_CTRL_MODEL"/>
<!-- Localização do serviço de nomes -->
<NameService
  NameServiceURL = "http://www.dca.fee.unicamp.br:8080"/>
<!-- Parâmetros relativos as portas de fluxo -->
<StreamPorts>
   <QoS
      video_framerate = "25"
      video_colorDepth = "24"
      video_colorModel = "1"
```

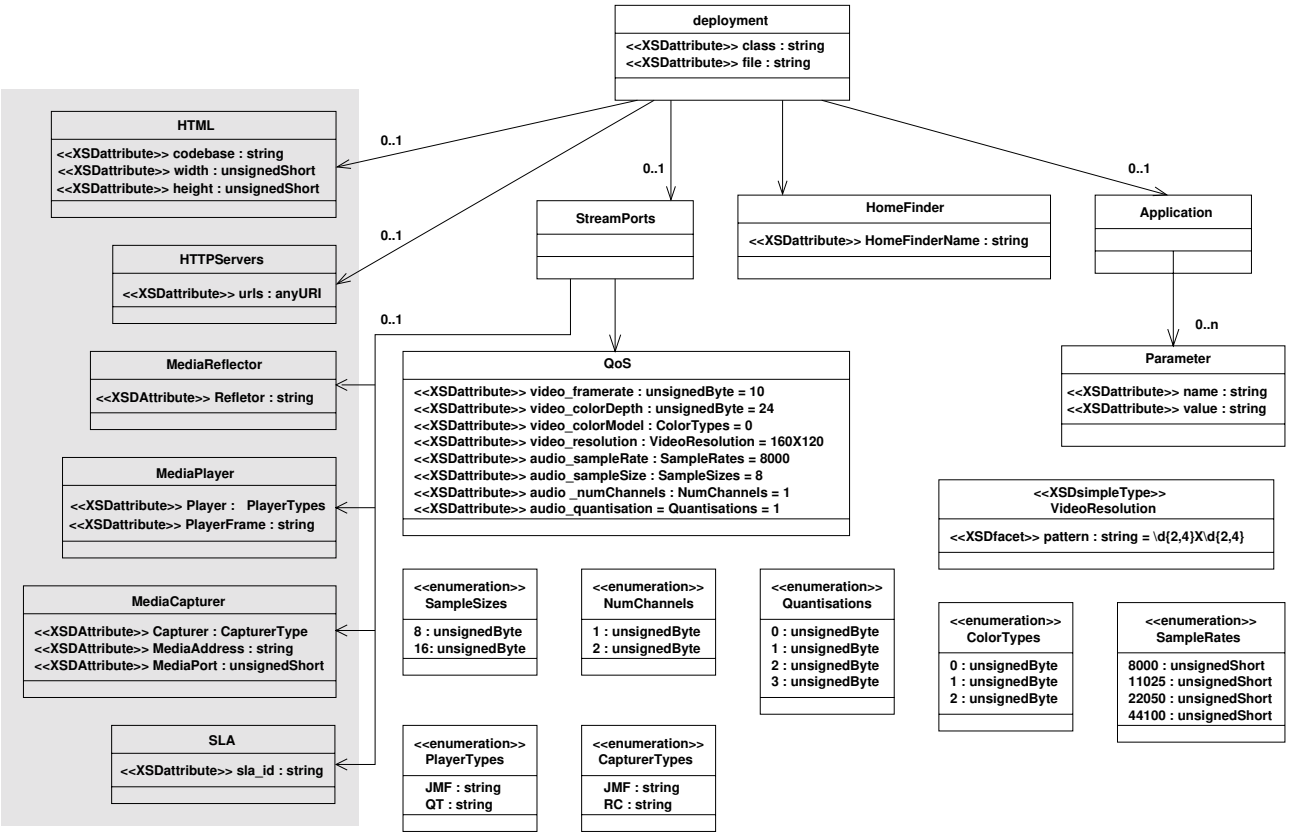

Fig. 4.4: Representação em UML de XML Schema para a especificação de distribuição CCM-tel (demais parâmetros).

```
      video_resolution = "160X120"
      video_quality = "0.5"
   <SLA
      broquer = "DSBBroker">
      sla_id = "qwerty"/>
</StreamPorts>
<!-- Parâmetros especificos da aplicacao -->
<Application>
  <Parameter name = "RobotURL"  value = "atria:8002"/>
</Application>
</deployment>
```

4.2.3 Transformações CCM-tel

Transformações são a base do modelo de implementação física CM-tel descrito na Seção 3.5.

A plataforma CCM-tel emprega definições de transformação especificadas na linguagem XSLT

(*Extensible Stylesheet Language Transformation*) [31]. A Figura 4.5 ilustra o processo de transformação, apresentado no capítulo anterior pela Figura 3.37, agora ilustrando a solução tecnológica escolhida.

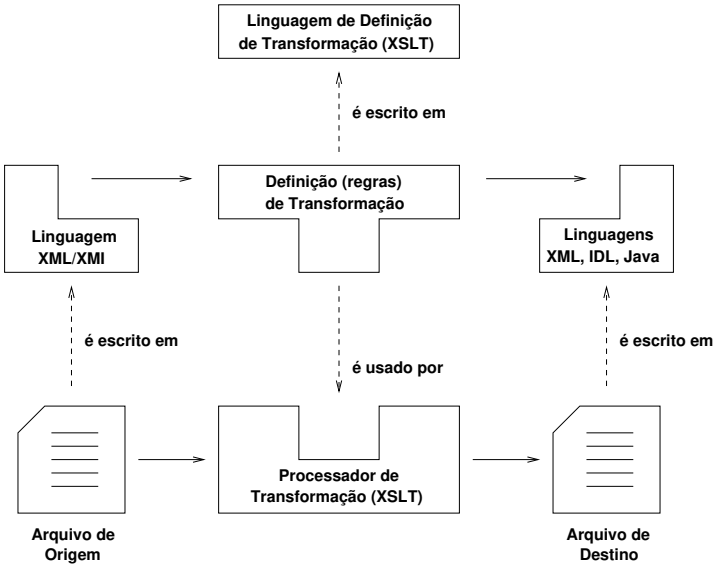

Fig. 4.5: Processo de transformação CCM-tel.

XSLT é um vocabulário XML que especifica um formato (estilo) de apresentação para documentos XML. Esta especificação de formato é denominada "folha de estilo" (*stylesheet*). Folhas de estilo especificam as regras de transformação de documentos XML. Por exemplo, folhas de estilo são utilizadas para especificar como um documento XML é formatado em HTML para apresentação em navegadores Web. Estas folhas de estilo correspondem aos gabaritos, descritos na Seção 3.5, agora realizados utilizando as tecnologias escolhidas.

A linguagem XSLT define um conjunto de regras de transformação (*templates*) que são inseridos nas folhas de estilo formando um esqueleto de documento no formato alvo. Estas regras de transformação permitem extrair informações de documentos XML (por exemplo, atributos e valores de marcações), testar a presença de determinada marcação, atributo ou valor, percorrer o documento fonte, e efetuar determinado processamento (por exemplo, somar dois números). Uma ferramenta de transformação (ou processador de transformação XSLT) processa as regras de transformação, substituindo o resultado do processamento na sua correspondente posição na folha de estilo, gerando assim um novo documento. A plataforma CCM-tel emprega um conjunto de folhas de estilo para cada fase de transformação.

Transformações Especificação-Projeto

As transformações especificação-projeto, descritas na Seção 3.5, utilizam folhas de estilo e regras de transformação XSLT para transformar a especificação UML de componentes e contêineres descritos em XMI (via uma ferramenta CASE) em documentos XML correspondentes, definidos pelo modelo CM-tel.

Transformações Projeto-Implementação Física

As transformações projeto-implementação física, descritas na Seção 3.5, utilizam folhas de estilo e regras de transformação XSLT para transformar uma representação XML de componentes, contêineres e descritores de distribuição em arquivos contendo a implementação destes artefatos de *software* em formatos de código fonte (Java, IDL) e de construção (XML).

O processo de geração automática de código utilizando folhas de estilo XSLT isenta o desenvolvedor dos seguintes detalhes:

- Manipulação dos serviços CORBA. Os serviços CORBA são utilizados pelo *framework* de componentes e de agentes e pelo contêiner, e não invocados diretamente pela aplicação. Por exemplo:
  - provê conversão de eventos Java para eventos CORBA e vice-versa nas portas de sinal;
  - permite manipular propriedades de componentes e contêineres por meio de funções de acesso tipo *get* e *set*, bem como monitorar a alteração de propriedades por meio de notificações (via eventos Java);
  - encapsula toda a manipulação de fluxos multimídia produzidos e consumidos pelas portas de fluxo, inclusive gerenciamento de qualidade de serviço no nível de rede e de aplicação;
  - provê geração completa de *valuetypes* IDL (*Interface Definition Language*) para cada tipo de evento especificado em XML;
  - provê gerenciamento completo de mobilidade para agentes móveis.
- Utilização de padrões de projeto. Por exemplo, determinadas folhas de estilo contêm em sua implementação o padrão de projeto *Observador* [16] para notificação de eventos e alteração de propriedades.
- Gerência e comunicação de objetos distribuídos. Todas as funções relativas à transparência de distribuição e localização são realizadas pelo *framework de componentes* e pelo contêiner.

4.3 A Plataforma CCM-tel

A plataforma CCM-tel suporta o modelo CM-tel e em decorrência disto é uma plataforma aberta e neutra. A plataforma enfatiza a utilização de padrões, por exemplo padrões associados à Internet (HTTP, HTML, XML e Java), padrões OMG (UML e CORBA) e padrão W3C (XSLT). A plataforma CCM-tel possibilita que os requisitos não-funcionais da aplicação sejam especificados (e não implementados) pelo desenvolvedor da aplicação.

A Figura 4.6 apresenta a arquitetura *3-tier* da plataforma CCM-tel com as soluções tecnológicas empregadas para a implementação do modelo de componentes CM-tel. Esta arquitetura é uma instância do padrão arquitetural CM-tel, definido na Seção 3.4. No terceiro *tier*, correspondente ao *framework* de integração, a tecnologia CORBA é empregada e proporciona as funcionalidades de interoperação básicas para o acesso remoto a componentes, contêineres e agentes móveis. Alguns exemplos de tais funcionalidades são o ORB, repositórios CORBA e o suporte à manipulação de propriedades, propagação de eventos, fluxos de mídia contínua e chamada de métodos específicos dos componentes, contêineres e agentes móveis. O segundo *tier*, consiste de arquivos disponibilizados no formato *jar* como, por exemplo, o acesso a cada um dos quatro serviços e facilidades CORBA, os serviços comuns, as classes correspondentes aos contêineres, os *frameworks* de componentes e os *frameworks* de agentes móveis. O primeiro *tier*, contém os componentes e agentes móveis que foram desenvolvidos de acordo com os contratos de uso (*stubs e skeletons* IDL e contratos de especificação funcional semântica em XML) e de realização impostos pelo modelo de projeto CM-tel. A máquina virtual da plataforma Java é utilizada como plataforma de suporte de execução (*runtime*).

A plataforma CCM-tel é construída a partir dos seguintes elementos:

- uma ferramenta de implementação física (CCMtel-Builder);
- um conjunto de serviços comuns para a implementação de componentes, agentes móveis e contêineres;
- serviços e facilidades CORBA para suporte à interação entre componentes, contêineres e agentes móveis.

Na seqüência, apresentamos a descrição dos três elementos acima.

4.3.1 Ferramenta de Implementação Física CCM-tel

A Figura 4.7 ilustra os elementos da implementação física da plataforma CCM-tel (ferramenta CCMtel-Builder). CCMtel-Builder realiza o modelo de implementação física descrito na Seção 3.5 e consiste, basicamente, de uma máquina de transformações e de uma máquina de construção de código.

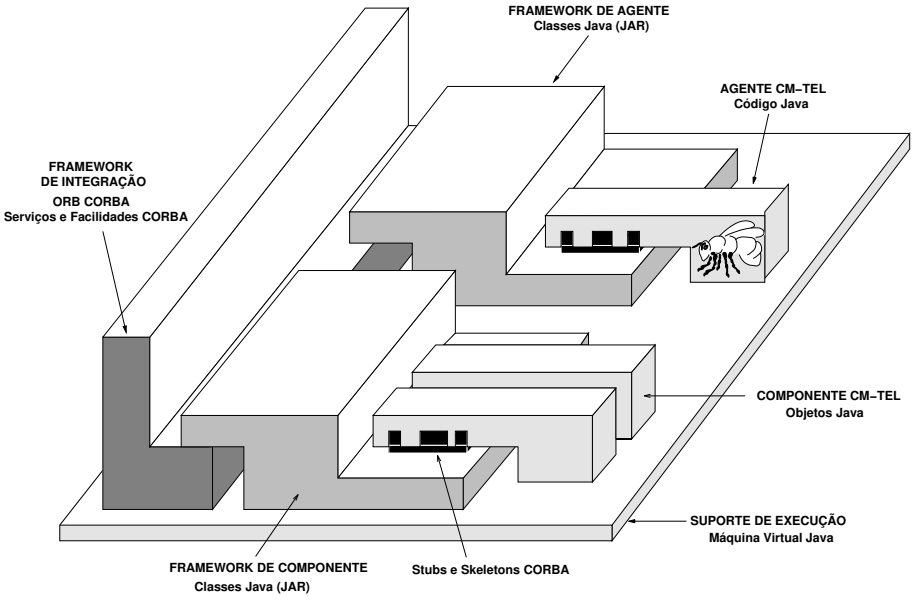

Fig. 4.6: Arquitetura da plataforma CCM-tel.

Máquina de Transformações

A máquina de transformações, que implementa as transformações descritas na Seção 4.2.3, consiste de dois subsistemas de software: um analisador XML e um processador XSLT. A máquina de transformações opera da seguinte maneira. O analisador XML valida o documento de entrada de acordo com o *XML Schema* estabelecido para este documento. Se a validação for bem sucedida, a árvore do documento associada ao arquivo de entrada XML é percorrida pelo analisador. O analisador seleciona, para cada nó na árvore, o conjunto de folhas de estilo assinalados a este nó. Esta seleção se dá com o auxílio de um arquivo de configuração em forma de tabela. A chave de acesso da tabela é o nome do nó (marcação XML) e os valores associados à chave são os arquivos correspondentes às folhas de estilo. Para cada folha de estilo associada ao nó, o analisador invoca o processador XSLT para transformar o documento XML de entrada de acordo com as regras de transformação presentes na folha de estilo. A título de exemplo, suponha o processamento de um documento XML especificando um componente CM-tel que define uma porta transmissora de mídia contínua. O analisador XML ao encontrar o nó de nome *transmitter* invoca duas transformações no documento XML, uma para gerar a implementação e outra para gerar a fábrica da porta transmissora.

As transformações são conduzidas em duas etapas. Na primeira etapa, a máquina de transformação determina se o arquivo de entrada é um arquivo XMI. Caso seja, o conjunto de folhas de estilo referentes à transformação especificação-projeto (Seção 4.2.3) é aplicado ao documento de entrada. Como resultado, tem-se um arquivo XML de saída contendo a representação do componente ou con-

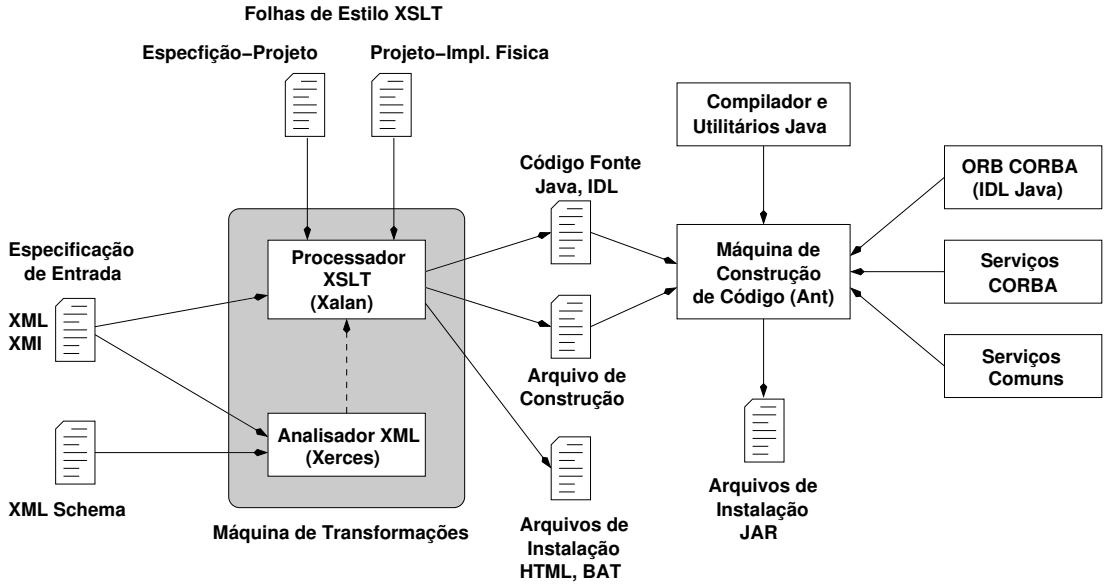

Fig. 4.7: Elementos da ferramenta CCMtel-Builder.

têiner CM-tel. Tendo este arquivo como entrada (ou o arquivo original, caso o mesmo não seja um arquivo XMI), a máquina de transformações inicia a segunda etapa, aplicando o conjunto de folhas de estilo referentes às transformações projeto-implementação física (Seção 4.2.3) ao documento de entrada. Esta etapa produz arquivos contendo código fonte, arquivos de construção e arquivos de distribuição JAR. Adicionalmente, na segunda etapa, o descritor de distribuição no formato XML é transformado para um formato adequado para a tecnologia, no caso HTML e BAT.

A plataforma CCM-tel utiliza em sua máquina de transformações o analisador XML Xerces2 e o transformador XSLT Xalan, ambos disponibilizados pelo projeto Apache[2]. Xerces2 e Xalan são escritos em Java e considerados como as ferramentas mais versáteis para processamento de documentos XML.

Máquina de Construção de Código

A máquina de construção de código processa o código fonte gerado pela máquina de transformação. Este processamento consiste na compilação, empacotamento e assinatura de código. A Figura 4.8 ilustra o processo de construção de código.

A plataforma CCM-tel utiliza a ferramenta de construção Ant do projeto Jakarta[3], um sub-projeto do projeto Apache. Ant é ferramenta tipo *make* escrita em Java e independente do *shell* do sistema

[2] `http://www.apache.org/`
[3] `http://www.apache.org/jakarta/`

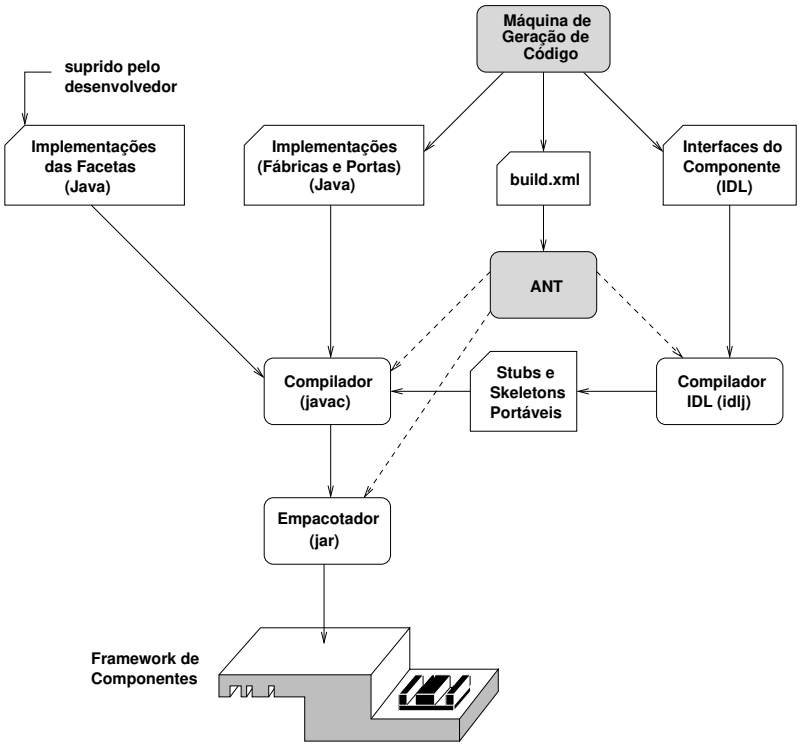

Fig. 4.8: Processo de construção do código executável.

operacional. Ant utiliza um arquivo XML para conduzir o processo de construção de código (dependências, parâmetros de compilação e empacotamento, etc.).

A máquina de construção de código utiliza ainda um compilador IDL (idlj), um compilador Java (javac), um empacotador (jar) e um assinador de código (jarsigner). Todas estas ferramentas fazem parte da plataforma Java (J2SE). Além do código gerado, a máquina de construção de código emprega ainda pacotes Java que implementam o ORB, os serviços comuns e os serviços e facilidades CORBA.

Código Gerado

A Figura 4.9 ilustra a interface da ferramenta CCMtel-Builder com arquivos de entrada de componentes, contêineres e descritores de distribuição especificados em XML ou XMI, a partir dos quais o código é gerado.

A ferramenta CCMtel-Builder gera os seguintes artefatos de *software* para suporte a componentes:

1. interfaces IDL que especificam:

   - a interface equivalente do componente;
   - as portas e elemento de configuração do componente;

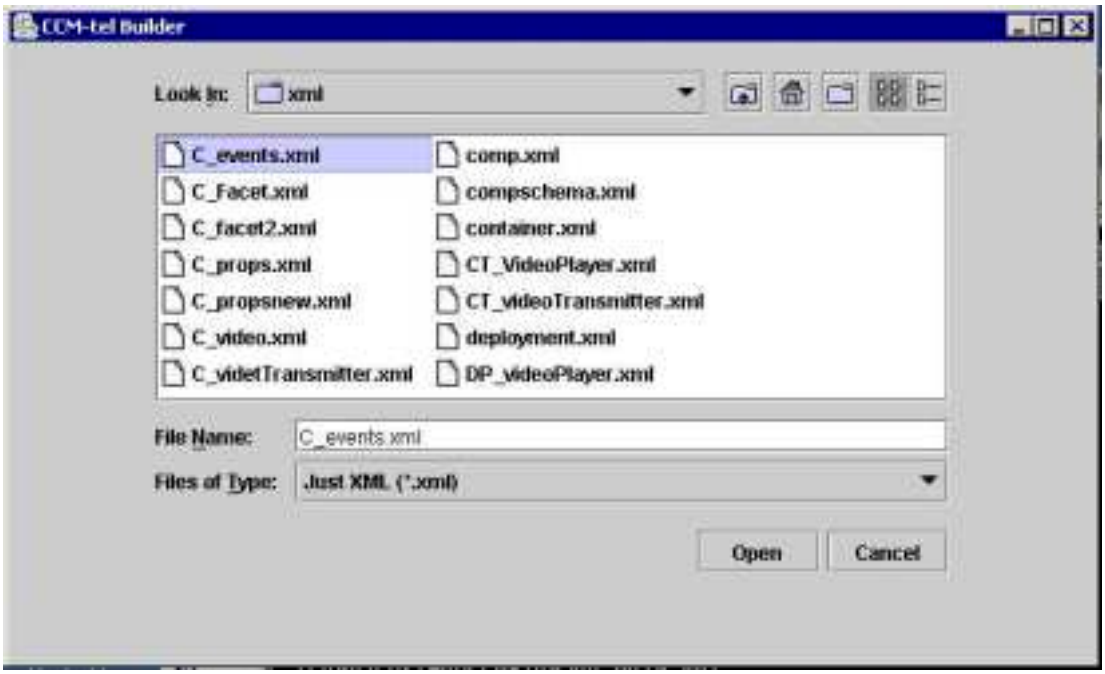

Fig. 4.9: Interface da ferramenta CCMtel-Builder [2].

- a fábrica do componente (*ComponentHome*);

2. classes Java que implementam:

   - as interfaces IDL citadas acima, denominadas serventes (*servants*) CORBA (interface equivalente, portas, elemento de configuração e fábrica do componente);
   - as fábricas para cada um dos serventes CORBA;
   - facilidades para manipulação das propriedades definidas pelos elementos de configuração do componente;
   - facilidades para localizar instâncias do componente especificado em contêineres (classe tipo *Finder* que retorna a referência da interface equivalente de um componente dado o seu contêiner e chave primária).

Para o suporte a contêineres CCM-tel, a ferramenta CCMtel-Builder gera os seguintes artefatos de software:

1. interfaces IDL que especificam:

   - o localizador de fábricas do contêiner (*HomeFinder*);

- a fábrica (agência) do agente móvel (*AgentHome*).

2. classes que implementam:

   - as interfaces IDL citadas acima (serventes CORBA);
   - facilidades para manipulação das propriedades de configuração do contêiner;
   - o contêiner especificado na forma de processo ou *applet* Java.

Estes artefatos uma vez compilados e empacotados formam o *framework* de componentes para o componente especificado.

A ferramenta CCMtel-Builder gera dois tipos de arquivos para a instalação de contêineres CCM-tel:

- arquivo de lote (*shell scripts*), para instalar contêineres baseados em processos;
- arquivo HTML, para instalar contêineres baseados em *applets* Java.

Finalmente, a ferramenta CCMtel-Builder gera um arquivo XML (build.xml) utilizado pela ferramenta Ant para a construção do código gerado.

Nomenclatura para o Código Gerado

As regras para a convenção de nomes para os artefatos de *software* gerados a partir da especificação de componentes, contêineres e descritores de distribuição em XML fazem parte da especificação do mapeamento CCM-tel. Tais regras estipulam como interfaces e classes são nomeadas.

Interfaces IDL

Para um componente CCM-tel são especificadas interfaces IDL para a fábrica, as portas, elemento de configuração e interface equivalente do componente. Tais interfaces são definidas no escopo de um módulo cujo nome é formado acrescentando-se o sufixo *Component* ao tipo do componente (dado pelo atributo *type* da especificação XML do componente). O nome da fábrica é formado acrescentando-se o sufixo *Home* ao tipo do componente. Excetuando-se os receptáculos, as interfaces das portas e elementos de configuração são nomeados com o próprio nome destes elementos (dados pelo atributo *name* da especificação XML). Finalmente, o nome da interface equivalente coincide com o tipo do componente.

As operações para estas interfaces são definidas conforme o modelo de projeto CM-tel descrito na Seção 3.3. Uma pequena alteração foi introduzida na definição das operações. Esta alteração consiste

em "tipificar" as operações de navegação (*provide*) e de conexão entre portas (*connect*, *disconnect*, etc.). A estas operações é acrescentado o nome da interface (separado pelo caractere _). Esta modificação contribui para a robustez do código gerado (a manipulação incorreta destas operações são detectadas em tempo de compilação) além de eliminar o parâmetro das operações tipo *provide*.

Por exemplo, associando as regras aos valores obtidos da especificação XML para o componente CM-tel de tipo *C_Comp*, conforme descrito na Seção 3.3.2 e que define a porta transmissora de vídeo *TR_VideoCamera* com padrão de interação ponto-ponto, são proporcionadas as seguintes interfaces IDL e operações:

```
module C_CompComponent {
  interface C_CompHome { ... };     // fábrica do componente
  interface C_Comp {                // interface equivalente
     TR_VideoCamera provide_TR_VideoCamera();
     };
  interface TR_VideoCamera {        // porta transmissora de vídeo
     connect_TR_VideoCamera ( ... );
     disconnect_TR_VideoCamera ( ... );
     };
};
```

Adicionalmente, construções IDL são geradas para os tipos de eventos e propriedades declarados na especificação do componente. Eventos declarados pelas portas de sinal são mapeados em objetos por valor (*valuetypes*) IDL com os mesmos atributos definidos na especificação dos eventos. No caso de elementos de configuração, as propriedades tipo *struct* e *sequence* são mapeadas para IDL utilizando-se as construções IDL *struct* e *sequence*. Propriedades tipo *simple* são mapeadas para os tipos IDL correspondentes (octet, long, etc.).

Classes Java

As convenções de nomes para as classes Java geradas a partir da especificação em XML de um componente são:

- implementações das interfaces têm o sufixo *Servant*.java, acrescentado ao nome da interface;
- implementações da interface equivalente têm o sufixo *Servant*.java, acrescentado ao tipo do componente;
- implementações das fábricas de componentes têm o sufixo *HomeServant*.java, acrescentado ao tipo do componente;

- fábricas de componentes, portas e elemento de configuação têm o sufixo *Factory*.java, acrescentado ao tipo do componente, nome da porta ou nome do elemento de configuração;
- localizadores de componentes têm o sufixo *Finder*.java, acrescentado ao tipo do componente;
- classes de manipulação de propriedades de componentes têm o sufixo *Properties*.java, acrescentado ao nome do elemento de configuração.

As operações de manipulação de propriedades do componente também foram "tipificadas" acrescentando-se o nome da propriedade a estas operações. Por exemplo, associando as regras aos valores obtidos da especificação XML para o componente CM-tel de tipo *C_Comp*, que define o elemento de configuração *Conf* e a propriedade *background* de tipo *string*, conforme descrito na Seção 3.3.2, é apresentado:

```
public class ConfProperties {
   String getbackground() { ... }
   void setbackground(String _value) { ... }
}
```

As convenções de nomes para as classes Java geradas a partir da especificação em XML de um contêiner são:

- a classe que implementa as funcionalidades do contêiner tem o sufixo *Container*.java, acrescentado ao tipo do contêiner.
- a classe executável para o caso do contêiner ser instalado como processo tem o sufixo *Server*.java, acrescentado ao tipo do contêiner.
- a classe executável para o caso do contêiner ser instalado como *applet* tem o sufixo *ServerApp*.java, acrescentado ao tipo do contêiner.

As convenções de nomes para os arquivos de instalação gerados a partir da especificação em XML de um descritor de distribuição são:

- arquivo de instalação para o caso do contêiner executar como um processo tem o nome do campo de extensão *bat* acrescentado ao nome do arquivo especificado em XML.
- arquivo de instalação para o caso do contêiner executar como uma *applet* tem o nome do campo de extensão *html* acrescentado ao nome do arquivo especificado em XML.

4.3.2 Serviços Comuns da Plataforma CCM-tel

Os serviços comuns da plataforma CCM-tel correspondem a uma coleção de serviços que oferecem funcionalidades comuns a componentes, contêineres e agentes móveis. Estes serviços se subdividem em serviços de componentes e *framework* de agentes.

Serviços de Componentes

Os serviços de componentes consistem de classes utilitárias empacotadas em um único arquivo (*comp.jar*). Este arquivo deve ser acessível pelos contêineres durante a sua instalação. As classes utilitárias são listadas abaixo.

- a classe Localizador de Fábricas (*HomeFinderServant*), que define o método *find* para a obtenção das referências das fábricas dos componentes e dos agentes móveis. Esta classe foi apresentada na Seção 3.3.3;
- a classe Localizador do Serviço de Nomes (*NSFinder*), que define o método *find* para a obtenção da referência do serviço de nomes CORBA;
- a classe *DeploymentDescriptor*, que armazena os valores das propriedades obtidas a partir do descritor de distribuição;
- a classe *DeploymentProperties*, que define os métodos *get* e *set* para a manipulação de propriedades de instalação, definidas para contêineres.

Frameworks de Agentes CCM-tel

O *Framework* de Agentes CCM-tel é uma infra-estrutura que viabiliza a execução de agentes móveis em contêineres CCM-tel. Consiste das seguintes classes e interfaces Java:

- a classe *AgentHomeServant* que fornece a fábrica do agente. Esta classe implementa três interfaces:
  1. a interface *MAFAgentSystem* especificada em IDL pelo MAF. As operações desta interface são delegadas para a implementação do MAF;
  2. a interface *AgentHome* (Seção 3.3.3);
  3. a interface *MAFAgency* que define métodos que a agência disponibiliza para os agentes móveis. Através desta interface, um agente móvel acessa as referências da agência que o instanciou, o contêiner que abriga a agência e o ORB e POA do contêiner.

- a interface Java *MAFAgent*, que deve ser implementada por todo agente móvel CCM-tel. As operações desta interface são as mesmas da classe Java *AgentImpl* descrita na Seção 3.3.3.

4.3.3 Serviços CORBA da Plataforma CCM-tel

Atualmente, quatro serviços CORBA são disponibilizados pela plataforma CCM-tel. Estes serviços correspondem ao serviço de propriedades, de eventos, de gerência e controle de fluxo de mídia contínua (A/V Streams) e de agentes móveis (MAF). Estes serviços foram implementados em Java, sendo integrados à plataforma por meio de arquivos no formato JAR.

Serviço de Propriedades

O serviço de propriedades do OMG [107] permite associar um conjunto de propriedades (pares atributo-valor) a determinado objeto CORBA. Atributos são cadeias de caracteres (*strings*) e valores são tipos opacos (Any). O serviço define seis interfaces IDL para o suporte às suas funcionalidades. As interfaces relevantes para a implementação do serviço de propriedades correspondem às quatro apresentadas a seguir. As duas restantes são especializações da interface *PropertySet*.

- a interface *PropertySet* define um conjunto de propriedades com operações para a manipulação destas propriedades, tais como definir, obter, alterar, listar e remover propriedades;
- a interface *PropertySetFactory*, corresponde à fábrica que cria *PropertySets*;
- a interface *PropertiesIterator* proporciona operações para acesso a múltiplas propriedades de um *PropertySet*;
- a interface *PropertyNamesIterator* proporciona operações para acesso a múltiplos nomes das propriedades.

A Figura 4.10 ilustra os principais elementos do serviço de propriedades que são implementados por meio de objetos que suportam as interfaces *PropertySet* e *PropertySetFactory*. Em adição, ilustra a classe *PropertyObject* que modela as propriedades.

As propriedades são encapsuladas em objetos da classe *PropertyObject*. Estes objetos armazenam o nome da propriedade, o seu tipo, o seu valor corrente, a permissão de acesso, bem como um conjunto de subscritores (*listeners*). Os subscritores são objetos locais que se cadastram para serem notificados quando ocorrem alterações em determinadas propriedades. O modelo de subscrição e de notificação, implementado pelo *PropertyObject*, segue o padrão de projeto *Observador*, de tal forma que quando uma propriedade armazenada em um objeto *PropertyObject* é alterada ou removida, todos os subscritores são notificados desta alteração.

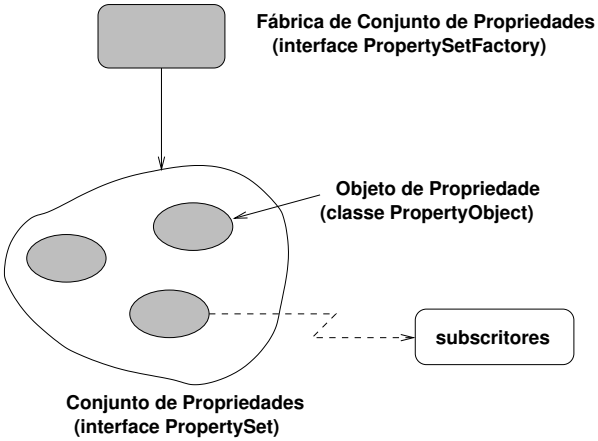

Fig. 4.10: Principais elementos do serviço de propriedades do OMG.

Na implementação do serviço de propriedades, todos os objetos CORBA especificados pelo OMG foram implementados para as interfaces IDL relevantes à nossa implementação de acordo com o documento de padronização [26].

Integração do Serviço de Propriedades na Plataforma CCM-tel

A Figura 4.11 ilustra a integração do serviço de propriedades na plataforma CCM-tel. Os elementos de configuração de componentes e contêineres são realizados diretamente sobre o serviço de propriedades. Quando um elemento de configuração é definido, uma classe contendo as propriedades presentes no elemento de configuração é gerada (*InitialPropertySet*). A fábrica do elemento de configuração (*PropertyFactory*) instancia uma fábrica de conjunto de propriedades do serviço (*PropertySetFactory*) e, a partir desta, um conjunto de propriedades (*PropertySet*) contendo as propriedades iniciais armazenadas na classe *InitialPropertySet*. Cada propriedade é armazenada em um objeto de propriedade (*PropertyObject*). Este objeto é capaz de armazenar em seu valor propriedades do tipo *simple*, *struct* e *sequence*. A Figura 4.11 ilustra um objeto (*PropertyListener*) monitorando certas propriedades pelo registro nos objetos de propriedade. Finalmente, uma classe com operações tipo *get* e *set* para estas propriedades (classe *Properties*) também é gerada. Esta classe é utilizada pelo cliente para manipular as propriedades presentes no elemento de configuração.

A Figura 4.12 ilustra a dinâmica para um dos possíveis cenários de interação para as classes presentes na Figura 4.11.

1. a fábrica da porta (*PropertyFactory*) obtém os valores das propriedades especificadas em XML, utilizando-se do método *get_initial_propertyset* da classe *InitialProperties*.

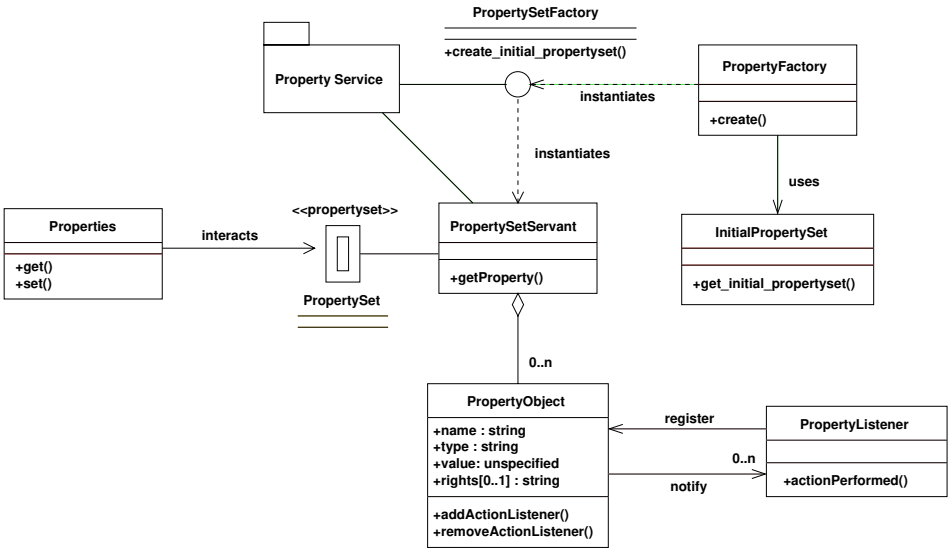

Fig. 4.11: Integração do serviço de propriedades na plataforma CCM-tel.

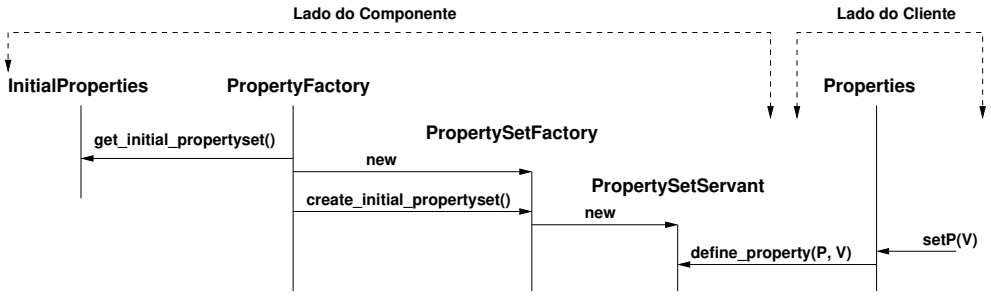

Fig. 4.12: Dinâmica da operação envolvendo propriedades.

2. de posse dos valores das propriedades, a fábrica da porta instancia a fábrica *PropertySetFactory* do serviço de propriedades e, a partir desta, um *PropertySet* com estas propriedades iniciais.

No lado do cliente do serviço de propriedades, o desenvolvedor deve implementar os seguintes passos:

1. obter a referência do *PropertySet* por meio da operação *provide_* da interface equivalente do componente que especificou o elemento de configuração no lado do componente.

2. instanciar a classe *Properties* fornecida pela plataforma, que contém métodos do tipo *get* e *set* para a manipulação das propriedades, passando a referência obtida acima.

3. utilizar o método *setP(V)* para definir ou alterar o valor V da propriedade P. Este método invoca o método *define_property(P,V)* do serviço de propriedades para atualizar o valor da

propriedade.

Propriedades podem ser monitoradas por meio das operações *addActionListener* e *removeActionListener* definidas na classe *PropertyObject*. Estas operações possibilitam o registro e o cancelamento de registro de monitores de propriedades. Monitores de propriedades devem implementar o método *actionPerformed* pelo qual serão enviadas as notificações de alteração de valores de propriedades (ou a sua remoção) por parte dos objetos da classe *PropertyObject*.

Serviço de Notificação de Eventos

O serviço de eventos define dois tipos de pontos terminais de eventos: produtores e consumidores de eventos. Os produtores geram eventos, enquanto que os consumidores os processam, tomando ações próprias da lógica da aplicação.

A especificação do serviço de eventos do OMG [108] proporciona um padrão aberto, independente de plataforma e baseado em CORBA. Os eventos são notificados entre os produtores e consumidores por meio de requisições CORBA. Este serviço consiste de um modelo arquitetural com um conjunto de objetos distribuídos e de interfaces IDL padronizadas que prescrevem as regras de interação para a manipulação de eventos.

O modelo arquitetural fornece dois tipos de modelos para a notificação de eventos: *push* e *pull*. No modelo *push*, os produtores de eventos tomam a iniciativa de notificar os consumidores por ocasião da ocorrência de um determinado tipo de evento. No modelo *pull*, os consumidores de eventos tomam a iniciativa de requisitar os eventos junto aos produtores.

O serviço proporciona toda a gerência de eventos, através de um canal de comunicação entre produtores e consumidores, de forma totalmente assíncrona e sem nenhum conhecimento mútuo. Este canal desacopla totalmente as notificações, atuando como um intermediário entre os pontos terminais de eventos. Ele é um objeto genérico que transfere eventos de forma opaca (sem inspecionar ou alterar o conteúdo dos eventos que transfere). Os produtores e consumidores determinam entre si as semânticas dos eventos.

A Figura 4.13 ilustra os principais elementos do serviço de eventos operando segundo o modelo *push*. A parte superior da figura ilustra o processo de conexão de produtores e consumidores através do canal de eventos. O canal consiste de dois objetos tipo *proxy*, um interagindo com o produtor e outro com o consumidor. Estes objetos provêem operações tipo *connect* e *disconnect* para a conexão e disconexão de produtores e consumidores de eventos junto ao canal.

A parte inferior da Figura 4.13 ilustra o processo de difusão de eventos. O objeto *ProxyPushConsumer* oferece ao produtor de eventos uma operação (*push*) através da qual o produtor submete eventos ao canal. Eventos recebidos pelo objeto *ProxyPushConsumer* são repassados ao objeto *ProxyPush-*

Supplier para difusão. A difusão se dá através da operação *push* implementada pelos consumidores de eventos.

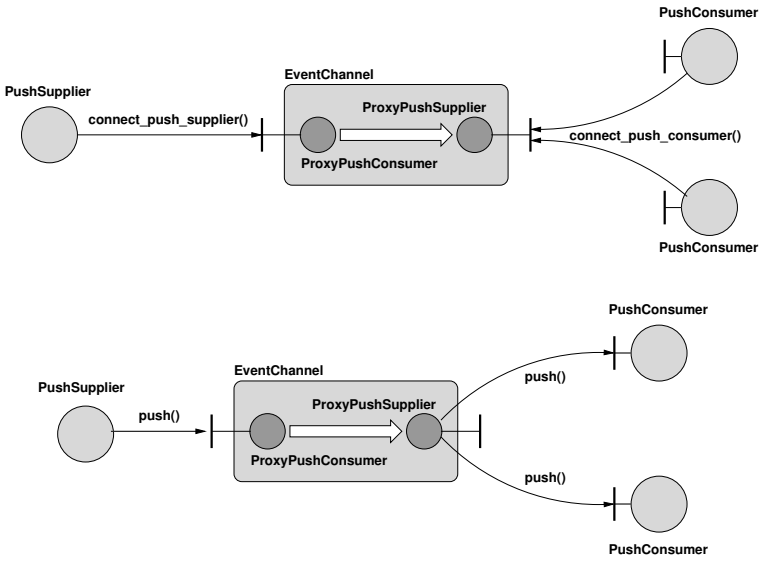

Fig. 4.13: Principais elementos do serviço de eventos do OMG.

Na implementação do serviço de eventos, todos os objetos CORBA especificados pelo OMG foram implementados para as interfaces IDL, de acordo com o documento de padronização [26].

Integração do Serviço de Eventos na Plataforma CCM-tel

A Figura 4.14 ilustra a integração do serviço de eventos na plataforma CCM-tel. As portas produtoras e consumidoras de eventos dos componentes herdam as funcionalidades propiciadas pelo serviço de eventos por meio das interfaces *PushSupplier* e *PushConsumer*. A utilização do canal de eventos é necessária apenas no caso de portas publicadoras de eventos. Portas emissoras e consumidoras de eventos são interconectadas diretamente, sem a intermediação do canal de eventos. Observa-se ainda na Figura 4.14 que as portas produtoras de eventos agem como *listeners* de eventos locais (eventos Java), enquanto que as portas consumidoras de evento agem como geradoras de eventos locais. As portas consumidoras implementam métodos internos para o registro e cancelamento do registro de componentes da aplicação, que processam os eventos recebidos (operações *addActionListener* e *removeActionListener*, respectivamente). Os componentes da aplicação que processam eventos devem implementar o método *actionPerformed* de notificação, que recebe como parâmetro o evento gerado pela aplicação. A conversão de eventos Java para eventos CORBA e vice-versa é disponibilizada no código gerado para as portas produtoras e consumidoras de eventos.

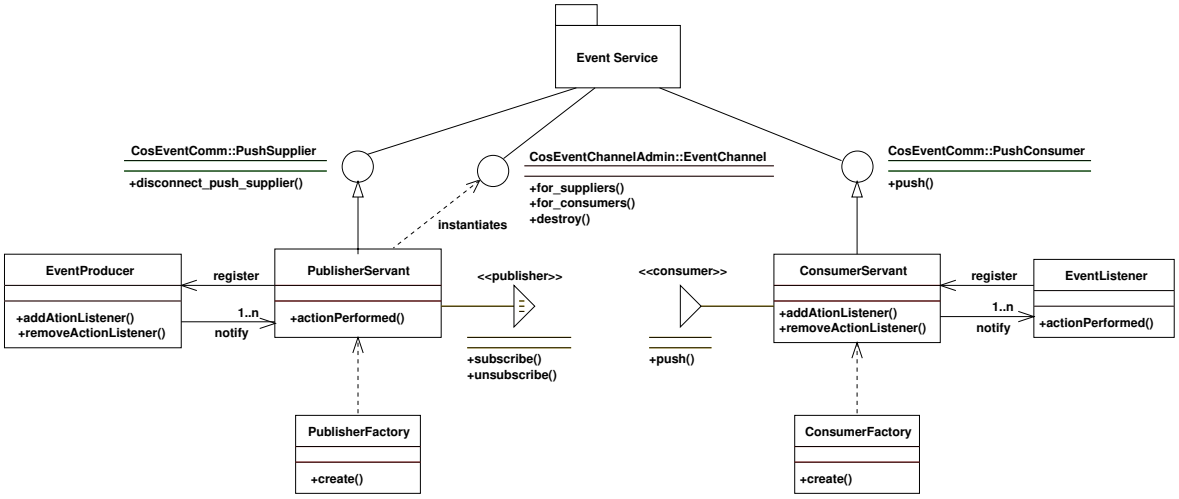

Fig. 4.14: Integração do serviço de eventos na plataforma CCM-tel.

A Figura 4.15 ilustra a dinâmica para um dos possíveis cenários de interação para as classes presentes na Figura 4.14.

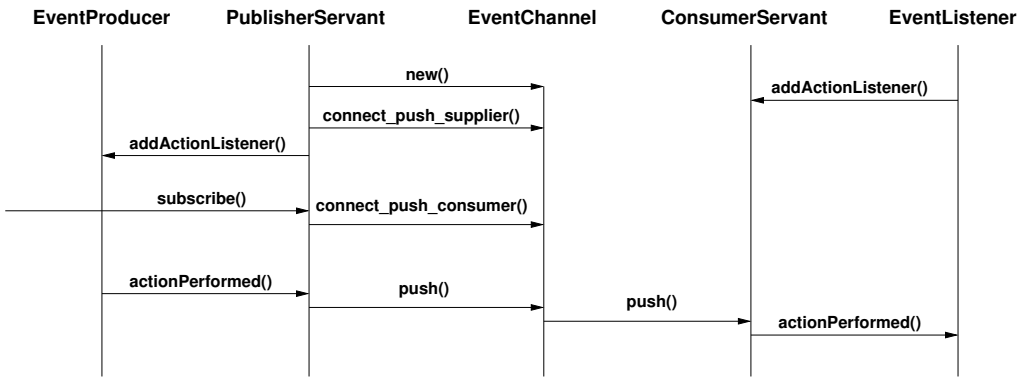

Fig. 4.15: Dinâmica da operação envolvendo eventos.

1. A porta publicadora de eventos do componente (*PublisherServant*) instancia um canal de eventos e se conecta ao canal.

2. A porta publicadora de eventos se registra em uma classe da aplicação responsável por gerar eventos locais (classe *EventProducer*).

3. No lado do componente consumidor de eventos, a classe da aplicação responsável por tratar eventos (classe *EventListener*) se registra na porta consumidora de eventos do componente.

4. No momento que a porta consumidora se subscreve na porta publicadora, esta registra a porta consumidora no canal de eventos.

5. Assim que a classe *EventProducer* gera um evento local, a porta publicadora o converte em um evento CORBA e o submete ao canal para difusão. Ao receber o evento CORBA, a porta consumidora o converte para um evento local e notifica a aplicação.

É importante notar que a aplicação manipula apenas eventos Java. O código gerado implementa todo o processo de conversão e difusão destes eventos através da rede via serviço de eventos CORBA.

Serviço de Gerência e Controle de Fluxos de Áudio e Vídeo

A especificação do OMG de *Audio / Video Streams* (A/V Streams) [23] proporciona um padrão aberto, independente de plataforma e aderente a CORBA para controle e gerência de fluxos de mídia contínua (*streams*) com garantias de qualidade de serviço. *Streams* são os elementos básicos de suporte à comunicação multimídia e representam a transferência de um ou mais fluxos de mídia contínua de um produtor para um ou mais consumidores.

A/V Streams consiste de um modelo arquitetural com um conjunto de objetos distribuídos e de interfaces IDL padronizadas. Este modelo arquitetural proporciona módulos bem definidos, semânticas e definições necessárias para o estabelecimento e controle de *streams*, bem como as interfaces IDL que disponibilizam as suas operações visíveis externamente. A/V Streams trata configurações de fluxo ponto-ponto e ponto-multiponto que podem ser utilizadas como blocos de construção para fluxos muitos-para-muitos e muitos-para-um. Conforme esta especificação, todas as operações de controle e de sinalização são realizadas através do ORB. No entanto, os segmentos de mídia constituintes de um fluxo são transportados por fora do ORB, via um protocolo específico de transferência de mídia (TCP, UDP, RTP/UDP e ATM/AAL5). Por conter as caracterísiticas acima, a especificação A/V Streams oferece interoperabilidade entre aplicações multimídia distribuídas implementadas por diferentes fornecedores.

A Figura 4.16 ilustra a arquitetura e os principais elementos da especificação A/V Streams que proporciona dois perfis da especificação: perfil completo (*full profile*) e perfil leve (*light profile*). No perfil leve a aplicação manipula os fluxos de forma agregada, enquanto no perfil de especificação completo a aplicação manipula fluxos individuais.

Na Figura 4.16, os elementos sombreados estão disponíveis apenas no perfil completo da especificação. O "duto" representa um *stream* transportando um fluxo de áudio e um fluxo de vídeo. Os fluxos ligam pontos terminais (*flow endpoints*), modelados através da interface *FlowEndPoint*. Esta interface é especializada para produtores de fluxo (*FlowProducer*) e consumidores de fluxo (*FlowConsumer*). Um objeto denominado dispositivo de fluxo (*flow device*) age como fábrica de pontos

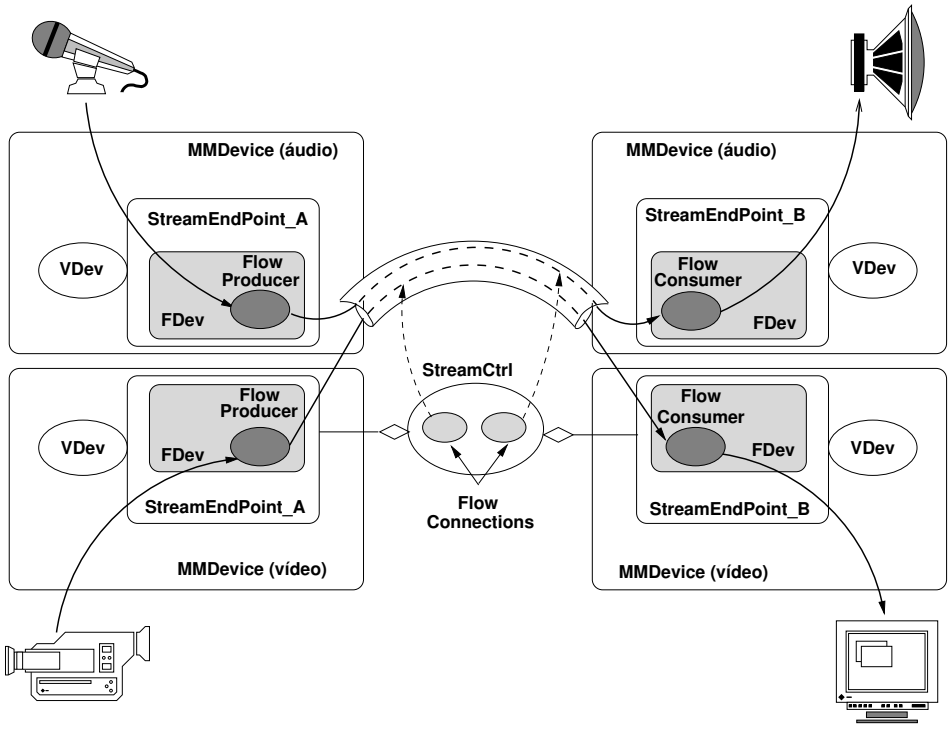

Fig. 4.16: Principais elementos do serviço A/V Streams do OMG.

terminais de fluxo. Este objeto é modelado pela interface *FDev*. A conexão de um produtor a um ou mais consumidores ocorre através de uma conexão de fluxo (*flow connection*). A interface *FlowConnection* abstrai o conceito de conexão de fluxo.

No perfil completo, as conexões e pontos terminais de fluxo específicos têm as suas operações visíveis e acessíveis para o controle de fluxos individuais. No perfil leve, a aplicação tem um controle em um nível de abstração maior, onde as interfaces *FlowEndPoint*, *FlowConnection* e *FDev* não são visíveis externamente. Neste perfil, os pontos terminais de fluxo são agregados em um objeto maior: o ponto terminal de *stream* (*stream endpoint*). A interface *StreamEndPoint* modela os pontos terminais de um *stream*. Os pontos terminais de um *stream* têm ainda um componente associado para fins de configuração (por exemplo, tipo de mídia). Este objeto é modelado pela interface *VDev* (*virtual device*). De maneira análoga, neste perfil as conexões de fluxo são agregadas em um único objeto: o controle de *stream* (*stream control*). A interface *StreamCtrl* modela este objeto. Os pontos terminais de *stream* são agregados em dispositivos multimídia (*multimedia devices*), modelados por meio da interface *MMDevice*. O dispositivo multimídia representa os dispositivos lógicos ou físicos que geram ou consomem os fluxos de mídia.

As interfaces *StreamCtrl*, *FlowConnection*, *FlowEndPoint* e por especialização as interfaces *FlowProducer* e *FlowConsumer* provêem operações para o estabelecimento e controle dos fluxos (*start,*

stop e destroy).

A implementação do padrão A/V Streams utilizada na plataforma CCM-tel fornece todas as interfaces para os dois perfis (completo e leve) previstos na especificação do OMG. Os pontos terminais de fluxo fazem uso do *Java Media Framework* (JMF) [104] para captura e apresentação de áudio e vídeo. Desta forma, tem-se uma implementação 100% Java do padrão A/V Streams. Áudio e vídeo são transferidos através da rede via protocolo RTP (*Real Time Protocol*) que utiliza o transporte sem conexão UDP. RTP encapsula segmentos de áudio no formato PCM (*Pulse Code Modulation*) e de vídeo no formato JPEG. Esta implementação é descrita em maiores detalhes nas referências [50][51]. Recentemente, foi incorporada à implementação a apresentação de mídia pelo produto QuickTime da Apple Computer Inc.[4].

Integração do Serviço A/V Streams na Plataforma CCM-tel

A Figura 4.17 ilustra a integração do serviço A/V Streams na plataforma CCM-tel. Esta integração utiliza o perfil completo da especificação.

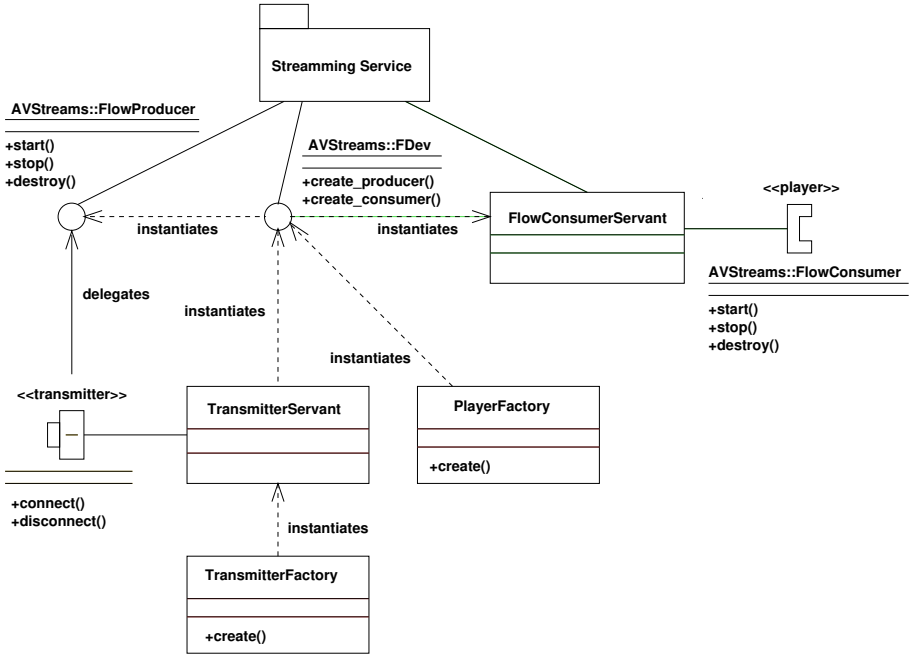

Fig. 4.17: Integração do serviço A/V Streams na plataforma CCM-tel.

Portas transmissoras e difusoras de fluxo delegam o processo de captura e transmissão de mídia para a interface *FlowProducer* do serviço A/V Streams. Os objetos que implementam estas portas (*TransmitterServant*, na Figura 4.17) instanciam um objeto *FDev* do serviço e, a partir deste, um

[4] `http://www.apple.com/qt/`

produtor de fluxo. As portas consumidoras de fluxos são realizadas diretamente sobre o serviço A/V Streams. A fábrica destas portas (*PlayerFactory*) instancia suas implementações (*FlowConsumerServant*) por meio da interface *FDev*.

Serviço de Agentes Móveis

A especificação MAF (*Mobile Agent Facility*) do consórcio OMG [108] proporciona a infraestrutura de suporte a agentes móveis em ambientes heterogêneos. MAF se concentra na padronização dos aspectos de gerência e de mobilidade (transferência) do agente móvel visando a interoperabilidade entre diferentes sistemas baseados em agentes. MAF não especifica como sistemas de agentes são construídos, mas como tais sistemas podem interoperar. Segundo a especificação MAF, toda a comunicação entre os sistemas baseados em agentes ocorre através do ORB. Entretanto, a comunicação entre agentes não é tratada pela especificação.

Neste trabalho, a definição de agentes e de agentes móveis está de acordo com a da especificação MAF, que se concentra fundamentalmente em agentes móveis [108]:

- "um agente é um programa de *software* autônomo, que age em benefício de uma pessoa ou organização. Cada agente tem sua *thread* própria de execução, de forma que as tarefas podem ser executadas a partir de iniciativas próprias";
- "um agente é móvel quando a sua execução não está restrita ao sistema no qual o agente iniciou a sua execução. Ele possui a habilidade de realizar o seu transporte, ou seja de se mover de um sistema baseado em agentes para outro na rede, a fim de acompanhar o cumprimento de sua tarefa. Ao se mover o agente transporta junto o seu código e estado". Neste contexto, o estado do agente pode ser o seu estado de execução ou propriedades a ele atribuídas que determinam o que ele deve fazer quando é reiniciado em outro sistema.

MAF especifica duas interfaces IDL. A interface *MAFAgentSystem* define as operações que um ambiente de execução do agente (agência) aderente à especificação deve prover. Estas são operações de controle e gerência de agentes, tais como operações de criação, transferência, controle de execução e monitoramento de agentes. A interface *MAFFinder* representa um serviço de nomes específico para agentes. Esta interface define operações para registrar, cancelar o registro e localizar agentes e sistemas baseados em agentes.

A Figura 4.18 ilustra os principais elementos do serviço de agentes. A parte superior da figura ilustra o processo de registro onde as agências (*MAFAgentSystem*) se registram no serviço de localização (*MAFFinder*). Agências podem registrar contextos de execução (*places*) por elas mantidos. Um contexto de execução define políticas e permissões para execução de agentes (por exemplo, um agente não autenticado deve executar em um contexto mais restritivo).

A parte inferior da figura ilustra o processo de transferência de agentes. A transferência se dá por meio da operação *receive_agent* disponibilizada pela agência. Ao receber o agente, este é registrado no serviço de localização, de forma que a entidade proprietária do agente possa localizá-lo em qualquer agência.

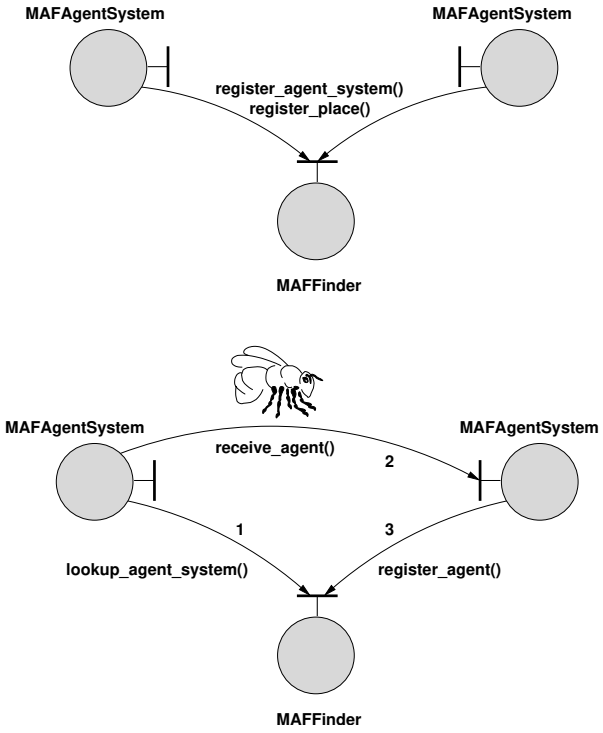

Fig. 4.18: Principais elementos do serviço de agentes móveis do OMG.

A implementação na linguagem Java do padrão MAF na plataforma CCM-tel é compatível com a especificação MAF e provê a implementação das interfaces *MAFAgentSystem* e *MAFFinder*. O sistema baseado em agentes, leve e de baixo *overhead*, emprega serialização e carregamento dinâmico [5] de classes nativas da plataforma Java. Agentes são serializados para transferência entre sistemas baseados em agentes. As classes necessárias para a restauração do agente na agência de destino podem ser carregadas dinamicamente a partir de um servidor HTTP.

Integração do Serviço de Agentes Móveis na Plataforma CCM-tel

Agentes móveis CCM-tel são desenvolvidos pela aplicação e devem implementar uma interface Java *MAFAgent* (Seção 3.3.3). Esta interface deriva das interfaces Java *Serializable* e *Runnable*, tornando assim o agente serializável e passível de execução na agência como uma *thread* Java, res-

[5] Local ou pelo protocolo HTTP.

pectivamente. A interface *MAFAgent* pode expor facetas (interfaces externas) e define ainda métodos de *callback* que tornam possível o controle do agente por parte do sistema de agentes. Estes métodos permitem à agência suprir o agente com a referência da agência que o abriga e a identidade atribuída ao agente. Ainda por meio desta interface, o agente é informado quando a sua execução é suspensa, retomada ou encerrada.

Os agentes móveis podem empregar todos os recursos da tecnologia CORBA; por exemplo, um agente pode acessar os serviços mantidos pelo ORB, acessar o ORB como cliente ou registrar alguns de seus objetos no adaptador de objetos portável como servidores CORBA. Esta capacidade dos agentes de utilizar os recursos do CORBA, permite que agentes móveis e componentes interajam através do ORB. Por exemplo, um agente pode obter a referência da interface equivalente de um componente e, a partir desta referência obter as demais interfaces do componente. De posse de tais interfaces, o agente poderá interconectar componentes, alterar propriedades dos componentes ou invocar métodos definidos pelas suas portas.

Ainda por meio do ORB e em adição à capacidade de interação com componentes, os agentes móveis podem executar funções, tais como:

- obter a referência do contêiner onde está hospedado e através desta interagir com as fábricas para criar e remover componentes e agentes;
- adicionar novas funcionalidades ausentes ao código original do contêiner, possibilitando desta forma estendê-lo e adaptá-lo em tempo de execução;
- interagir com outros agentes móveis (por exemplo, localizar agentes e interagir com as suas interfaces externas);
- efetuar ações de gerência de componentes e de contêineres, por exemplo identificando as conexões dos componentes hospedados;
- reconfigurar contêineres alterando as suas propriedades, por exemplo alguns de seus parâmetros de QoS.

A integração do serviço de agentes móveis na plataforma CCM-tel é ilustrada na Figura 4.19. A classe *AgentHomeServant*, parte do *framework* de agentes, é instanciada pelo contêiner quando o mesmo especifica *homes* de agentes. Esta classe instancia agentes e delega todo o gerenciamento de mobilidade ao serviço de agentes (interface *MAFAgentSystem*).

A Figura 4.20 ilustra um cenário típico contendo os passos envolvidos no processo de utilização de agentes móveis na plataforma CCM-tel. As principais interações envolvidas entre contêiner,

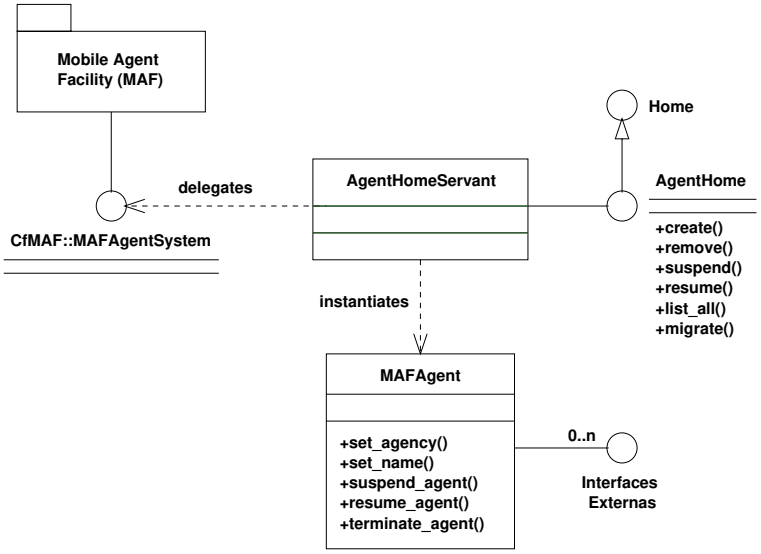

Fig. 4.19: Integração do serviço de agentes móveis na plataforma CCM-tel.

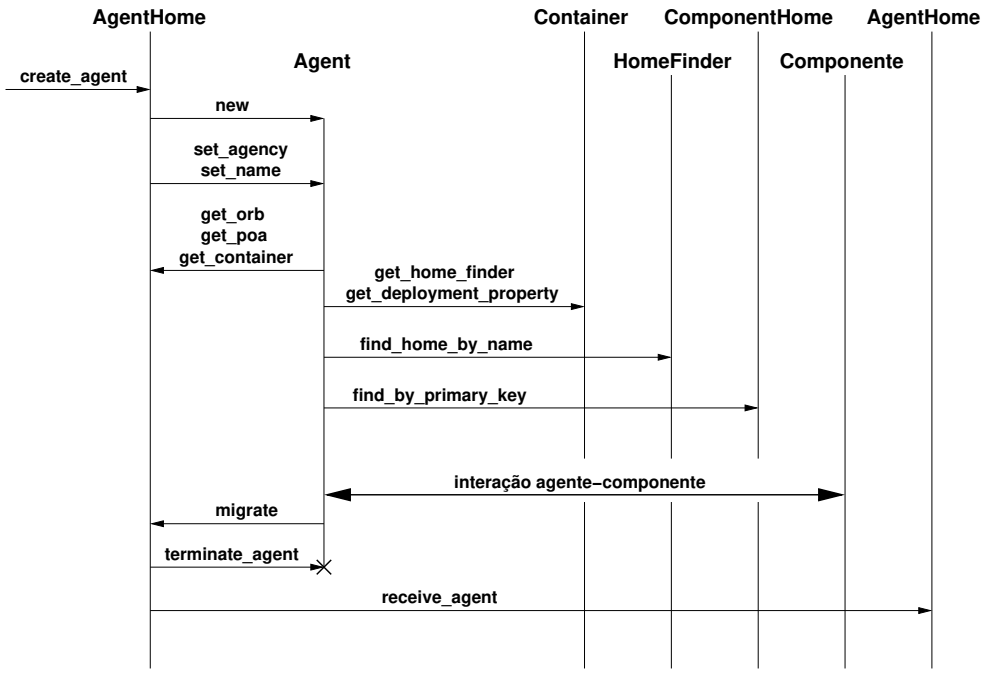

Fig. 4.20: Dinâmica da operação envolvendo agentes móveis.

AgentHomes, *ComponentHomes* e agentes móveis são apresentadas no diagrama de seqüência desta figura.

Resumidamente, um agente é instanciado em uma agência (*AgentHome*) pela operação *create_agent*. Uma vez instanciado, a agência invoca operações de *callback* no agente informando a referência da

agência e o nome atribuído ao agente na instanciação. De posse da referência da agência, o agente invoca operações internas da agência para obter as referências do ORB, POA e contêiner. Ele pode obter a referência do objeto *HomeFinder*. Com a referência do contêiner, o agente pode localizar por meio do *HomeFinder* as fábricas (*ComponentHome* e *AgentHome*) e, por meio destas, os componentes e outros agentes localizados neste contêiner. Neste momento, através do ORB, o agente pode interagir com os componentes e com outros agentes. Finalmente, o agente solicita à sua agência que o mesmo seja transferido para outra agência de destino (operação *migrate*). A agência serializa o estado do agente e termina a execução do mesmo. O processo de migração se completa com a transferência do agente[6] para a agência de destino.

4.3.4 Suporte a Facilidades de Qualidade de Serviço na Plataforma CCM-tel

O suporte a facilidades de qualidade de serviço (QoS) proporcionadas pela plataforma CCM-tel garante que os fluxos de mídia contínua tenham um tratamento preferencial em termos de alocação de recursos.

Por ser um requisito não-funcional, QoS deve ser suportada pelo contêiner. Este suporte, na plataforma CCM-tel, demanda extensões na infra-estrutura do contêiner, de forma que este seja capaz de executar funções de gerência de recursos de QoS em diferentes níveis. Conforme descrito na Seção 3.3.3, esta gerência de QoS (especificação, negociação, monitoramento, controle e renegociação dos níveis de qualidade de serviço) aplica-se desde o nível da aplicação (*hardware*, *software*, sistema operacional, plataforma de *middleware*) até o nível de infra-estrutura de rede (protocolos de rede, garantia de parâmetros de tráfego em arquiteturas de rede para provimento de QoS - DiffServ ou IntServ, etc.).

A plataforma CCM-tel gera a infra-estrutura necessária para que o contêiner forneça o suporte e facilidades para uma aplicação implementar a sua própria gerência de QoS em comunicações multimídia distribuídas. Neste suporte QoS é tratada nos seguintes níveis:

1. especificação, monitoramento e controle de QoS no nível da aplicação;
2. QoS no serviço A/VStreams;
3. reserva de recursos no nível de rede.

Especificação, Monitoramento e Controle de QoS no Nível da Aplicação

A qualidade de serviço desejada para os fluxos de mídia contínua é expressa inicialmente no nível da aplicação. A aplicação especifica no descritor de distribuição CCM-tel (marcação XML

[6]Mais precisamente, do estado do agente.

QoS), os valores dos parâmetros de QoS específicos para cada tipo de mídia (áudio ou vídeo) a ser utilizada. Estes atributos são necessários para a configuração inicial do contêiner durante a sua instalação e valem para todas as portas de fluxo de mídia declaradas nos componentes que executam neste contêiner. No exemplo do descritor de distribuição especificado em XML, apresentado na Seção 4.2.2, são declarados cinco parâmetros de qualidade de serviço para vídeo e quatro parâmetros de qualidade de serviço para áudio.

O mecanismo para o monitoramento e controle de QoS é provido pela aplicação. Para esta, CCM-tel não provê um mecanismo interno, mas sim facilidades para a implementação de tal mecanismo. Esta decisão deve-se ao fato de que estes mecanismos são altamente dependentes da aplicação e do ambiente de rede no qual a aplicação executa. Por exemplo, uma aplicação pode deixar a cargo do usuário a escolha de certos parâmetros de QoS, enquanto outra pode selecionar os mesmos parâmetros com base no desempenho da rede.

O monitoramento e controle de QoS consiste em monitorar os valores dos parâmetros de QoS no nível de rede, detectar violação de limites estabelecidos para estes parâmetros e, se for o caso, realizar ações de controle visando corrigir discrepâncias. Por exemplo, a taxa de bits em uma porta apresentadora de vídeo pode ser monitorada e, caso esta taxa exceda a taxa reservada, pode-se efetuar uma diminuição da qualidade de vídeo na porta transmissora (aumentando-se a taxa de compressão ou diminuindo-se a resolução espacial).

A plataforma CCM-tel também disponibiliza duas facilidades adicionais: um conjunto de propriedades para o monitoramento e outro para o controle de QoS. Estas propriedades são descritas a seguir.

Propriedades Para Monitoramento de QoS

Ao especificar um componente com portas produtora e apresentadora de fluxo contínuo, o projetista da aplicação pode definir um elemento de configuração (conjunto de propriedades) com o fim específico de monitoramento de QoS para estas portas. Este elemento de configuração deve possuir o mesmo nome da porta com o sufixo *Monitoring*. Cinco propriedades são definidas neste elemento de configuração:

- taxa de bits - propriedade *BitRate*;
- taxa de pacotes - propriedade *PacketRate*;
- *jitter* - propriedade *RtcpJitter*;
- fração de pacotes perdidos - propriedade *RtcpPacketLost*;

- fração de bytes perdidos - propriedade *RtcpFractionLost*.

Com base nos valores obtidos do gerente de sessão do serviço A/V Streams (*Session Manager* disponibilizado pelo JMF), a plataforma provê informações recentes sobre QoS e atualiza os valores das propriedades citadas acima a cada cinco segundos (este tempo é ajustável). As duas primeiras propriedades são obtidas localmente e correspondem a taxa de bits ou pacotes gerados pela porta produtora ou consumidos pela porta apresentadora de fluxos de mídia. As três propriedades restantes são obtidas de relatórios enviados por meio do protocolo RTCP (*Real Time Control Protocol*). Estes relatórios são trocados entre produtores e apresentadores de mídia que empregam o protocolo RTP (*Real Time Protocol*).

Propriedades Para Controle de QoS

Os parâmetros de instalação presentes no descritor de distribuição são armazenados como propriedades de configuração do contêiner. Por meio destas propriedades e após a instalação do contêiner, os parâmetros de instalação iniciais podem ser inspecionados e alterados. Dentre estes parâmetros, os parâmetros de QoS são de especial interesse. Uma aplicação pode obter a referência do elemento de configuração do contêiner e alterar os parâmetros das propriedades de QoS; por exemplo, qualidade de vídeo ou taxa de amostragem de áudio. A alteração das propriedades de configuração do contêiner afeta as conexões futuras de portas de fluxo contínuo para os componentes abrigados neste contêiner. Para as conexões já estabelecidas, pode-se atualizar os seus parâmetros de QoS para os novos parâmetros procedendo-se à parada e ao reinício do fluxo de mídia pelos métodos de controle de fluxos multimídia *stop* e *start*, expostos nas portas de fluxo contínuo.

QoS no Serviço A/V Streams

Na implementação do serviço A/V Streams, as operações de estabelecimento de fluxo, seja para conexão ponto-ponto (um transmissor e um apresentador) ou para conexão ponto-multiponto (um difusor e vários apresentadores), possuem parâmetros de qualidade de serviço. QoS é negociada entre as portas produtoras e apresentadoras de fluxo contínuo no momento da interconexão das portas.

A plataforma CCM-tel disponibiliza cinco parâmetros de qualidade de serviço para vídeo:

- taxa de amostragem - quadros/s;
- formato de vídeo - resolução (número de cores por pixel);
- tamanho da janela de apresentação (largura e altura em pixels);
- qualidade de vídeo (valor entre 0 e 1);

- padrão de codificação de mídia (RGB ou YUV).

Em adição, CCM-tel disponibiliza quatro parâmetros de qualidade de serviço para áudio:

- taxa de amostragem (segmentos/s);
- tamanho do segmento (bits);
- número de canais (1 ou 2);
- padrão de codificação (uLaw, Linear ou GSM).

Reserva de Recursos no Nível de Rede

A plataforma CCM-tel utiliza Interceptadores Portáveis CORBA para incorporar as novas funcionalidades de suporte para a reserva de recursos de QoS para fluxos contínuos à infra-estrutura do contêiner. Como descrito anteriormente, estes interceptadores podem interceptar métodos direcionados a componentes, agentes móveis ou até mesmo a serviços e facilidades CORBA.

Neste trabalho, utilizamos uma implementação de um interceptador de QoS (IQoS) para tratar, de forma totalmente transparente para o desenvolvedor, toda a complexidade e os diversos níveis de detalhes de comunicação e de negociação de QoS envolvidos, do nível da aplicação ao nível da rede. Este interceptador é instalado no contêiner CCM-tel em tempo de configuração e interage com um *Bandwidth Broker* em uma rede de serviços diferenciados (*DiffServ*). Uma primeira versão do *Bandwidth Broker* utilizado foi implementada em Java, por um dos membros da equipe [34].

O interceptador IQoS é expresso pela aplicação no descritor de distribuição, por meio da marcação XML *PortableInterceptor*, onde o atributo *class* fornece o nome da classe que implementa o interceptador. Em adição, a aplicação também declara, por meio da marcação XML *SLA*, o atributo correspondente ao identificador de contrato *DiffServ* utilizado na interação com o *Bandwidth Broker*. A plataforma CCM-tel gera e disponibiliza toda a infra-estrutura necessária para que o interceptador seja instalado no ORB do contêiner.

O interceptador IQoS proporcionado pela plataforma tem a função de interceptar algumas chamadas de métodos do serviço A/V Streams no seu retorno, dando um tratamento adequado a cada uma delas. No nosso caso, este tratamento consiste em contactar o *Bandwidth Broker*, responsável pela gerência da alocação de banda no domínio ao qual pertence. IQoS obtém os parâmetros de QoS no retorno da chamada e os mapeia para parâmetros no nível de rede (no caso, banda necessária ao fluxo). Neste momento, o interceptador implementado trata de toda a negociação de reserva de recursos de QoS. O mapeamento realizado pelo interceptador IQoS é simples e baseado em tabelas de mapeamento obtidas via medidas do tráfego gerado por diversas combinações de parâmetros de QoS [7].

[7] *trabalho de iniciação científica.*

De posse do identificador *SLA*, IQoS contacta o *Bandwidth Broker DiffServ* e solicita a reserva da banda fornecendo o destino do fluxo (o endereço IP do componente consumidor). Este esquema de suporte à reserva de recursos de QoS baseado em interceptação é bem flexível, pelas seguintes razões:

- a classe de serviço é determinada pelo *Bandwidth Broker*, de acordo com o parâmetro de instalação SLA, em função dos recursos disponíveis, da classe do usuário e do ambiente de rede onde o contêiner executa;
- caso os recursos não tenham sido alocados ao contêiner ou nenhum *Bandwidth Broker* esteja presente na rede, os fluxos de mídia terão tratamento do tipo "melhor esforço";
- caso a aplicação deseje alterar a qualidade de um fluxo, por exemplo pela ação de um monitor de QoS, o serviço A/V Streams provê operações para alterar os níveis de QoS de fluxos já estabelecidos. Estas operações são igualmente interceptadas pelo IQoS que irá interagir com o *Bandwidth Broker* para renegociar a reserva já estabelecida.

Cenário de Uso para Estabelecimento de Fluxos de Mídia com QoS

A Figura 4.21 ilustra um cenário possível das facilidades oferecidas para QoS representando uma visão espacial da interação entre um produtor e um apresentador de mídia.

A Figura 4.22 expõe o comportamento e os passos envolvidos no processo de estabelecimento de fluxos de mídia contínua com QoS. O diagrama de seqüência, nesta figura, provê uma visão temporal das interações representadas na Figura 4.21. Os passos no diagrama correspondem aos números da figura que provê a visão espacial. Apresentamos este cenário com a intenção de ilustrar o acesso ao serviço A/V Streams no caso de um componente especificado para utilizar a forma de interação ponto-ponto. Para o caso de uma especificação para a utilização da forma de interação ponto-multiponto, o modelo fornece uma seqüência de passos similar. Em adição, enfatizamos também que todos os passos ilustrados por esta seqüência estão incorporados no código gerado pela plataforma CCM-tel, obtido a partir da especificação das portas de fluxo contínuo (transmissor e apresentador) dos componentes envolvidos e dos parâmetros correspondentes a uma determinada qualidade de serviço (QoS) estabelecidos em XML no descritor de distribuição.

O componente do lado do produtor foi instalado, instanciado e configurado e aguarda requisições do componente do lado do consumidor da aplicação. Os passos subseqüentes são:

1. a aplicação requisita à classe que implementa a porta transmissora de fluxo uma conexão para iniciar um fluxo de mídia contínua entre um componente produtor e um consumidor. Para isto, ela fornece a referência da porta apresentadora ao método *connect* da porta transmissora;

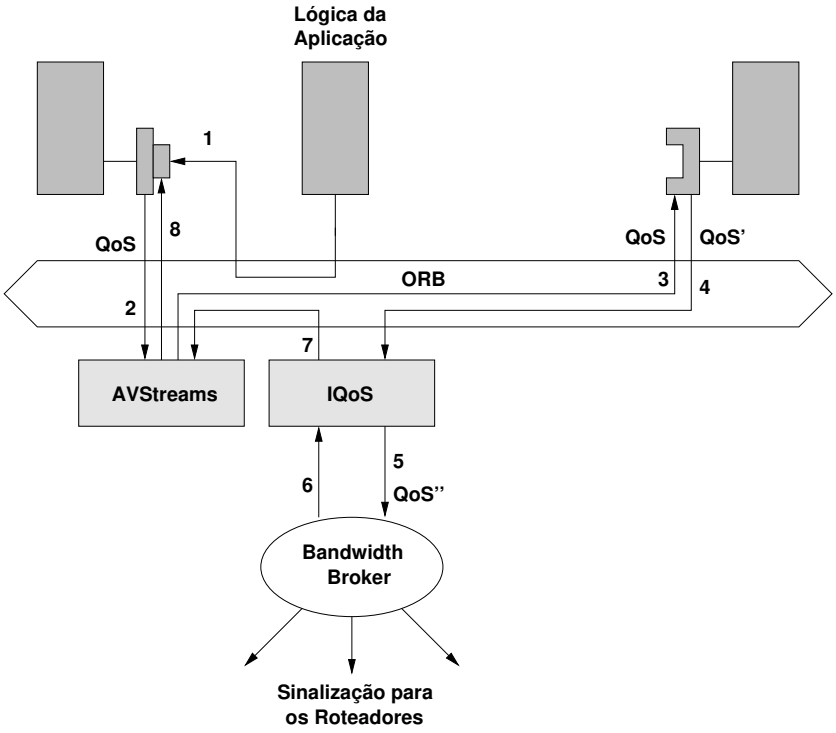

Fig. 4.21: Suporte a facilidades de QoS na plataforma CCM-tel.

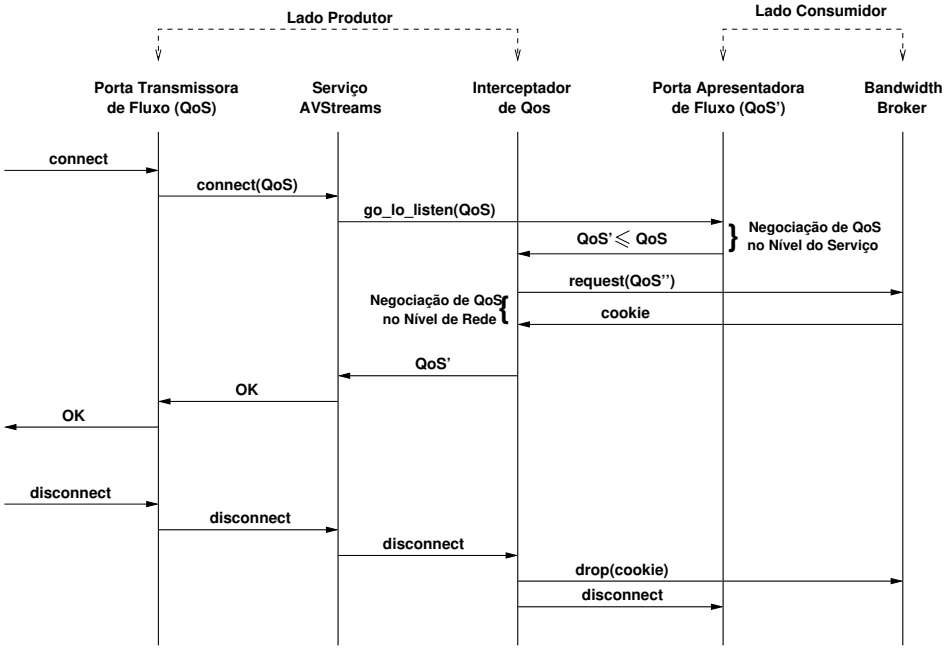

Fig. 4.22: Dinâmica para estabelecimento de fluxos de mídia com QoS.

2. a porta transmissora requisita ao serviço A/V Streams uma conexão entre o produtor e o consumidor com o nível de QoS estabelecido pelo produtor, conforme especificado no descritor de distribuição do contêiner que hospeda o componente produtor;

3. o serviço A/V Streams solicita ao consumidor (porta apresentadora de fluxo) um determinado *port* para o envio de mídia, fornecendo também a qualidade de serviço (QoS) que o produtor pode fornecer; por exemplo, 30 quadros por segundo, 24 bits, JPEG e resolução de 320x240.

4. o consumidor verifica a qualidade (QoS) e pode retornar uma determinada qualidade de serviço QoS' menor ou no máximo igual à QoS fornecida pelo produtor, por exemplo 15 quadros/seg, 24 bits, JPEG, 320x240. Esta negociação ainda está localizada no nível do serviço A/V Streams;

5. o interceptador de QoS intercepta o retorno da requisição fornecida pelo consumidor. Os parâmetros de QoS de alto nível (especificados pelo produtor e consumidor) são traduzidos para parâmetros de rede (QoS"), notadamente banda. Por exemplo, 70000bps. Em seguida, o interceptador interage com o *Bandwidth Broker* requisitando QoS". Este verifica se a requisição pode ser atendida pela rede (autenticação, disponibilidade de recursos, etc.)

6. o *Bandwidth Broker* retorna um identificador (*cookie*) para esta requisição;

7. o interceptador de QoS armazena este *cookie* e retorna;

8. o processamento das operações (itens 3, 2 e 1) se completa.

Analogamente, quando a desconexão das portas transmissora e apresentadora de fluxo ocorre, o interceptador de QoS desfaz a reserva junto ao *Bandwidth Broker*.

4.3.5 Suporte a Adaptação Dinâmica na Plataforma CCM-tel

A plataforma CCM-tel suporta adaptação dinâmica de contêineres de duas formas:

1. por propriedades definidas pelo contêiner;

2. por instalação de código ou agentes móveis no contêiner.

Os atributos definidos no descritor de distribuição são agregados em um elemento de configuração (conjunto de propriedades) mantidos pelo contêiner. De forma semelhante às propriedades para controle de QoS descritas anteriormente e tal como ocorre com as propriedades de configuração de componentes, as propriedades mantidas pelo contêiner também podem ser alteradas e monitoradas em tempo de execução. Uma aplicação pode instalar monitores de propriedades no contêiner e agir

face a alterações nestas propriedades. Por exemplo, quando um parâmetro de qualidade de serviço se altera, o objeto monitorando esta propriedade pode desconectar as portas de fluxo e reconectá-las com o novo parâmetro de QoS. O monitor de propriedades pode ser um agente móvel instalado no contêiner para este fim.

Os agentes móveis podem também sensoriar o ambiente onde foram instalados e executar certas ações em tempo de execução. Exemplos de configuração dinâmica propiciados por agentes móveis incluem:

- incorporar novas funcionalidades ao contêiner: o agente móvel pode adicionar dinâmicamente novos serviços ou monitorar certas funções do contêiner tais como funções de QoS e segurança;
- monitorar o ambiente e adaptar o contêiner: o agente móvel pode monitorar o ambiente onde o contêiner foi instalado e adaptá-lo a este ambiente. Por exemplo, o agente pode monitorar a taxa de transferência da rede (por exemplo, via SNMP) e reconfigurar os parâmetros de QoS de acordo com a taxa de transferência monitorada;
- especializar o contêiner pós-instalação: o agente pode alterar parâmetros de instalação ou configuração, por exemplo, utilizar capturadores e apresentadores de mídia capazes de operar em redes com alta taxa de erro (por exemplo, redes sem fio);
- especializar um componente: o agente pode alterar o comportamento de um componente alterando suas propriedades (por exemplo, alterando uma referência armazenada em um receptáculo do componente) ou processando alguma de suas funções (através de delegação);
- gerenciar o contêiner: o agente pode representar uma entidade de gerência, por exemplo, para identificar alguns componentes e as suas conexões.

4.3.6 Elementos Essenciais Implementados na Plataforma CCM-tel

Nesta seção, faz-se uma breve descrição das soluções tecnológicas adotadas na implementação da plataforma CCM-tel para os elementos essenciais CM-tel, descritos na Seção 2.2.

Padrão para Especificação do Comportamento de Componentes

O modelo CM-tel define como o comportamento de um componente CM-tel é descrito por:

- um padrão de interação, descrito na Seção 3.2.2;

- uma especificação do componente por uma notação neutra utilizando a linguagem de notação UML ou a linguagem XML (ou seja, não utiliza uma notação dependente de tecnologia tal como a linguagem IDL ou interface Java);
- interfaces (contrato de uso):
  - a natureza sintática dos aspectos funcionais é provida pelas interfaces IDL geradas pela plataforma CCM-tel;
  - a natureza semântica (comportamental) dos aspectos funcionais é provida por restrições em UML e em XML (invariantes, pré- e pós-condições).
- aspectos não-funcionais (contrato de realização):
  - suporte a fluxos de áudio e vídeo com facilidades de qualidade de serviço; a agentes móveis; mecanismos para a adaptação do ambiente de execução; e, extensões por meio de interceptadores portáveis;
  - aderência ao modelo de projeto garantido pela geração automática de código.
- documentação do componente realizada por arquivo em XML[8].

Esquema de Nomes Padronizado

A plataforma CCM-tel utiliza o serviço de nomes do CORBA para o registro e acesso a componentes, agentes móveis e contêineres, que é feito por meio do registro do localizador de fábricas (*HomeFinder*) no serviço de nomes.

O padrão para nomes CCM-tel define nomes globais únicos para a identificação de componentes, agentes móveis, fábricas de componentes e de agentes móveis e localizador de fábricas, onde:

- o nome do localizador de fábricas (*HomeFinder*) é registrado no serviço de nomes e representa o nome de uma instância de um contêiner;
- o nome da fábrica de componentes (*ComponentHome*) é registrado no localizador de fábricas do contêiner;
- o nome da fábrica de agentes móveis (*AgentHome*) é registrado no localizador de fábricas;
- cada instância de um tipo de componente possui uma chave arbitrária (String), que é registrada na fábrica do componente;

[8]No momento, esta documentação consiste apenas na especificação do componente em XML.

- cada instância de um tipo de agente possui um nome (String), que é registrado na agência.

A Figura 4.23 apresenta o espaço de nomes que ilustra a hierarquia de nomes utilizada pela plataforma CCM-tel para a localização (obtenção da referência) de componentes e agentes móveis a partir dos localizadores de fábricas. Note que utilizando esta hierarquia para a obtenção de referências de componentes e de agentes móveis é possível a realização de um filtro de segurança (baseado, por exemplo, em interceptadores), onde são verificados os possíveis acessos a estes elementos.

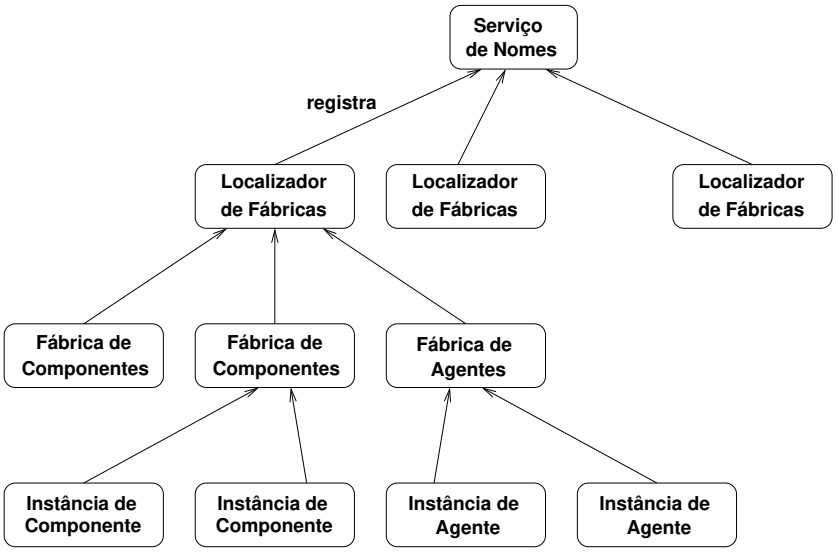

Fig. 4.23: Hierarquia de nomes na plataforma CCM-tel.

Padrão para Metadados

A plataforma CCM-tel define os conceitos de navegação para componentes, onde a partir da interface equivalente do componente é possível acessar as suas demais interfaces, bem como para agentes móveis, onde é possível a partir da agência acessar as referências dos agentes hospedados.

A plataforma CCM-tel possibilita gerar entradas para os repositórios de interface e de implementação CORBA para a obtenção de informações adicionais sobre as interfaces e os seus métodos. Em adição, está previsto uma extensão ao modelo onde uma interface de introspecção do componente fornece um documento XML com estas informações. Este documento será útil para que ferramentas de distribuição conheçam informações sobre um determinado componente após a sua instalação.

Padrão para Interoperabilidade

A plataforma CCM-tel obtém interoperabilidade por meio do CORBA/IIOP, padrão usado para comunicação entre componentes. Por ser uma solução independente de linguagem, permite apenas a

passagem de objetos passíveis de serem definidos em IDL, em invocações de métodos definidos na interface do componente.

Padrão para Especialização

A plataforma CCM-tel define especialização de componentes e contêineres pelo elemento de configuração (propriedades). Tipicamente, as propriedades são ativas podendo ser configuradas no momento de instanciação de componentes e alteradas em tempo de execução.

Padrão para Composição

A plataforma CCM-tel define um padrão para composição que suporta as três principais formas de composição [34], que são comumente definidas pelas tecnologias de componentes existentes atualmente:

- técnica de composição orientada-a-objetos, ou seja, os objetos podem ser compostos (no sentido de usados juntos), o que é proporcionado pela arquitetura CORBA;

- técnica de composição orientada-a-conexão, que é proporcionada pelo modelo CM-tel pela interconexão das portas de componentes.

- técnica de composição contextual, por meio do contêiner CCM-tel. Um contêiner encerra completamente as instâncias de componentes e de agentes móveis que ele hospeda. Além disso, um contêiner está implícitamente conectado a todas as instâncias hospedadas e este pode interceptar e gerenciar todas as comunicações que chegam e saem, por exemplo, tratar aspectos relacionados à segurança. Ou seja, todas as instâncias hóspedes se beneficiam uniformemente dos aspectos não-funcionais fornecidos e estão restritas às políticas e propriedades impostas pelo seu contêiner.

Padrão para Suporte à Evolução

A plataforma CCM-tel permite que mudanças em um componente (substituição por um outro componente que contenha uma implementação mais atual com novas interfaces e preservando as interfaces existentes; ou evolução com a atualização de novas versões de uma mesma implementação) sejam realizadas sem qualquer efeito sobre os demais componentes. Porém, a aplicação é responsável pela atualização das novas referências e a destruição das referências removidas (por exemplo, atualizações em receptáculos que abrigam a referência do componente antigo).

Padrões para Empacotamento e Distribuição

A plataforma CCM-tel utiliza arquivos no formato nativo da linguagem Java (tipo JAR) para o empacotamento das classes necessárias à distribuição de componentes e de contêineres.

A plataforma CCM-tel utiliza descritores de distribuição expressos na linguagem XML. Além de utilizar um descritor de montagem (*assembly*), como no CCM, o modelo CM-tel (e, conseqüentemente a plataforma CCM-tel) permite que o descritor de montagem possa ser obtido diretamente a partir de diagramas UML, de colaboração e de distribuição (esta facilidade encontra-se em desenvolvimento).

4.4 Avaliação da Plataforma CCM-tel

Em uma avaliação inicial da plataforma CCM-tel constatam-se alguns pontos de relevância. Aspectos importantes dos modelos existentes foram contemplados, particularmente os aspectos do modelo de componentes especificado pelo OMG no âmbito da arquitetura CORBA (CCM). Entretanto, adotaram-se algumas simplificações visando diminuir a complexidade e facilitar o seu uso, além da adição de aspectos disponibilizados pelo modelo CM-tel conforme discutido no Capítulo 3, não encontrados em outros modelos de componentes e que são fundamentais para algumas categorias de aplicações. Em resumo, a plataforma CCM-tel é similar às plataformas que implementam o modelo de componentes CCM nos seguintes aspectos:

- é uma plataforma neutra, portanto os componentes podem ser implementados utilizando diferentes linguagens de programação e executados em diferentes sistemas operacionais;
- os componentes são configurados dinâmicamente por meio de propriedades;
- os componentes podem emitir mensagens de notificação de eventos (produzir e consumir eventos);
- os componentes podem ser persistentes e participar de transações atômicas (embora estes serviços ainda não tenham sido implementados na plataforma CCM-tel);
- os componentes podem ser reconfigurados dinâmicamente pela aplicação (mas neste momento, com nenhum suporte à verificação da integridade do sistema, por exemplo a destruição de um componente que está sendo utilizado por outros);
- os descritores de distribuição são especificados em XML;

- têm uma estrutura similar, inclusive em relação a outros modelos de componentes existentes tais como EJB e COM+, como por exemplo fábrica de componentes, elementos de conexão, empacotamento, distribuição, contêiner (ambiente de execução), capacidade de geração de código a partir de uma especificação e separação dos aspectos funcionais e não-funcionais.

Entretanto, algumas diferenças entre a plataforma CCM-tel e as plataformas CCM são dignas de nota:

- a plataforma CCM-tel é mais simples do que as plataformas CCM. Com relação a este aspecto, chamamos a atenção para o fato de que isto ocorre porque derivamos um modelo de componentes menos complexo. A nossa intenção não foi a de implementar o modelo CCM ou parte deste. Em vez disso, optamos por manter apenas os aspectos mais importantes do CCM e adicionamos algumas extensões que atendem a uma extensa classe de aplicações emergentes;

- na plataforma CCM-tel, a linguagem IDL é preservada, sem extensões, o que significa que as arquiteturas de suporte ao modelo podem ser construídas a partir de implementações CORBA atuais; já o CCM estende a linguagem IDL para a adição de componentes, necessitando de desenvolvimento de novos compiladores;

- na plataforma CCM-tel, os componentes expõem interfaces de manipulação de fluxos de mídia contínua (*streams*), capazes de produzir, transferir e consumir fluxos de mídia contínua tais como áudio e vídeo; o CCM não disponibiliza esta interface;

- na plataforma CCM-tel, os componentes são especificados utilizando a linguagem XML ou UML; no CCM os componentes são especificados utilizando uma extensão da linguagem IDL;

- na plataforma CCM-tel, além dos componentes, os contêineres são configurados dinâmicamente por meio de propriedades;

- na plataforma CCM-tel, os contratos contemplam os aspectos funcionais (sintáticos e comportamentais) e os aspectos não-funcionais (qualidade de serviço para interfaces *stream*, mobilidade, segurança e adaptação dinâmica de recursos). Na plataforma CCM não existe uma forma declarativa de especificar os aspectos funcionais semânticos de contratos, contendo apenas o aspecto funcional sintático em IDL (contrato entre o componente e o seu cliente e de *callback* com o seu contêiner).CCM proporciona, pelos descritores de distribuição, os aspectos não-funcionais (segurança, transações e persistência);

- CCM-tel provê suporte para código móveis; CCM não provê tal suporte;

- CCM-tel fornece a possibilidade de construção de plataformas para equipamentos móveis de pequeno poder computacional; isto não é fornecido por nenhuma outra plataforma de componentes existente;

- na plataforma CCM-tel, o contêiner possui uma arquitetura orientada ao suporte dos aspectos não-funcionais que não foram contemplados nos modelos de componentes existentes;

- CCM-tel possibilita a declaração explícita em XML (expresso por meio de contratos) da especificação semântica dos aspectos funcionais relacionados às dependências de contexto de cada componente com outros componentes e com seu ambiente de execução, o que direciona e facilita as fases de empacotamento e distribuição (este suporte às dependências explícitas é importante para a reconfiguração em tempo de execução e prescreve as implicações quanto a remoção ou substituição de um componente);

- CCM-tel disponibiliza mecanismos para adaptação dinâmica de componentes e contêineres, o que não é contemplado pelo modelo CCM;

- CCM-tel adiciona reconfiguração como uma forma de mudar o comportamento em tempo de execução. Para isto fornece o suporte à interceptação durante a requisição de métodos por meio de interceptadores CORBA e de esqueletos (*templates* de invariantes, pré- e pós-condições), permitindo a inspeção de detalhes internos e a adição de código para o monitoramento (por exemplo, de políticas de reconfiguração de recursos tais como dispositivos, de qualidade de serviço, entre outras) e para a adição de novos comportamentos que não requerem uma reconfiguração dos componentes existentes no ambiente de execução (por exemplo, fiscalizar a segurança de acesso na requisição de métodos).

4.5 Considerações Finais

Este capítulo contemplou a plataforma de suporte ao modelo CM-tel para as tecnologias CORBA e Java. Para isto, foi definido um mapeamento do modelo para estas tecnologias (mapeamento CCM-tel), bem como estabelecidos os detalhes internos da plataforma. Uma das aplicações desenvolvidas utilizando a plataforma CCM-tel, o laboratório virtual REAL, será apresentada no próximo capítulo.

CCM-tel é capaz de, a partir de uma especificação em UML ou XML, gerar código de suporte aos seguintes requisitos funcionais e não-funcionais que as aplicações emergentes demandam da plataforma:

- suporte à comunicação multimídia com facilidades de qualidade de serviço (QoS) para fluxos

de áudio e vídeo. Este requisito não-funcional é importante para aplicações telemáticas, como por exemplo para laboratórios virtuais;

- suporte à mobilidade de código e agentes móveis. Este requisito não-funcional é importante para ubiqüidade, computação móvel, configuração dinâmica e adaptação ao ambiente de execução;
- capacidade de adaptação dinâmica ao ambiente de processamento (terminal, rede, periféricos, etc.) e aos usuários;
- o acesso aos serviços CORBA disponibilizados, o que não exige do desenvolvedor o conhecimento dos mesmos (interfaces, formas de utilização, etc.) nem a determinação de suas dependências (de outros serviços, por exemplo);
- suporte aos aspectos funcionais sintáticos dos requisitos funcionais da aplicação pela geração automática do código dos *stubs* e *skeletons*. O uso deste código por parte da aplicação garante que os métodos remotos que implementam a lógica da aplicação serão corretamente invocados.

Capítulo 5

Laboratório Virtual REAL

Neste capítulo, apresentamos o desenvolvimento orientado a componentes de uma aplicação telemática construída de acordo com o modelo de componentes CM-tel. Esta aplicação, desenvolvida sobre a plataforma CCM-tel, utiliza comunicação multimídia, código móvel (agente móvel) e adaptação dinâmica de QoS para a validação deste modelo. O protótipo implementado, denominado REAL (*Remotely Accessible Laboratory*), é um laboratório virtual para o acesso remoto a equipamentos robóticos através da Internet. REAL oferece uma infra-estrutura que permite a um usuário remoto manipular sofisticados robôs móveis por três modos de interação: básico, avançado e assistido. Estes modos são escolhidos de acordo com a experiência do usuário na utilização dos robôs.

5.1 Laboratórios Virtuais

No domínio de aplicações telemáticas, os laboratórios virtuais, também denominados laboratórios de acesso remoto, despertam interesse especial. Laboratórios virtuais proporcionam um mecanismo de interação que permite aos usuários remotos utilizar os recursos do laboratório através de uma rede de computadores. Deste modo, laboratórios virtuais permitem o acesso remoto a materiais educacionais e laboratoriais existentes em locais distintos, muitas vezes distantes daquele onde encontra-se o usuário que os acessa. Os usuários podem planejar, executar, testar e validar experimentos práticos remotamente, além de coletar dados experimentais para análise e processamento a posteriori. Dependendo dos tipos de experimentos a serem suportados, estes laboratórios precisam prover algum mecanismo de telepresença para "transportar" o usuário para o laboratório. Por exemplo, na área de robótica, um laboratório virtual deve possibilitar que um usuário observe o robô durante uma missão por ele submetida. Isto pode ser proporcionado ao usuário, em seu ambiente computacional, através de canais de vídeo (caso necessário, também de áudio) para a visualização de imagens de vídeo, mapas bidimensionais ou uma combinação destes. Para permitir esta "imersão" de usuários remotos

em um laboratório físico, esta classe de aplicações demanda redes de alta velocidade e comunicação multimídia com facilidades de qualidade de serviço.

Laboratórios virtuais constituem uma importante ferramenta educacional e experimental aproximando grupos de pesquisa, permitindo a pesquisadores, educadores e estudantes o compartilhamento e comunicação de dados, documentos, sons, imagens de vídeo e promovendo a integração de seus recursos computacionais pela interoperabilidade de dados e aplicativos. Eles são, portanto, uma forma de compensar a falta de recursos e de equipamentos em universidades e centros de pesquisa, além de fornecer o suporte a aprendizagem a distância. Dentre os muitos benefícios proporcionados por estes tipos de laboratórios destacam-se:

- melhor interação entre grupos de pesquisa com o compartilhamento de dados experimentais e equipamentos, até então restritos a um número limitado de pessoas, por múltiplos usuários em diferentes localidades geográficas;
- provê uma base experimental comum aos diferentes grupos de pesquisa;
- o aumento do acesso a equipamentos de pesquisa por parte de pesquisadores e estudantes. Estes equipamentos têm elevado custo para muitas instituições que desta forma complementam as suas infra-estruturas laboratoriais;
- a experiência prática, elemento fundamental das áreas de educação e pesquisa, onde os recursos necessários para a aquisição de equipamentos ou construção de experimentos no estado da arte estão além da possibilidade de muitas instituições de ensino e pesquisa no Brasil;
- o estabelecimento de padrões científicos de validação em diversas áreas como, por exemplo, na área de robótica onde os pesquisadores podem demonstrar, de forma experimental, métodos avaliados apenas em ambientes simulados;
- a capacitação de pesquisadores na participação de experimentos distribuídos geograficamente.

O Centro de Pesquisas Renato Archer (CenPRA) iniciou em 1996 o projeto de um laboratório virtual de robótica. Esta experiência converteu-se na implementação de uma primeira versão de um laboratório de robótica e visão computacional denominado REAL (*Remotely Accessible Laboratory*), o primeiro laboratório virtual na área de robótica desenvolvido no país. Este trabalho foi proposto como tema de uma dissertação de mestrado no Instituto de Computação da UNICAMP em 1998 [109]. Esta versão foi implementada como uma aplicação WWW (*World Wide Web*) [110, 111]. Com o desenvolvimento desta primeira versão tornou-se clara a necessidade de uma nova infra-estrutura que atenuasse as complexidades de desenvolvimento dos serviços providos pelo laboratório virtual

REAL, fornecendo o suporte a aspectos relacionados ao desenvolvimento e ao controle de acesso a estes serviços a partir de ambientes heterogêneos, às restrições de tempo real, à proteção e segurança dos equipamentos robóticos, bem como facilidades de qualidade de serviço.

Desde 1997, um acordo de cooperação [112][113][114] foi estabelecido entre o CenPRA e a Faculdade de Engenharia Elétrica e de Computação da UNICAMP com o objetivo de construir plataformas de *middleware* baseadas em padrões abertos para o desenvolvimento de serviços telemáticos e de computação móvel. Este acordo resultou na implementação da segunda versão do laboratório REAL, incorporando novas tecnologias tais como a arquitetura de objetos distribuídos CORBA. Esta versão incorporou funcionalidades extras como, por exemplo, o controle do robô pelo usuário por algoritmos de navegação desenvolvidos pelo próprio usuário. Adicionalmente, permitiu uma melhor imersão do usuário no laboratório e o acompanhamento de uma missão através de dois canais de vídeo em tempo real. Um primeiro canal, alimentado por uma câmera panorâmica, envia imagens de vídeo do ambiente onde se localizam os robôs móveis e um segundo canal, alimentado por uma câmera embarcada no computador de bordo do robô móvel, envia imagens de vídeo a partir deste robô. Esta implementação foi apresentada na mostra de Ciência e Tecnologia para o Desenvolvimento (CIENTEC) [44], realizada em 2001 na UNICAMP. A partir de 2000, este esforço resultou no desenvolvimento da terceira versão (atual) do REAL, uma evolução da anterior e utilizando o modelo CM-tel e a plataforma CCM-tel. Esta nova versão do laboratório virtual é apresentada nas próximas seções.

5.2 O Laboratório Virtual REAL

O laboratório virtual REAL [39] permite que usuários utilizem remotamente um robô móvel como se estivessem presentes fisicamente no laboratório do CenPRA. REAL oferece o acesso por meio da Internet a robôs móveis de alta qualidade, os robôs XR4000 e NOMAD200, para pesquisadores e estudantes.

As referências [115][116][117][77][76] descrevem alguns projetos no campo de laboratórios virtuais e de tele-robótica. REAL compartilha muitas das características encontradas nestes projetos, incluindo interatividade através da WWW e suporte de vídeo para a supervisão da missão. No entanto, REAL difere destes projetos principalmente na sua arquitetura de *software*. Em vez de implementar uma arquitetura de *software* baseada em objetos distribuídos, REAL emprega uma arquitetura de *software* baseada em componentes, desenvolvida utilizando a plataforma CCM-tel. O desenvolvimento do REAL emprega unicamente padrões abertos, tais como CORBA, UML, WWW com suas tecnologias relacionadas (HTTP, HTML, XML, JSP, *Servlets*) e a linguagem Java.

A infra-estrutura fornecida no domínio do laboratório virtual REAL, conforme ilustrado pela Figura 5.1, consiste dos seguintes elementos:

- dois robôs móveis (XR4000 e NOMAD200), manipulados pelos usuários por meio de seus terminais remotos, que devem operar sem nenhuma conexão física para navegar livremente pelo ambiente;

- um sistema de aquisição de imagens de vídeo obtidas da câmera a bordo do robô e da câmera panorâmica do laboratório, bem como a sua transmissão contínua ao usuário através da Internet. A transmissão ocorre durante a operação do robô, para que o usuário remoto possa acompanhar o andamento do seu experimento;

- uma rede local (LAN - *Local Area Network*) integra um conjunto de processadores, os diversos servidores e os equipamentos do laboratório, tais como um roteador e dois pontos de acesso para comunicação através de rede sem fio;

- duas redes sem fio (WLAN - *Wireless Local Area Network*) conectam os processadores embarcados de controle e visão do robô à rede local;

- uma interface homem-máquina que permite o acesso ao serviço, programação do robô, recebimento dos dados de sua operação e término do serviço. O usuário remoto utiliza este serviço sem a necessidade de instalação de *hardware* e *software*, sendo que este acesso transparente é realizado a partir de qualquer computador conectado à Internet.

Os robôs têm um processador de bordo dedicado ao seu controle. Este processador usa uma rede sem fio de 1.6 Mb/s para se comunicar com o processador de controle conectado à LAN. O processador de controle executa o *software* que interage diretamente com o robô, como por exemplo, processos *deamons* de segurança, código fornecido pelo usuário, etc. Um computador portátil, localizado na base do robô, executa a captura de imagens de vídeo da câmera de bordo do robô e está conectado ao ponto de acesso da rede sem fio por uma conexão de 11 Mb/s.

Um servidor Web responde às requisições enviadas pelos usuários e disponibiliza um conjunto de *applets* assinados, páginas dinâmicas (JSP - *Java Server Pages*) e de *servlets* Java para a implementação da gerência de acesso e de uso do serviço. A infra-estrutura de suporte aos serviços consiste de componentes e contêineres CCM-tel empacotados no formato JAR, bem como páginas HTML. Esta infra-estrutura está disponibilizada em servidores implementados em plataformas e sistemas operacionais heterogêneos, tais como microcomputadores e estações de trabalho executando sistemas operacionais Linux e Windows. Os servidores de acesso aos robôs utilizam implementações na sua linguagem nativa (C++).

No domínio do usuário remoto, REAL provê um acesso uniforme aos serviços com interfaces para a subscrição, acesso e uso dos serviços disponibilizados pelo laboratório. Deste modo, o usuário precisa somente de um ambiente computacional capaz de abrigar um navegador Web (por exemplo,

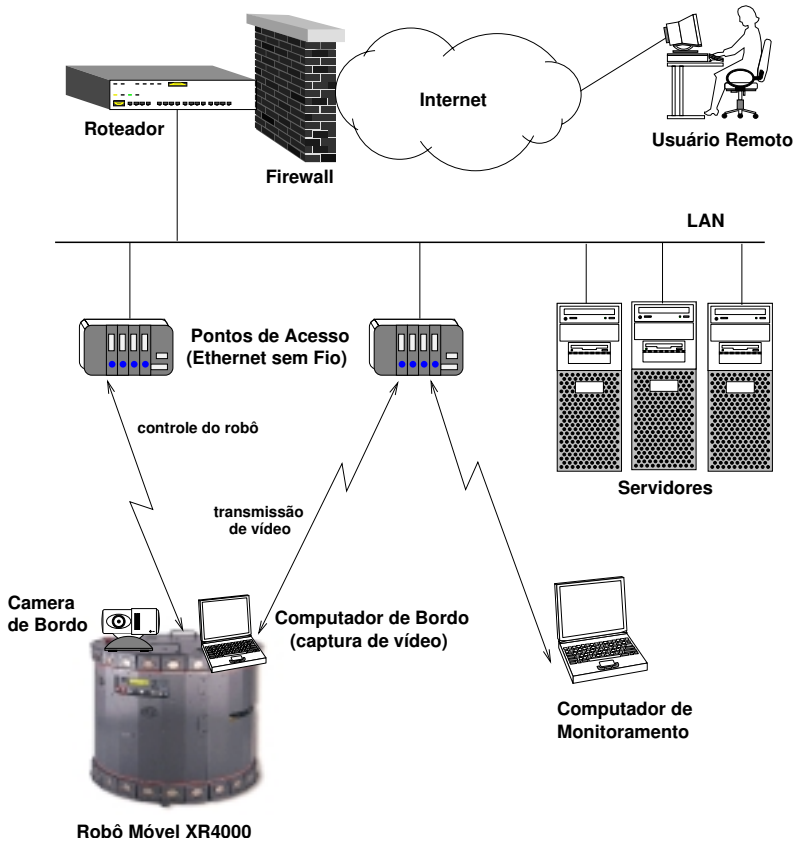

Fig. 5.1: Infra-estrutura do laboratório virtual REAL.

Netscape 7, Internet Explorer ou Konqueror), que suporte a máquina virtual Java 2 (eventualmente por um *Plug-in* Java) com capacidade para executar *applets*. A infra-estrutura de suporte aos serviços é instalada automaticamente no ambiente do usuário, sempre que este decida utilizá-lo. Projetos como o REAL muito se beneficiarão da utilização de redes com velocidades maiores, como por exemplo Internet-2. No caso de terminais com baixo suporte computacional o modelo CM-tel provê a possibilidade da aplicação utilizar uma segunda plataforma construída a partir de um mapeamento do modelo CM-tel para serviços Web e para a linguagem Java (plataforma WSCM-tel) para executar em dispositivos móveis.

Ao acessar os serviços providos pelo REAL, os usuários remotos poderão realizar as seguintes atividades:

- executar funções administrativas (cadastramento, subscrição aos serviços, reserva de horários, etc);
- conduzir experimentos em vários campos da robótica, por exemplo, navegação autônoma, mapeamento do ambiente e planejamento de missões;

- manipular um dos robôs por meio de comandos ou de um *joystick*, no caso de usuários leigos ou iniciantes em robótica;
- desenvolver, compilar, submeter o código ao robô, executar uma missão e testar algoritmos de navegação, no caso de usuários mais experientes em robótica;
- observar o experimento realizado por um instrutor;
- acompanhar a missão através de imagens de vídeo.

5.2.1 Arquitetura de Software do REAL

REAL considera e implementa laboratórios virtuais como serviços telemáticos complexos e não como simples aplicações WWW. Isto se deve à semelhança entre estes serviços e os serviços de telecomunicações nos seguintes aspectos:

- os papéis do provedor e do usuário do serviço são bem definidos;
- requerem um controle de cadastramento, de subscrição, de acesso e de uso de serviços a partir de uma reserva de horário de uso, verificação de segurança e, em alguns casos, de tarifação;
- requerem estratégias para gerência dos serviços;
- demandam uma sessão de comunicação em tempo real com suporte a fluxos de mídia contínua com facilidades de qualidade de serviço para o suporte a telepresença em laboratórios virtuais;
- aspectos de segurança e privacidade devem ser amplamente considerados.

De forma a caracterizar as aplicações telemáticas e ubíquas, identificamos um modelo de serviço que as diferencie das demais aplicações na Internet. A nossa proposta para a arquitetura de *software* do REAL utiliza um modelo simplificado baseado no modelo de serviço de telecomunicações proposto pelo consórcio TINA (*Telecommunications Information Network Architecture*) [9]. O consórcio TINA, encerrado em dezembro de 2000, produziu um conjunto de especificações de uma arquitetura aberta para sistemas de telecomunicações e baseada no modelo de referência RM-ODP. Dentre os conceitos e princípios da arquitetura de serviço TINA destacam-se: o conceito de sessão; a separação lógica entre as aplicações e o ambiente em que são executadas; e a separação entre o acesso e uso do serviço.

TINA introduz o seguinte conceito de sessão: “uma sessão representa a informação usada por todos os processos envolvidos na provisão de um serviço pelo tempo de sua duração, garantindo uma visão coerente dos vários eventos e relacionamentos associados à provisão deste serviço“. Em

TINA, três sessões são definidas, cada uma correspondendo a um tipo de atividade: sessão de acesso, sessão de serviço e sessão de comunicação. A sessão de acesso suporta as interações necessárias para iniciar e gerenciar a comunicação entre o usuário e o provedor, além das necessárias para requisitar os serviços. A sessão de serviço suporta as interações que controlam, manipulam e gerenciam o comportamento do serviço e o fluxo de informação associado. A sessão de comunicação suporta a comunicação multimídia com QoS entre os componentes da aplicação.

A estratégia utilizada para o desenvolvimento dos serviços proporcionados pelo REAL foi a implementação simplificada de um modelo de serviço composto das três sessões inspiradas no modelo de serviços TINA. Estas sessões são implementadas segundo o modelo de componentes CM-tel. Desta forma, um conjunto de componentes CM-tel foi especificado para o laboratório REAL e agrupado em três categorias: componentes de acesso, de serviço e de comunicação. Cada categoria suporta uma sessão correspondente.

No modelo de serviço proposto para o laboratório virtual REAL, os serviços disponibilizados foram construídos a partir da composição de componentes novos e do reuso de componentes existentes. Estes serviços podem ser distribuídos em plataformas heterogêneas e facilmente configurados e gerenciados, não havendo necessidade de instalação manual de *software* para a execução dos serviços no computador do usuário.

5.3 Sessão de Acesso

A sessão de acesso é responsável por estabelecer uma relação segura e gerenciável entre entidades pertencentes a domínios administrativos diferentes (usuário e provedor do serviço). Esta relação implica, genericamente, em uma relação contratual entre os domínios envolvidos, permitindo ao usuário o acesso aos serviços disponibilizados pelo provedor de serviços.

O projeto REAL desenvolveu uma sessão de acesso para serviços implementados segundo o modelo CM-tel. Esta sessão proporciona os mecanismos de suporte às funções de subscrição, reserva de recursos e gerência do serviço. A partir de uma sessão de acesso podem ser iniciadas muitas sessões de serviço que estarão vinculadas à sessão de acesso.

A Figura 5.2 ilustra os principais elementos da arquitetura da sessão de acesso utilizada pelo REAL. A sessão de acesso é implementada por *Java Servlets*, páginas dinâmicas JSP, componentes e contêineres CM-tel. Os *servlets* interagem com um sistema gerenciador de base de dados que disponibiliza um *driver* compatível com JDBC. Na base de dados são armazenadas informações sobre os serviços disponibilizados, os usuários cadastrados e reservas de horário para uso dos serviços. Um *servlet* específico interage com um componente CCM-tel, o componente de sessão descrito a seguir. Finalmente, as páginas dinâmicas são utilizadas para interação com os usuários do serviço.

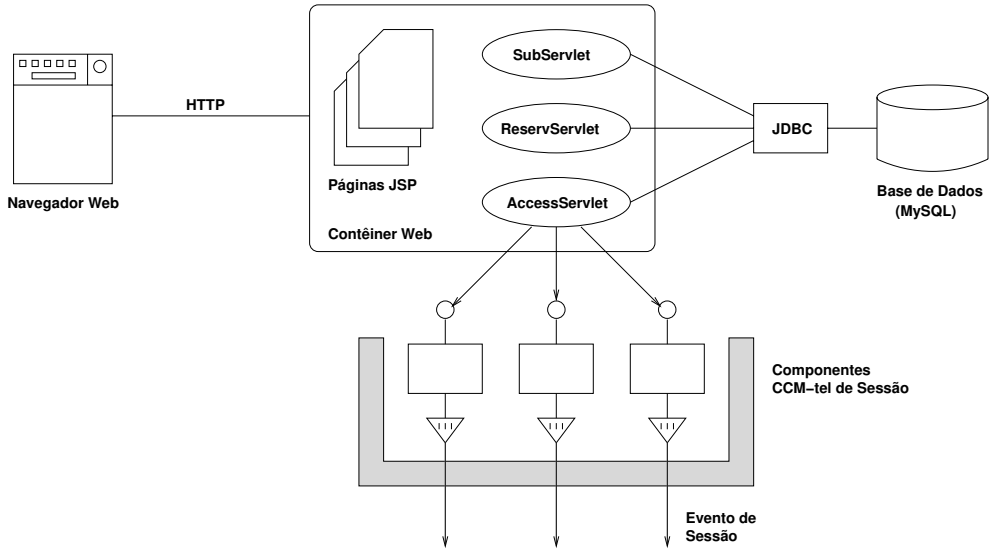

Fig. 5.2: Arquitetura da sessão de acesso do REAL.

5.3.1 Funções da Sessão de Acesso

As três funções proporcionadas pela sessão de acesso são disponibilizadas por interfaces gráficas apresentadas no navegador do usuário. Estas interfaces, implementadas por meio de páginas JSP, atuam como um cliente do provedor iniciando e gerenciando a comunicação do usuário com o laboratório virtual. As funções de subscrição de usuários, reserva de recursos e de gerência de serviço (acesso e uso do serviço) são descritas a seguir.

Subscrição de Usuários

A subscrição de usuários é necessária para que estes possam usufruir dos serviços oferecidos pelo provedor dos serviços. Para isto, é disponibilizada uma interface para subscrição, a partir da qual o usuário fornece informações para o seu cadastramento. Estas informações consistem do fornecimento de seus dados, tais como nome completo, identificação do usuário, senha de acesso, telefone, instituição, posição, serviços subscritos e endereço eletrônico (*e-mail*). O *servlet SubServlet* (Figura 5.2) processa as informações submetidas pelo usuário, adicionando-as à base de dados caso isentas de erros. As informações de identificação e senha fornecidas serão solicitadas nas interações subseqüentes do usuário com o serviço.

Reserva de Recursos

Serviços como o REAL que necessitam de acesso exclusivo aos recursos (no caso o robô) demandam da sessão de acesso um mecanismo de reserva de recursos. A sessão de acesso implementada possui uma interface para reserva de recursos, pela qual o usuário pode reservar um intervalo de tempo para acesso ao serviço. No caso do REAL, a interação com o robô é permitida apenas para usuários com reserva de horário. O *servlet ReservServlet* (Figura 5.2) processa as informações de reserva de recursos submetidas pelo usuário, autenticando-o e verificando a existência de conflitos (horário já reservado). As reservas aceitas são armazenadas na base de dados.

Acesso ao Serviço

O acesso ao serviço consiste no carregamento do serviço no terminal do usuário. Este carregamento ocorre simplesmente pelo acesso à URL do serviço (usualmente a partir da página principal do provedor de serviços). O acesso à URL propicia o carregamento de páginas HTML contendo referências a *applets* e arquivos JAR. Os arquivos JAR contêm classes Java que implementam os componentes e contêineres do serviço no lado do usuário do serviço. Adicionalmente ao serviço, uma interface da sessão de acesso é apresentada ao usuário. Esta interface permite que o usuário use efetivamente o serviço. A separação entre carregamento e uso do serviço é uma vantagem principalmente para acessos de baixa velocidade, onde o usuário pode carregar o serviço sem que seu uso seja contabilizado durante o tempo de carregamento.

Uso do Serviço

Após o carregamento, o serviço pode ser efetivamente usado. Para tal, o usuário fornece identificação e senha na interface da sessão de acesso disponibilizada juntamente com o serviço. Esta interface envia a identificação e senha a um *servlet* da sessão de acesso (*AccessServlet* ilustrado na Figura 5.2), que irá autenticar o usuário e verificar se o mesmo possui horário reservado. O serviço somente se inicia caso estas verificações sejam bem sucedidas. O início do serviço consiste na interação deste *servlet* com o componente da sessão responsável por gerenciar o serviço (Figura 5.2). Este componente é o "ponto de contato" entre o serviço no lado do provedor e a sessão de acesso.

O componente de sessão expõe duas portas: uma faceta e um publicador de eventos. Existe uma instância deste componente para cada serviço gerenciado pela sessão de acesso. Instâncias deste componente são registradas no *Home* com o próprio nome do serviço que gerenciam. Ao receber uma requisição de acesso ao serviço, o *servlet AccessServlet*, após as verificações descritas acima, localiza a instância do componente de sessão responsável pela gerência do serviço e invoca o método *create_session* definido na faceta do componente. Este método atribui um identificador único de

sessão e ajusta um contador de tempo para o período de reserva que ainda resta ao usuário do serviço. O componente de sessão notifica, pela emissão de um evento tipo *start* na sua porta publicadora de eventos, os componentes de configuração de serviço conectados. Este evento desencadeará o processo de montagem da aplicação, conforme descrito na seqüência.

Uma vez terminada a interação do *servlet* com o componente, o *servlet* redireciona o processamento da requisição para uma página JSP que irá substituir a página da sessão de acesso utilizada para iniciar o serviço. Esta nova página possui um botão de encerramento do serviço e um script Java (Javascript) que confirma o uso do serviço a cada intervalo de 60 segundos (ambos pela submissão de um documento HTML ao *servlet AccessServlet*). O encerramento e a confirmação são processados pelo mesmo *servlet*. Este *servlet* invoca as operações de confirmação (*ping_session*) ou encerramento (*end_session*) de sessão na faceta do componente de sessão. A operação de confirmação reinicia o contador de tempo que monitora a inatividade da sessão.

A sessão pode ser encerrada em três situações:

1. encerramento pelo usuário: o usuário ativa o botão de encerramento da interface da sessão de acesso;

2. encerramento devido a inatividade: a inatividade é noticiada pela ausência de confirmação causada, por exemplo, pelo fechamento do navegador do usuário ou sobreposição da página do serviço;

3. encerramento por *timeout*: o período de tempo reservado para uso do serviço expirou.

Qualquer que seja o motivo do encerramento, o componente de sessão notifica, por meio de um evento tipo *end* na sua porta publicadora de eventos, os componentes de configuração de serviço conectados. Este evento desencadeará o processo de desmontagem da aplicação.

5.3.2 Interação entre as Sessões do Modelo de Serviço do REAL

As interações entre as três sessões definidas pelo modelo de serviço do REAL são importantes para a realização da gerência e controle dos serviços. Esta seção mostra a interação entre os componentes pertencentes a estas sessões para concretizar o estabelecimento e utilização dos serviços disponibilizados pelo laboratório virtual.

Interação Entre as Sessões de Acesso e Serviço

A Figura 5.3 ilustra uma visão geral dos componentes e seus contêineres envolvidos na interação entre as sessões de acesso e de serviço. A interação entre a sessão de acesso e a sessão de serviço

ocorre por meio de eventos propagados entre o componente de sessão e o componente de configuração do serviço. Ao receber um evento tipo *start*, o componente de configuração do serviço inicia o processo de configuração inicial do serviço, procedendo à instanciação e conexão dos demais componentes da sessão de serviço. A instanciação e conexão dos demais componentes da aplicação torna o serviço pronto para uso por parte do usuário. Analogamente, ao receber um evento tipo *end*, o componente de configuração do serviço desfaz as conexões entre os componentes da sessão de serviço destruindo determinados componentes (tipicamente aqueles instanciados no domínio do usuário). Nesta ação são liberados os recursos alocados durante o estabelecimento da sessão de serviço.

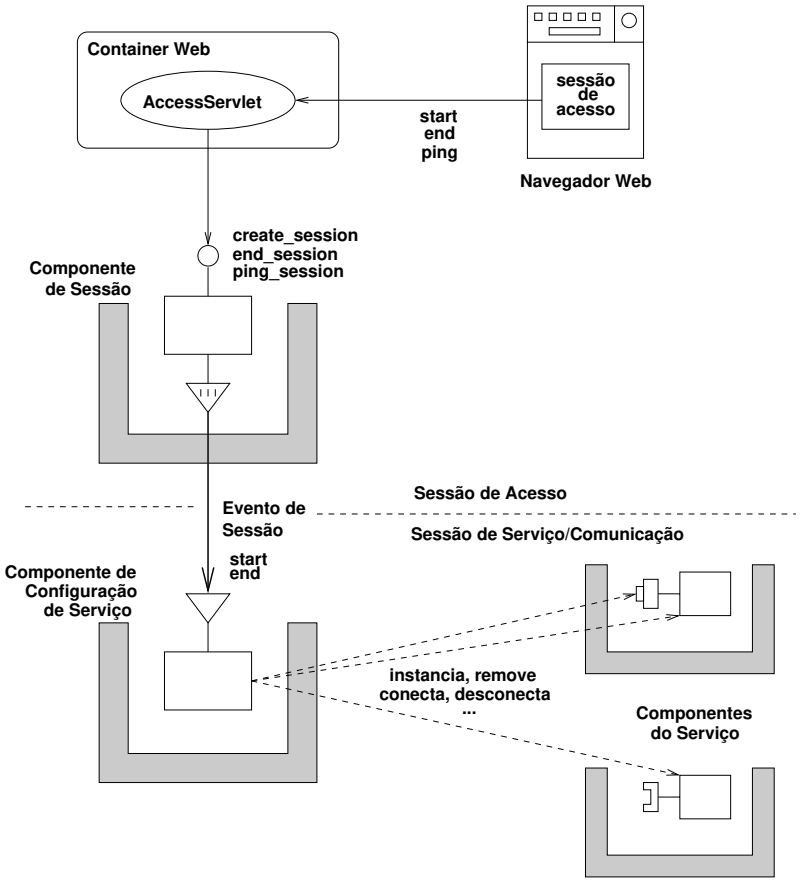

Fig. 5.3: Interação entre componentes das sessões de acesso e de serviço.

Para serviços multipartite, uma sessão de acesso pode abrigar vários usuários. É o caso do modo de interação assistido do REAL descrito na Seção 5.4.3. Para tais serviços, o componente de configuração do serviço deve ser capaz de proceder a montagem e desmontagem de componentes do serviço associados a um usuário específico. Tal situação ocorre quando um usuário ingressa ou abandona a sessão de serviço. A identificação do usuário e seu domínio estão presentes nos parâmetros do evento de início e término de sessão. Com esta identificação, o componente de configuração do serviço

é capaz de proceder à instanciação (destruição) e conexão (desconexão) de componentes para um usuário específico. Para serviços multipartite, um usuário específico não pode encerrar a sessão de acesso. Neste caso, a sessão de acesso é encerrada por *timeout* ou quando o último usuário abandona o serviço.

Interação entre as Sessões de Serviço e de Comunicação

Uma sessão de serviço possui uma sessão de comunicação associada que é composta de vários fluxos de mídia contínua. Os fluxos são produzidos através da interconexão de componentes produtores e apresentadores de mídia. Durante o tempo de duração da sessão de serviço, a sessão de comunicação tem natureza dinâmica quanto aos fluxos. Fluxos podem ser estabelecidos, suspensos, reiniciados e finalizados em qualquer instante de tempo, ter sua estrutura alterada (por exemplo, a inclusão de um consumidor em um fluxo ponto-multiponto), ou ter seus parâmetros modificados no tempo (por exemplo, a qualidade do vídeo pode se alterar em função do desempenho da rede).

A interação entre os componentes das sessões de serviço e de comunicação também ocorre por meio do componente de configuração associado ao serviço. Os componentes da sessão de comunicação (produtores e apresentadores de mídia contínua) são instanciados e conectados pelo componente de configuração do serviço durante o processamento do evento tipo *start*. Analogamente, os componentes da sessão de comunicação são desconectados e destruídos em resposta a eventos do tipo *end*.

5.3.3 Detalhes da Implementação da Sessão de Acesso

A implementação da sessão de acesso do REAL foi realizada na linguagem de programação Java utilizando a plataforma Java 2 SDK, versão 4.2. O código de implementação do componente de sessão CM-tel também foi gerado em Java pela plataforma CCM-tel e, a partir do descritor de distribuição, foi gerado um arquivo para a instalação do contêiner no domínio do provedor na forma de processo Java.

Para processamento de formulários e páginas HTML, utilizamos o servidor Apache Tomcat [118] no provedor de serviços. As informações cadastrais dos usuários, as reservas de horário e os serviços oferecidos pelo REAL são armazenadas em uma base de dados relacional gerenciada pelo MySQL [119].

5.4 Sessão de Serviço

A sessão de serviço do REAL proporciona os mecanismos para o suporte a execução, controle e gerência de serviços propiciados pelo laboratório virtual. Esta sessão implementa a lógica do serviço,

isto é, as funcionalidades que o serviço proporciona para os usuários.

Esta sessão não existe sem que haja uma sessão de acesso associada e previamente estabelecida, dado que o usuário só tem acesso aos serviços após estar autorizado pela sessão de acesso. Todo o controle de acesso é proporcionado pela sessão de acesso de forma independente do serviço. Quando uma sessão de acesso é encerrada, a sessão de serviço associada também é finalizada.

No REAL, uma sessão de serviço é estabelecida quando o usuário inicia um serviço e executa as ações disponibilizadas pelo laboratório. Para isto, foram desenvolvidas interfaces gráficas que são apresentadas em navegadores do usuário. Estas interfaces, implementadas como *applets* na linguagem Java, atuam como um cliente dos serviços do laboratório e fazem a comunicação com este laboratório, de forma transparente ao usuário.

Os componentes desenvolvidos para esta sessão permitem que usuários manipulem um dos robôs móveis por meio de três modos de interação: básico, avançado e assistido, descritos nas seções subseqüentes. Cada modo de interação é um serviço para a sessão de acesso. Cada sessão de serviço é implementada pela integração de componentes e de contêineres CM-tel, cuja interconexão é garantida pela plataforma CCM-tel.

Devido à complexidade dos modos de interação proporcionados, optamos por apresentar somente uma visão simplificada de apenas alguns dos seus principais componentes e de seus respectivos contêineres. As referências [40][30] fornecem detalhes da especifição, projeto e implementação dos modos de interação do REAL.

Os três modos de interação permitem que usuários observem no seu computador a movimentação do robô, através de imagens de vídeo recebidas das câmeras panorâmica e embarcada no robô. O componente que implementa as imagens de vídeo é o mesmo para todos os modos de interação, sendo reutilizado e passível de configuração (por exemplo, com parâmetros de QoS personalizados pelo desenvolvedor do serviço). A captura, transferência e apresentação do vídeo é responsabilidade da sessão de comunicação.

Adicionalmente, os modos de interação requerem um conjunto de medidas preventivas de segurança contra algoritmos de navegação e seqüências de movimentos mal-intencionados ou com falhas. Estas medidas foram implementadas utilizando-se um processo (*deamon*) de segurança de alta prioridade, que periodicamente pára o programa de navegação, examina os sensores de sonar do robô e, baseado na direção e velocidade do robô, determina se o robô se mantém a uma distância segura dos obstáculos. Se este for o caso, o programa de navegação continua a sua execução. Caso contrário, o processo de segurança pára o movimento do robô e termina o programa de navegação. Esta ação gera uma notificação de cancelamento de navegação para o usuário. Este processo de segurança é reutilizado em todos os modos de interação.

5.4.1 Modo de Interação Básico

O modo de interação básico é um serviço do REAL dedicado a usuários leigos ou com conhecimentos limitados na área de robótica, que queiram manipular os robôs por meio de teleoperação. Neste modo, os usuários interagem com o robô em um alto nível de abstração, em que o robô é visto como um veículo capaz de se mover e de seguir uma seqüência de comandos simples.

Este serviço do REAL fornece uma interface com duas formas simplificadas de interação com o robô, por meio das quais o usuário pode enviar comandos para o robô. Esta interface foi implementada como um painel que possui "abas" (*tabbed pane*) na borda superior, correspondendo cada uma delas a um conteúdo diferente do painel. A cada instante, apenas um desses conteúdos pode ser selecionado, clicando-se na aba correspondente. Assim, é possível acrescentar ao painel diversas formas diferentes de interação com o robô. O usuário pode escolher como quer interagir com o robô, selecionando uma das duas formas. Além disso, pode clicar em um botão para parar o robô ou para pedir instruções de auxílio. Caso o movimento solicitado seja impossível de ser realizado, por exemplo, devido a presença de obstáculos na rota do robô, o comando não é executado.

A primeira forma de interação, ilustrada pela Figura 5.4, disponibiliza um conjunto de setas de navegação (*joystick*). Esta interface permite aos usuários traçar caminhos a serem percorridos pelo robô, mantendo a sua orientação atual, em uma seqüência de passos discretos de 0.20 m a 1.0 m; girar em passos de 15 graus, mudando a sua orientação; ou executar ambos os movimentos (girar e se mover ao mesmo tempo). Na segunda forma de interação, o usuário fornece um alvo a ser atingido em um mapa de posição que mostra a posição inicial do robô. Quando o alvo é especificado, um botão de comando rotulado como "*move to target*" é habilitado. Quando o usuário clica neste botão, o robô se desloca empregando um algoritmo pré-definido baseado em campos potenciais [120] e atinge seu objetivo desviando de obstáculos presentes no ambiente.

Nas duas formas de interação, o usuário pode acompanhar os movimentos do robô por um mapa simplificado de posição (Fig. 5.4). Neste mapa está representada a área de movimentação do robô no ambiente do laboratório. O círculo representa o robô, os retângulos representam os obstáculos restringindo os movimentos desse robô. Quando o usuário clica em um ponto do "mapa" do laboratório especificando um alvo, aparece um "X" que representa a posição final para a qual o robô deverá se locomover. Enquanto este se move, o usuário acompanha a trajetória que é constantemente atualizada neste mapa.

Junto com as formas de interação, a interface do modo básico fornece outros dois painéis no navegador do usuário que complementam a interação entre o usuário e o laboratório REAL. Nestes painéis são exibidas as imagens de vídeo provenientes das câmeras embarcada e panorâmica, instaladas no laboratório. Estes painéis de vídeo, mantidos pela sessão de comunicação, são colocados na parte superior da interface e permitem ao usuário acompanhar as ações realizadas pelo robô.

Fig. 5.4: Interface do modo de interação básico do REAL.

Contêineres e Componentes do Modo de Interação Básico

A Figura 5.5 ilustra uma visão geral dos contêineres e dos componentes nos domínios do usuário e do provedor do serviço que suportam o modo de interação básico.

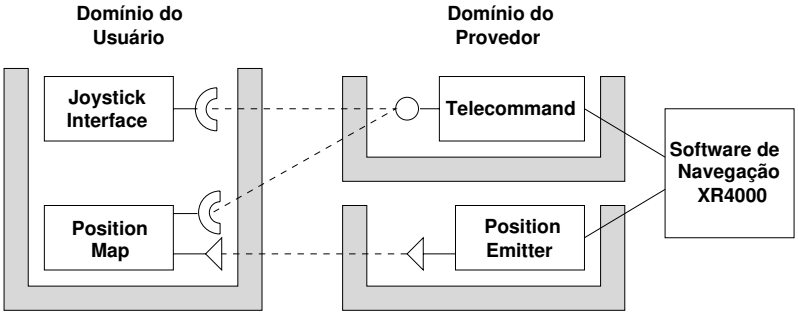

Fig. 5.5: Contêineres e componentes do modo de interação básico do REAL.

No domínio do usuário, o processo começa quando este acessa o serviço básico do REAL. Em seguida, recebe a interface deste modo que é instalada em seu navegador via um *applet* e que consiste de dois tipos de contêineres.

O primeiro tipo de contêiner gerencia dois tipos de componentes de interação: o componente que implementa as funcionalidades da forma de interação proporcionada pelo *joystick* (*JoystickInterface*) e o componente que implementa as funcionalidades da forma de interação proporcionada pelo mapa

de posição (*PositionMap*). Cada um destes componentes exporta uma porta receptáculo que armazena a porta faceta do componente *Telecommand* do domínio do provedor do serviço, que expõe métodos que implementam os comandos de controle do robô. Estes componentes submetem os comandos proporcionados pelo usuário para a faceta que os traduz em movimentos para o robô. Nesta interface são fornecidos três métodos principais: mover, mudar de direção (girar) e mover em direção a um alvo. Estes métodos transformam os seus argumentos para os correspondentes comandos de movimento do robô. O componente *PositionMap* expõe uma porta consumidora de eventos que recebe mensagens de notificação geradas pelo componente *PositionEmitter* do domínio do provedor. Este último é o responsável pela atualização das coordenadas do mapa de posição pelo envio de eventos de posição do robô ao usuário. Este componente calcula as coordenadas da posição do robô a intervalos de 0.5 seg, o que reflete a sua posição corrente (x;y) no momento da leitura no laboratório virtual. Para cada coordenada, o componente gera um evento de posição (*RobotPosition*) e o envia como uma mensagem de notificação através de uma porta de sinal (publicadora de eventos), que este componente expõe, para o componente *PositionMap*. Portanto, a atualização da trajetória do robô no mapa é baseada em eventos trocados entre estes dois componentes. O componente *PositionMap* extrai o conteúdo do evento de posição (as coordenadas do robô) e atualiza a posição no mapa.

Os outros contêineres do domínio do usuário são instâncias do segundo tipo de contêiner e gerenciam os mesmos componentes que suportam a sessão de comunicação, ou seja a realimentação das imagens de vídeo em tempo real através de duas instâncias de componentes apresentadores de vídeo.

O domínio do provedor do serviço, ou seja o laboratório REAL, consiste de três tipos de contêineres. O primeiro tipo de contêiner gerencia o componente *Telecommand*; o segundo tipo de contêiner gerencia o componente *PositionEmitter*; e, o terceiro tipo gerencia os componentes da sessão de comunicação.

Os componentes *Telecommand* e *PositionEmitter* interagem com um servidor responsável pelo controle do robô, que contém o *software* de navegação. Este servidor envia comandos ao robô, fazendo uso da biblioteca disponibilizada pelo seu fabricante. Ele também monitora a sua posição e, caso seja diferente da computada anteriormente, envia-a para o componente produtor de eventos. A comunicação entre os componentes e este servidor é realizada através de *sockets* na rede local do REAL.

De forma semelhante ao domínio do usuário, os outros contêineres do domínio do provedor do serviço são duas instâncias do terceiro tipo de contêiner e gerenciam os mesmos componentes que suportam a sessão de comunicação, conforme descrito na Seção 5.5. Essa sessão utiliza uma interação ponto-ponto com fluxos multimídia entre um capturador e um apresentador de vídeo.

5.4.2 Modo de Interação Avançado

O modo de interação avançado é dirigido a pesquisadores e estudantes com experiência em robótica que queiram desenvolver, planejar, executar e testar os seus próprios algoritmos, métodos de navegação e controle em robôs móveis, bem como experimentos robóticos complexos que exploram as funcionalidades oferecidas pelos robôs. Os algoritmos e experimentos podem explorar muitos campos de pesquisa e desenvolvimento da área de robótica, tais como navegação autônoma, mapeamento do ambiente, planejamento de missão, controle e acompanhamento do robô. Os experimentos consistem da execução do código fornecido pelo usuário e submetido ao processador de controle do robô. Este código pode executar qualquer operação suportada pela interface de programação do robô, tais como realizar um movimento, ler sensores, ou armazenar dados para um processamento posterior. O código fornecido pelo usuário deve ser escrito na linguagem de programação C++, dado que a interface de programação (API) de comunicação com o robô é disponibilizada apenas nesta linguagem.

O modo de interação avançado disponibiliza interfaces com um conjunto de funcionalidades para auxiliar o usuário a realizar os seus experimentos. O usuário pode selecionar localmente os arquivos contendo os seus algoritmos e, utilizando estas funcionalidades, estes códigos serão transferidos, compilados, ligados às bibliotecas de navegação do robô, armazenados e executados no ambiente do laboratório. O resultado da compilação é armazenado no servidor HTTP do REAL e apresentado em seguida no navegador do ambiente do usuário. Caso a compilação tenha êxito, o código executável também é armazenado no servidor de navegação e o usuário pode remotamente e a qualquer momento solicitar a sua execução no processador de controle do robô. Caso contrário, ele pode solicitar um arquivo contendo os erros emitidos durante a compilação.

O servidor de navegação do modo de interação avançado transfere e executa o código fornecido pelo usuário no computador de bordo do robô. Este servidor monitora a execução do experimento fornecido pelo usuário e utiliza o módulo de segurança para proteger o robô de operações que violem os limites de segurança ou que causem choques do robô com obstáculos. Além dos painéis de vídeo no navegador do usuário, este modo proporciona outra valiosa informação, que consiste em fornecer os dados de telemetria, adquiridos pelo código suprido pelo usuário através dos sensores do robô durante a missão. Estes dados ficam armazenados nos servidores do REAL durante um certo período de tempo, para possibilitar uma posterior transferência e avaliação pelo usuário remoto.

De forma semelhante aos demais modos de interação, uma sessão de comunicação é estabelecida tanto no domínio do usuário quanto do provedor do serviço, quando o serviço é acessado e é proporcionada pelos mesmos contêineres e componentes descritos na Seção 5.5. Da mesma forma que o modo de interação básico, utiliza uma interação ponto-ponto com fluxos multimídia entre um produtor e um apresentador. Este modo será descrito em detalhes em uma dissertação de mestrado [41].

5.4.3 Modo de Interação Assistido

O modo de interação assistido é dedicado ao aprendizado remoto pela Internet. Este modo oferece um ambiente colaborativo, que permite a múltiplos usuários o acesso simultâneo ao laboratório virtual REAL. Este modo visa, principalmente, proporcionar um primeiro contato de estudantes com tecnologias antes inacessíveis, por exemplo, na área de robótica.

Existem dois tipos de usuários, o instrutor e os estudantes. Um usuário principal, o instrutor, pode interagir diretamente com o robô, enquanto que os estudantes são atualmente capazes apenas de acompanhar o experimento realizado por este instrutor no robô. Este modo de interação disponibiliza no navegador dos usuários dois tipos de interfaces, cada uma para um tipo de usuário. A interface do instrutor permite que este acesse o robô empregando as interfaces com todas as funcionalidades permitidas dos modos básico e avançado, enquanto que a dos estudantes consiste essencialmente da interface do modo básico, porém com as suas funcionalidades de interação desabilitadas. Assim, é possível que os estudantes acompanhem os experimentos realizados no REAL sem interferir no rumo das atividades. Uma possível extensão deste modo poderia, por exemplo no caso de um curso mais especializado, permitir ao instrutor habilitar o campo de envio de comandos da interface de um dos estudantes. Isto permitiria a cada estudante inscrito praticar os experimentos ministrados.

Como os demais modos de interação apresentados, é necessário no modo assistido que tanto o instrutor quanto os estudantes estabeleçam uma sessão de acesso com este serviço, embora no caso dos estudantes não seja exigida uma reserva prévia de horário.

De forma semelhante ao modo básico (Figura 5.4), uma das interfaces apresenta imagens de vídeo em tempo real e um mapa de posição no navegador dos estudantes. Desta forma, estes estudantes podem acompanhar as aulas e observar a evolução dos movimentos do robô. Eles acompanham, também, os experimentos e as interações conduzidas, por exemplo, por um especialista em robótica. Esta interface não permite nenhuma interação com o robô, porém recebe e apresenta dois painéis contendo os fluxos de vídeo em tempo real obtidos das duas câmeras e os eventos de posição, conforme gerado pelos servidores do REAL.

Adicionalmente ao modo básico, as interfaces apresentadas para os estudantes e para o instrutor incorporam e disponibilizam três características extras: um canal de áudio, uma funcionalidade de interação e uma de apresentação de material didático. A primeira característica consiste em transportar um fluxo de áudio que permite aos estudantes ouvir a voz do instrutor em tempo real. Da mesma forma que os fluxos de vídeo, os fluxos de áudio também são diretamente suportados pelas portas de fluxo contínuo do modelo CM-tel. A segunda característica, ilustrada na Figura 5.6, é uma funcionalidade de *chat* baseada em texto que suporta um mecanismo de comunicação e retorno entre o instrutor e os alunos. Nesta interface encontram-se os campos para a edição e visualização de texto. Dois botões, disponibilizados apenas na interface do instrutor, permitem habilitar e desabilitar

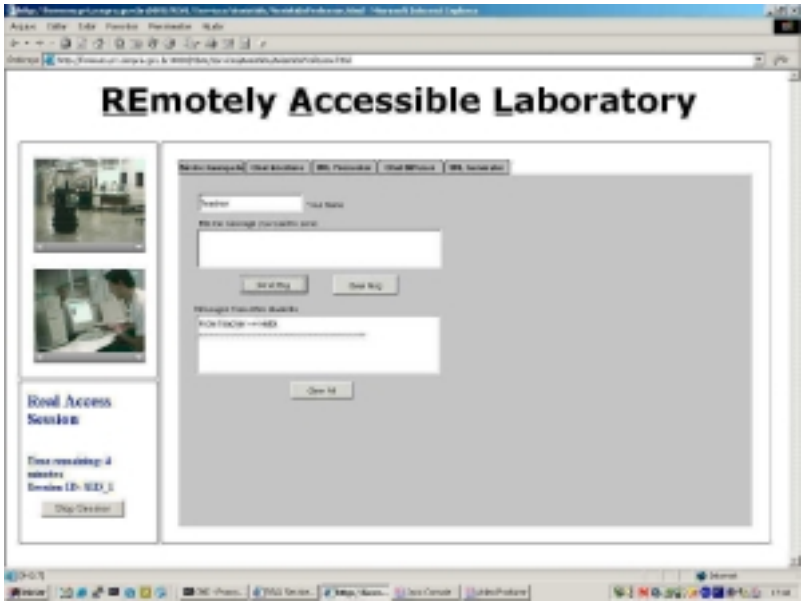

Fig. 5.6: Interface gráfica com campos para chat e botões do modo assistido.

o serviço de *chat*. Finalmente, um serviço de apresentação de material didático é disponibilizado, conforme ilustrado na Figura 5.7. O lado esquerdo desta figura, disponibilizado apenas na interface do instrutor, permite ao mesmo selecionar um endereço (URL) em uma lista de URLs, contendo o conteúdo a ser apresentado (texto, figura, mídia gravada, etc.). Este conteúdo é transferido via HTTP e apresentado no navegador do aluno (lado direito da Figura 5.7).

Contêineres e Componentes do Modo de Interação Assistido

A Figura 5.8 ilustra uma visão geral dos contêineres e dos componentes nos domínios dos usuários (estudantes) e do provedor (instrutor), que suportam o modo de interação assistido.

No domínio dos estudantes, o processo começa quando estes acessam o serviço assistido do REAL. Em seguida, recebem uma página HTML específica para os estudantes com quatro tipos de contêineres instalados na forma de *applets* Java. O primeiro tipo de contêiner gerencia um componente (*AudioConsumer*) que implementa o canal de áudio e expõe uma porta apresentadora de áudio. O segundo tipo gerencia um componente (*SlidePresentation*) que implementa o apresentador de material didático e expõe uma porta consumidora de eventos. O terceiro tipo gerencia um componente (*ChatInterface*) que implementa as mensagens de *chat* (por meio de uma porta faceta) e um conjunto de propriedades (interface do elemento de configuração). Os demais contêineres, não ilustrados na figura, são instâncias do tipo de contêiner e componentes que suportam a sessão de comunicação (descritos na Seção 5.5), ou seja, a realimentação das imagens de vídeo em tempo real através de

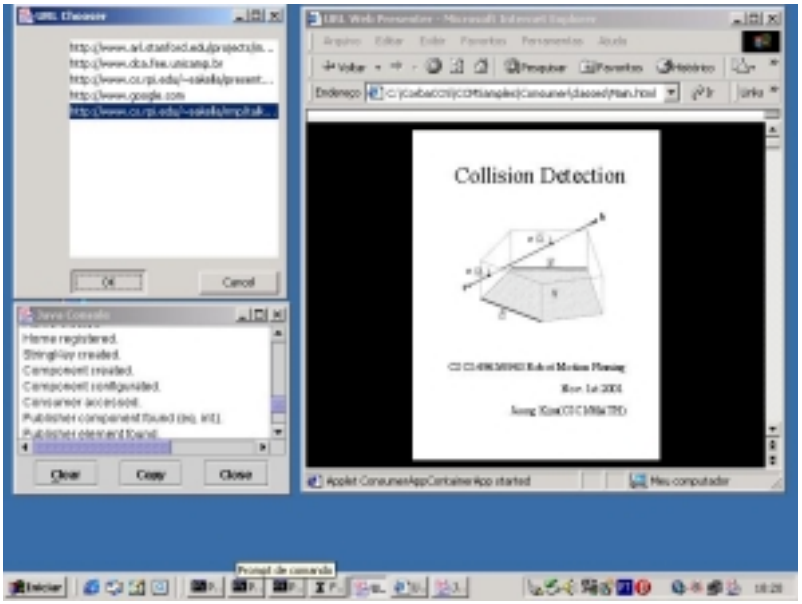

Fig. 5.7: Interface gráfica para apresentação de transparência da URL especificada no modo assistido.

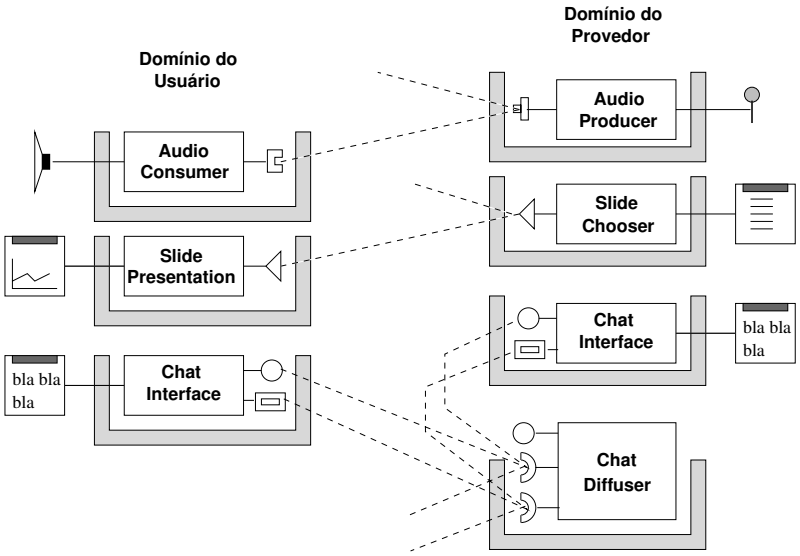

Fig. 5.8: Contêineres e componentes CM-tel do modo assistido do REAL.

duas instâncias de componentes que expõem portas apresentadoras de vídeo.

Quando o componente *SlidePresentation* recebe uma mensagem de notificação, ele extrai o evento contendo a URL, acessa o seu conteúdo via HTTP e, em seguida, apresenta este conteúdo em uma janela do navegador do estudante.

A faceta do componente *ChatInterface* define um método para apresentar as mensagens rece-

bidas no campo apropriado do navegador. O elemento de configuração deste componente armazena informações das propriedades correspondentes à habilitação ou não das mensagens de *chat*. Um componente no domínio do instrutor que interaja com este componente pode inspecionar e modificar estas propriedades.

No domínio do instrutor, o processo começa quando este acessa o serviço assistido do REAL. Em seguida, recebe uma página HTML em seu navegador com cinco tipos de contêineres instalados na forma de *applets* Java. O primeiro tipo de contêiner gerencia um componente (*AudioProducer*) que implementa o canal de áudio e expõe uma porta que captura fluxos de áudio. O segundo tipo gerencia um componente (*SlideChooser*) que expõe uma porta publicadora de eventos. Este evento contém a URL escolhida para enviar aos estudantes. O terceiro tipo gerencia um componente (*ChatDiffuser*) que expõe uma faceta para a recepção das mensagens dos estudantes e dois receptáculos que guardam as referências das facetas e do elemento de configuração de cada estudante. Do primeiro tipo de receptáculo, ele obtém a referência da faceta de cada estudante cadastrado e repassa a mensagem de *chat* recebida dos demais estudantes, para que participem do assunto discutido. Do segundo tipo de receptáculo, ele obtém a referência da propriedade a ser inspecionada e alterada. O quarto tipo gerencia uma instância do mesmo componente (*ChatInterface*) utilizado no lado dos estudantes. Os demais contêineres são instâncias do mesmo tipo de contêiner e gerenciam os componentes que suportam a sessão de comunicação, ou seja, componentes que expõem portas apresentadoras de vídeo.

Semelhantemente ao modo de interação básico e avançado, logo após o serviço ter sido acessado, é estabelecida uma sessão de comunicação tanto no domínio dos usuários quanto no do provedor do serviço e esta é proporcionada pelos mesmos contêineres e componentes descritos na Seção 5.5. Entretanto, este modo demanda uma capacidade *multicast* por parte da sessão de comunicação, que utiliza uma interação ponto-multiponto para propagar fluxos multimídia (vídeo e áudio) e mensagens de notificação de um único produtor para um ou mais apresentadores.

5.4.4 Detalhes da Implementação da Sessão de Serviço

Na implementação da sessão de serviço do REAL, os módulos de *software*, que interagem diretamente com os robôs XR4000 e Nomad 200, foram escritos na linguagem de programação C++, conforme demandado pela interface de programação das bibliotecas dos robôs. Pertencem a esta categoria o *software* fornecido pelo usuário e um conjunto de módulos implementados pelos três modos de interação para o suporte à interação com os robôs descritos nesta seção.

O código de implementação da lógica dos três modos de interação consistem de um conjunto de componentes e contêineres CM-tel, que foram codificados na linguagem de programação Java, onde a maior parte deste código foi gerado diretamente pela plataforma CCM-tel, a partir da sua correspondente especificação em UML.

Os contêineres instalados no domínio do usuário, que contêm a implementação dos três modos de interação, foram implementados como *applets* Java assinados, enquanto que no domínio do provedor de serviço estes foram implementados como processos. A partir da especificação XML do descritor de distribuição, a plataforma CCM-tel gerou arquivos do tipo lote, para a instalação de contêineres baseados em processos e do tipo HTML, para a instalação de contêineres baseados em *applets* Java. A plataforma Java 2, contendo a implementação CORBA (Java IDL), foi empregada em ambos os domínios. Os contêineres instalados no domínio do provedor de serviço executam em processadores com os sistemas operacionais Linux e Windows (2000 e XP). Um terceiro processador, com o sistema operacional Linux, abriga os servidores HTTP (Tomcat - servidor Web com extensões para JSPs e *servlets* e servidor da sessão de acesso,), o servidor de base de dados MySQL e o servidor de nomes CORBA (orbd) disponíveis na plataforma Java 2.

5.5 Sessão de Comunicação

A sessão de comunicação do REAL possibilita a "tele-presença" do usuário no laboratório virtual. Ela é responsável pelo suporte à comunicação multimídia distribuída entre o provedor (laboratório virtual) e os usuários do serviço, possibilitando a visualização e a comunicação entre as partes. A partir da especificação dos componentes, o modelo CCM-tel provê ao usuário remoto toda a infra-estrutura necessária para a utilização da transmissão de fluxos de áudio e vídeo com facilidades de qualidade de serviço (caso a rede permita).

No REAL, uma sessão de comunicação está associada a uma sessão de serviço e é gerenciada no domínio do provedor do serviço. Para os três modos de interação, esta sessão consiste do estabelecimento de dois fluxos de vídeo com conexões ponto-ponto (modos básico e avançado) e ponto-multiponto (modo assistido) para acompanhamento da missão do robô. Um dos canais transporta imagens de vídeo capturadas em tempo real pela câmera embarcada, enquanto o outro transporta imagens de vídeo capturadas pela câmera panorâmica. Em adição, para o modo de interação assistido, este suporte contempla um fluxo de áudio. Os fluxos de vídeo são apresentados em painéis localizados no navegador do usuário. Os fluxos de áudio são recebidos por dispositivos de apresentação de som (alto-falantes). Os detalhes da especifição, projeto e implementação completa desta sessão são apresentados nas referências [50][51].

5.5.1 Contêineres e Componentes da Sessão de Comunicação

A Figura 5.9 ilustra uma visão geral dos contêineres e seus componentes, para representação de fluxos de vídeo com conexão ponto-ponto, nos domínios dos usuários e do laboratório REAL

que suportam a sessão de comunicação. Desde que estes componentes necessitam alocar recursos por muito tempo, tais como o uso intenso de CPU (por exemplo, captura de mídia, codificação, decodificação e apresentação de mídia), eles são geralmente instalados em contêineres separados que desta forma, proporcionam recursos dedicados.

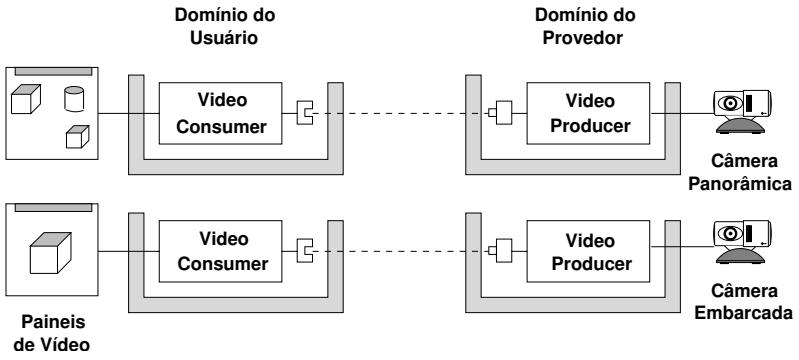

Fig. 5.9: Contêineres e componentes CM-tel da sessão de comunicação do REAL.

A realimentação das imagens de vídeo em tempo real das duas câmeras localizadas no laboratório virtual e apresentadas no navegador dos usuários são proporcionadas por duas instâncias de um mesmo tipo de contêiner e de duas instâncias de um mesmo tipo de componente especificado para transmissão de vídeo.

No domínio do REAL (provedor), esta sessão consiste de um tipo de contêiner que gerencia um componente que expõe uma porta produtora de vídeo (*VideoProducer*). De forma semelhante, um contêiner pode gerenciar um componente que expõe uma porta produtora de áudio (*AudioProducer*). Os fluxos de mídia contínua são estabelecidos ao se interconectar um ou mais componentes que expõem uma porta apresentadora de fluxo a um componente que expõe uma porta produtora de fluxo.

Conexões ponto-multiponto são intermediadas por um refletor que estabelece a comunicação produtor-refletor e refletor-consumidor através de uma interação ponto-ponto, evitando deste modo o bloqueio dos pacotes com endereço *multicast* transportados pela Internet.

No domínio do usuário, esta sessão também consiste de um tipo de contêiner que gerencia um componente que expõe uma porta apresentadora de fluxo (*VideoConsumer* ou *AudioConsumer*).

5.5.2 Detalhes da Implementação da Sessão de Comunicação

Na implementação da sessão de comunicação do REAL, os contêineres instalados no domínio do provedor de serviço foram implementados como processos Java, enquanto que os instalados no domínio do usuário foram implementados na forma de *applets* Java assinados. Todos os componentes

e contêineres CM-tel foram especificados em UML, a partir dos quais o código obtido foi gerado diretamente pela plataforma CCM-tel na linguagem de programação Java.

Um computador portátil, fixo no robô, proporciona as imagens de vídeo da câmera embarcada e executa no sistema operacional Windows 2000. Os demais processadores, incluindo o embarcado do robô e o da câmera panorâmica, executam em microcomputadores sob os sistemas operacionais Linux.

Conforme descrito anteriormente, a sessão de comunicação disponibiliza um refletor de mídia. Refletores de mídia são processos que recebem pacotes em uma porta e os replicam para todos os receptores a ele conectados. No REAL, escolhemos o refletor Darwin [121] da Apple para o suporte ao modo assistido. Darwin é um refletor sofisticado, fornecido como código aberto e disponível para múltiplas plataformas. Darwin suporta os protocolos RTP (*Real Time Protocol*), mensagens de controle RTCP (*Real Time Control Protocol*) e é capaz de interoperar com o JMF (*Java Media Framework*), utilizado pela plataforma CCM-tel para a captura de fluxos de mídia contínua.

5.6 Adaptação Dinâmica, Monitoramento e Controle de QoS no REAL

O objetivo desta seção é ilustrar por meio de uma aplicação o suporte fornecido pelo modelo CM-tel e pela plataforma CCM-tel à capacidade de adaptação dinâmica ao ambiente de execução e ao fornecimento de fluxos de mídia contínua com facilidades de QoS, conforme descrito nas Seções 4.3.4 e 4.3.5. Para isto, apresentamos um cenário onde é avaliada também a integração de componentes CM-tel e agentes móveis no fornecimento de um mecanismo de monitoramento e controle dinâmico de QoS.

O emprego de agentes móveis no monitoramento e controle de QoS em contraste com uma aplicação centralizada se justifica pelos pontos a seguir [122]:

- A plataforma CCM-tel oferece um sistema de agentes de baixíssimo *overhead* no qual o tempo de migração do agente é muito próximo de uma invocação de método remoto[1]. Assim sendo, a migração do agente para um monitoramento e controle *in loco* substitui com vantagens as diversas invocações de métodos necessárias em uma solução centralizada.
- O desenvolvimento do código de monitoramento e controle de QoS é simplificado, dado que todas as interações realizadas pelo agente são locais.
- Características próprias de agentes móveis tais como, autonomia, capacidades de adaptação

[1]Mais precisamente, o *overhead* está na serialização e de-serialização do estado do agente.

e inferência são atrativas para problemas envolvendo gerência de recursos, monitoramento de sistemas e tomada de decisão.

Esta aplicação foi desenvolvida com a participação de alunos da equipe [34][35]. Estes alunos contribuiram com a implementação da parte da reserva de recursos de QoS no nível de rede (redes Diffserv) e na sua integração com o interceptador portável (IQoS) disponibilizado pela plataforma CCM-tel no contêiner.

5.6.1 Cenário para Suporte a Adaptação Dinâmica no REAL

O cenário descrito a seguir, ilustra um mecanismo de monitoramento e controle de QoS para fluxos de mídia contínua, que utiliza as duas formas de adaptação dinâmica descritas (propriedades de QoS mantidas pelo contêiner e agentes móveis). Este cenário foi desenvolvido como parte do protótipo do REAL e apresenta um monitor de QoS implementado por código móvel. Este monitor utiliza um agente móvel que monitora certos parâmetros de QoS *in loco*, definidos na Seção 4.3.4. Com base em um conjunto de regras simples de decisão, este agente atua no sentido de aprimorar a qualidade do vídeo estabelecidos entre um componente produtor e um componente apresentador de vídeo, em função do desempenho da rede.

A aplicação de monitoramento e controle de QoS descrita a seguir consiste de duas entidades de *software*:

1. Monitor de QoS: entidade responsável pelo monitoramento e controle de QoS para determinados fluxos estabelecidos pela aplicação (nesta aplicação, para os fluxos de vídeo). Para cada fluxo, o Monitor de QoS delega o monitoramento e controle de QoS para um agente móvel.
2. Agente de QoS: é um agente móvel instanciado pelo Monitor de QoS com a incumbência de efetuar o monitoramento e controle de QoS para um fluxo específico.

O Monitor de QoS

O Monitor de QoS recebe como entrada um conjunto de fluxos de vídeo para fins de monitoramento e controle de QoS. Este fluxo é determinado pelas referências de suas portas produtora e apresentadora(s). Esta informação é fornecida pelo subsistema que efetua a montagem (ligação de portas) dos componentes da aplicação. O Monitor de QoS obtém ainda dos parâmetros de instalação do contêiner a taxa de bits (banda) ideal para o fluxo. Caso a rede ofereça reserva de recursos por meio do *Bandwidth Broker*, esta taxa foi negociada previamente através do interceptador *IQoS* (Seção 4.3.4). Caso contrário, esta taxa será encaminhada pela rede via política de melhor-esforço. Em ambos os

casos, a taxa é considerada uma taxa alvo para o fluxo, ou seja, a taxa submetida pela porta produtora à rede e consumida pelas portas apresentadoras.

Para cada fluxo, a tarefa de monitoramento e controle de QoS é delegada para uma instância do Agente de QoS. Este agente reportasse periodicamente ao monitor de QoS que, com base nestas informações pode notificar o usuário, renegociar a qualidade de serviço, ou ainda restabelecer o fluxo com novos parâmetros de QoS (por exemplo, tamanho da janela e taxa de quadros).

O Agente de QoS

O Agente de QoS executa um ciclo de cinco fases:

1. monitoramento: o agente obtém os parâmetros de QoS pela leitura das propriedades de monitoramento definidas para as portas produtoras e apresentadoras de fluxo;
2. decisão: o agente decide se há necessidade de uma atuação no sentido de alterar a qualidade do fluxo;
3. atuação: o agente altera a qualidade do fluxo visando ajustar a taxa de bits do fluxo ou aumentar a eficiência da conexão (ou seja, diminuir a taxa de perda de bits);
4. adaptação: o agente modifica sua própria política de atuação visando aprimorar o controle de QoS;
5. comunicação: o agente reporta para o monitor de QoS informações sobre a qualidade da conexão.

A fase de monitoramento é conduzida em todos os locais (contêineres) para os quais o agente migra. As demais fases são conduzidas apenas no lado da porta transmissora.

Ao instanciar o Agente de QoS, o Monitor de QoS informa ao agente a lista de referências das portas que compõem a conexão de fluxo (uma porta produtora e zero ou mais portas apresentadoras) e taxa alvo de bits para o fluxo (T).

Na fase de monitoramento o agente obtém os seguintes parâmetros:

- G - taxa de bits gerada pela porta produtora de fluxo;
- C_i - taxa de bits consumida pela porta apresentadora i, sendo $\bar{C}$ a taxa consumida média;
- J_i - *jitter* referente a porta apresentadora i obtido via relatório RTCP, sendo $\bar{J}$ o *jitter* médio.

Os seguintes parâmetros são computados a partir destas medidas:

- e_i: erro relativo na porta apresentadora i dado por $\frac{T-C_i}{T}$, sendo $\bar{e}$ o erro relativo médio;

- η_i - eficiência da conexão i dada por $\frac{C_i}{G}$, sendo $\bar{\eta}$ a eficiência média.

Na fase de decisão, o agente computa a ação de controle. No nosso caso, esta ação se traduz em uma variação da qualidade do vídeo computada como $\delta = K \times \bar{e}$, onde K é o ganho de controle proporcional.

Para não comprometer a estabilidade do sistema, emprega-se um conjunto de heurísticas que podem alterar o valor computado de δ:

- Se $|\bar{e}| \leq 0.10$ então $\delta = 0$ (nenhuma atuação é conduzida caso o erro seja menor que 10%);
- Se $|\delta| > 0.2$, então $|\delta| = 0.2$ (o valor absoluto da ação de controle deve saturar em 0.2 para evitar mudanças bruscas do ponto de operação do sistema);
- Se $\delta > 0$ e $\bar{\eta_i} < 0.95$, então $\delta = 0$ (não é permitido aumentar a qualidade se a eficiência média da conexão for inferior a 95%);
- Se $\bar{\eta_i} \leq 0.75$, então $\delta = -0.15$ (a qualidade deve ser diminuída se a eficiência média da conexão for inferior a 75%).

Com o valor de δ alterado pelas heurísticas, a fase de atuação tem início. O agente computa a nova qualidade do vídeo pela regra de controle $Q_{N+1} = Q_N + \delta$. A qualidade do vídeo é um valor relativo entre 0 (mínima) e 1 (máxima). O agente altera a propriedade *video_quality* do elemento de configuração do contêiner, pára o fluxo e o reinicia a seguir pelos métodos *stop* e *start* disponibilizados pela porta transmissora de mídia contínua. Efetuada uma ação de controle, a próxima ação irá ocorrer apenas após quatro ciclos do agente ou caso o erro médio exceda 30%.

Terminada a fase de atuação, o agente procede a uma adaptação caso sejam efetuadas quatro atuações em seqüência. As regras de adaptação alteram o valor do ganho proporcional K de acordo com as seguintes heurísticas:

- Se $\bar{\eta_i} \leq 0.75$, então mantenha K inalterado;
- Se sistema oscilante, faça $K = 0.5 \times K$;
- Se sistema sub-amortecido, faça $K = 1.2 \times K$.

As condições oscilante e de sub-amortecimento são detectadas armazenando-se os 4 últimos valores de δ. O sistema é oscilante quando valores positivos e negativos de δ se alteram. O sistema é sub-amortecido quanto todos os valores de δ forem positivos ou negativos.

Finalmente, a cada 5 ciclos, o agente relata as seguintes informações ao monitor de QoS: a eficiência média da conexão, o erro médio, e o *jitter* médio. Com base nestas informações, o monitor de QoS deve decidir alterar outros parâmetros da conexão. Em nossa aplicação, o monitor de QoS diminui o tamanho do quadro caso três relatórios do agente estabeleçam eficiência menor que 80%, erro médio maior que 30% ou *jitter* maior que 500 ms.

5.6.2 Implementação e Resultados

A Figura 5.10 ilustra a arquitetura da implementação do Monitor e Agente de QoS empregada para monitorar e controlar a qualidade de serviço da sessão de comunicação do laboratório virtual REAL [50][51][25].

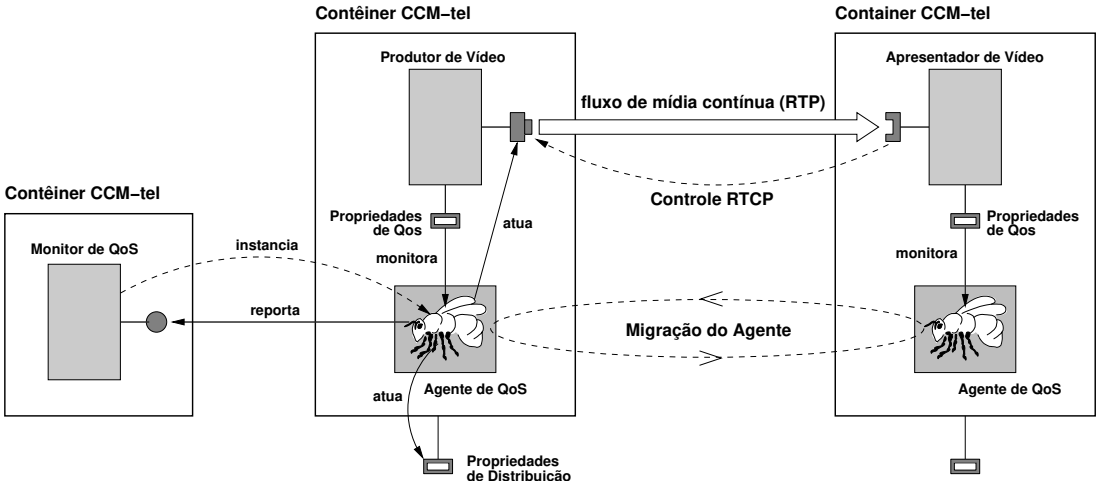

Fig. 5.10: Monitoramento e controle de QoS por agentes móveis.

A sessão de comunicação com controle de QoS emprega dois fluxos de vídeo produzidos pela câmeras de bordo e panorâmica. A Figura 5.11 ilustra para o componente apresentador de vídeo a sua representação UML e correspondente transformação para XML. O componente define propriedades de monitoramento de QoS para a sua porta de fluxo contínuo.

A Figura 5.12 ilustra as especificações em XML do contêiner e do descritor de distribuição para o componente produtor de vídeo.

Os resultados do monitoramento e controle de QoS foram realizados para um fluxo RTP/UDP no formato MJPEG de tamanho 160X120 *pixels*. Os valores iniciais atribuídos ao agente são 0.5 para a qualidade do vídeo (Q, com valores permitidos entre 0 e 1) e 0.5 para o ganho de controle proporcional (K). A taxa alvo no apresentador é 1 Mbis/s. A Figura 5.13 ilustra os valores de Q e K em função dos ciclos de controle. Um ciclo de controle dura 40 segundos e ocorre a cada 4 ciclos completos de migração do agente (o agente migra a cada 5 segundos, ou seja 10 segundos para percorrer os dois contêineres). A figura mostra a adaptação do ganho partindo de 0.5 e estabilizando após 30 ciclos em 0.03125. A qualidade do vídeo, após oscilar significativamente no início estabiliza-se em 0.76 após 15 ciclos.

Quanto ao erro, a Figura 5.14 ilustra a sua variação em função dos ciclos de controle. Após uma oscilação significativa no início, o erro estabiliza-se em torno de 5% após 15 ciclos. Nota-se que uma oscilação do erro em torno do 30º ciclo levou a uma diminuição do ganho proporcional neste ciclo.

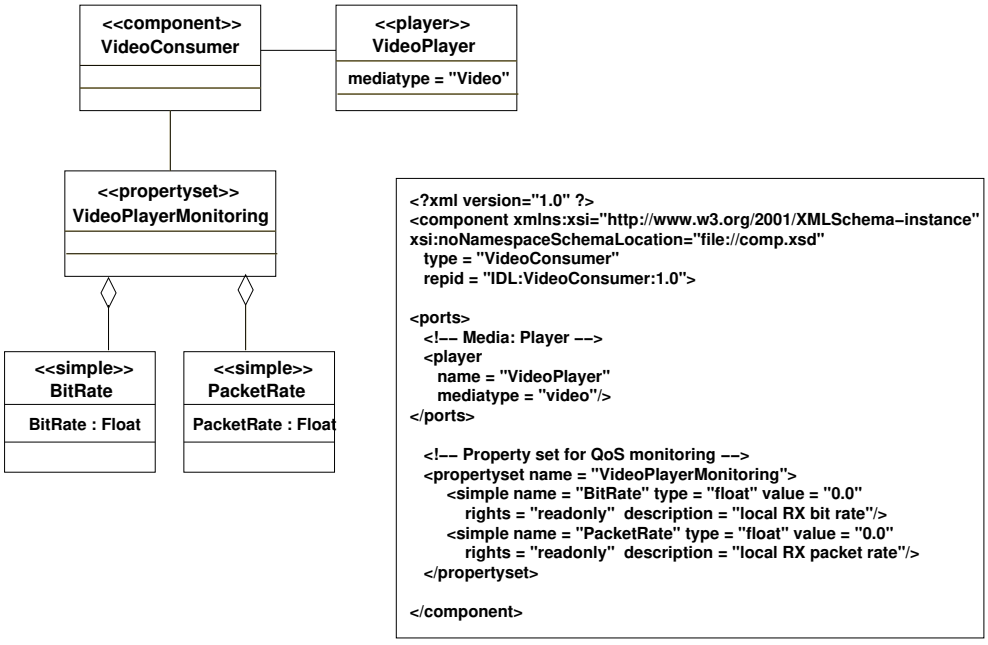

Fig. 5.11: Representações UML e XML do componente apresentador de vídeo.

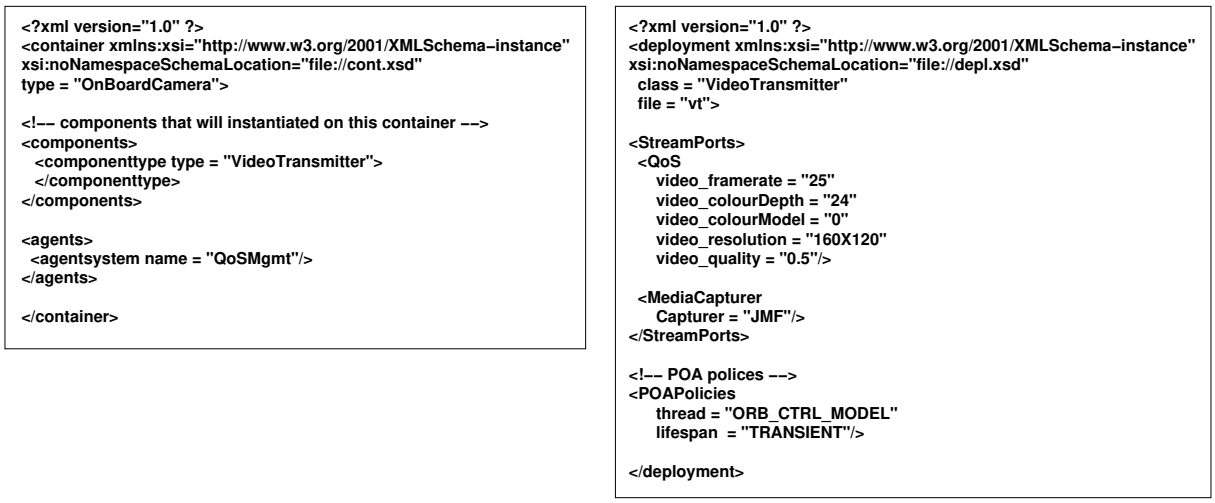

Fig. 5.12: Especificações em XML do contêiner e descritor de distribuição do componente produtor de vídeo.

Apesar dos testes serem conduzidos em uma rede local (mas com grande variação do volume de tráfego), é esperado que as heurísticas de controle sejam igualmente eficazes na Internet pública, talvez com maiores variações no erro. Apesar de estratégias de controle e adaptação mais elaboradas terem sido propostas (por exemplo, baseada em lógica fuzzy [123]) acreditamos que para situações de grandes variações de carga da rede, uma gerência de QoS baseada em heurísticas como as descritas

neste artigo apresentará melhor desempenho e estabilidade.

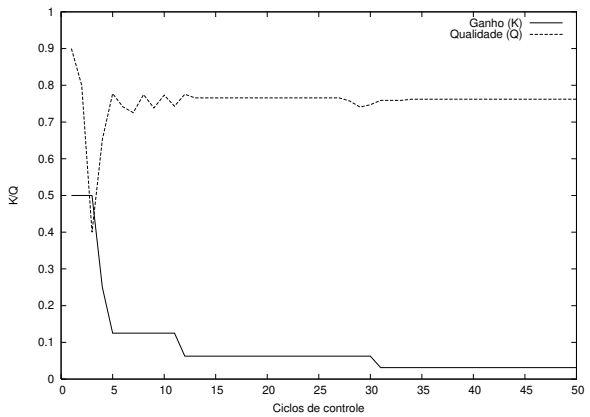

Fig. 5.13: Variação do ganho (K) e qualidade (Q) em função do número de ciclos de controle.

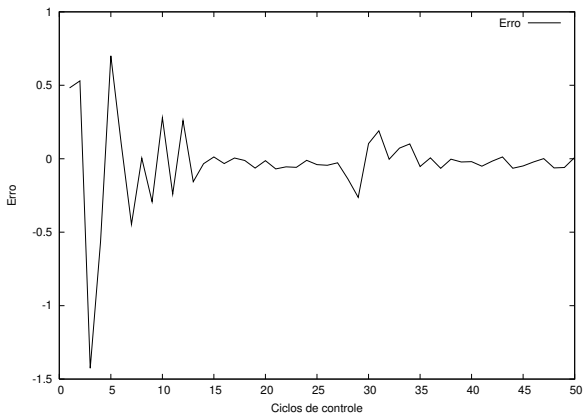

Fig. 5.14: Variação percentual do erro (%) em função do número de ciclos de controle.

5.7 Exemplo de Desenvolvimento Segundo o Modelo CM-tel

Esta sessão apresenta as diretrizes para o desenvolvimento de componentes e contêineres para aplicações telemáticas e ubíquas segundo o modelo CM-tel. Nossa intenção com estas diretrizes é mostrar os passos a serem seguidos pelo desenvolvedor, através dos dois níveis de abstração deste modelo. O primeiro nível, independente de tecnologia, usa os conceitos de modelagem do modelo CM-tel apresentados no Capítulo 3. No segundo nível, a plataforma CCM-tel é utilizada conforme apresentado no Capítulo 4. Ao especificar corretamente em UML os componentes e contêineres de acordo com o perfil UML estabelecido pelo modelo CM-tel é gerado para o especificador ou arquiteto da aplicação um documento XML compatível com o *XML Schema* definido pelo modelo no nível de projeto (Seção 3.3). A partir deste documento XML, os desenvolvedores ou realizadores que implementam os componentes e contêineres da aplicação, obtêm da plataforma o seu código gerado automaticamente. A Figura 5.15 ilustra a relação existente entre a aplicação usando o modelo CM-tel, através da sua plataforma CCM-tel, e um processo de desenvolvimento de *software*, por exemplo, o Modelo Unificado [80].

Um aspecto importante a ser salientado é que as diretrizes mencionadas acima não propõem um novo processo de desenvolvimento de *software*, mas a incorporação aos processos já existentes do modelo e da plataforma propostos neste trabalho. Da mesma forma, não é nosso propósito ilustrar os aspectos internos para o projeto e implementação dos componentes da aplicação, que podem utilizar os novos conceitos prescritos pela área de Engenharia de *Software*, tais como o uso de *frameworks* de classes e padrões de projeto.

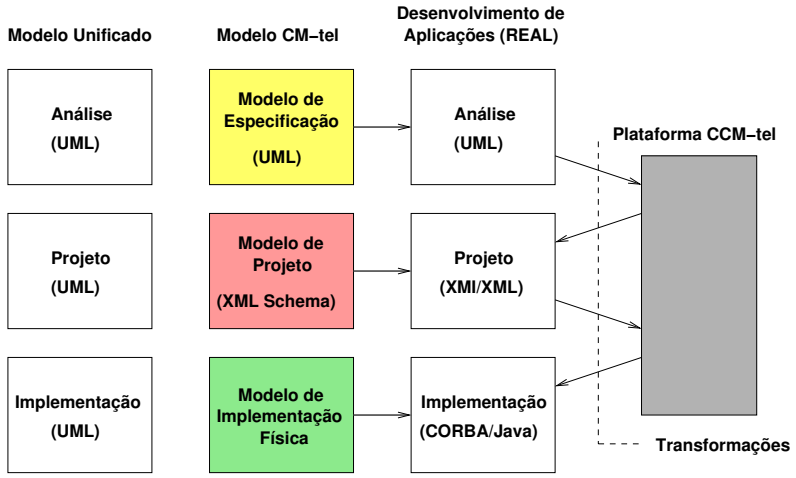

Fig. 5.15: Desenvolvimento de aplicações na plataforma CCM-tel.

De modo a ilustrar as diretrizes de desenvolvimento, escolhemos um subconjunto dos componentes do modo básico de interação, da sessão de comunicação e da sessão de acesso do laboratório virtual REAL. Estes componentes serão especificados nas fases de análise e projeto (independentes de tecnologia), bem como na fase de implementação (dependente de tecnologia).

5.7.1 Modelo de Análise

Em desenvolvimentos orientados a componentes, a fase de análise identifica os componentes e contêineres que irão suprir as funcionalidades da aplicação. Para tal, os componentes e contêineres são modelados por meio do seu modelo conceitual e do seu perfil UML correspondente, bem como de acordo com as restrições CM-tel expressas em OCL. O modelo conceitual define a estrutura de componentes e contêineres, enquanto o perfil UML relaciona as classes da aplicação com esta estrutura. Para o modelo CM-tel, estas classes devem ser modeladas de acordo com a representação apresentada na Seção 3.2, onde no caso de componentes o modelo conceitual é sumarizado pela Figura 3.9 e o perfil UML é apresentado na Tabela 3.1. E no caso de contêineres, o modelo conceitual é sumarizado pela Figura 3.17 e o perfil UML é apresentado na Tabela 3.2. As classes relacionadas com a classe que representa o componente especificam as interfaces dos componentes, ou seja, o contrato de uso entre o componente e seus clientes (componentes ou objetos).

Especificação de Componentes

A Figura 5.16 apresenta a especificação de dois componentes da sessão de comunicação, um componente da sessão de acesso e um componente da sessão de serviço do REAL. O componente

C_videoPlayer com uma porta apresentadora de vídeo (*PL_videoSink*) e o componente *C_videoTransmitter* com uma porta transmissora de vídeo (*TR_videoSource*) pertencem a sessão de comunicação. O componente *C_Assembler* pertence à sessão de serviço e é responsável pela etapa de configuração da aplicação. Este componente possui uma porta consumidora de eventos (*CO_sessionConsumer*). O componente *C_AS* pertence à sessão de acesso e define uma porta faceta (*FA_AS*) para a gerência dos serviços ativos e uma porta publicadora de eventos (*PU_sessionPublisher*) para notificar, através do evento tipo *SessionEvent*, o início e fim da sessão a todos os componentes de serviço conectados (no caso, o componente *C_Assembler*). A Figura 5.16 ilustra também a interconexão das portas destes componentes.

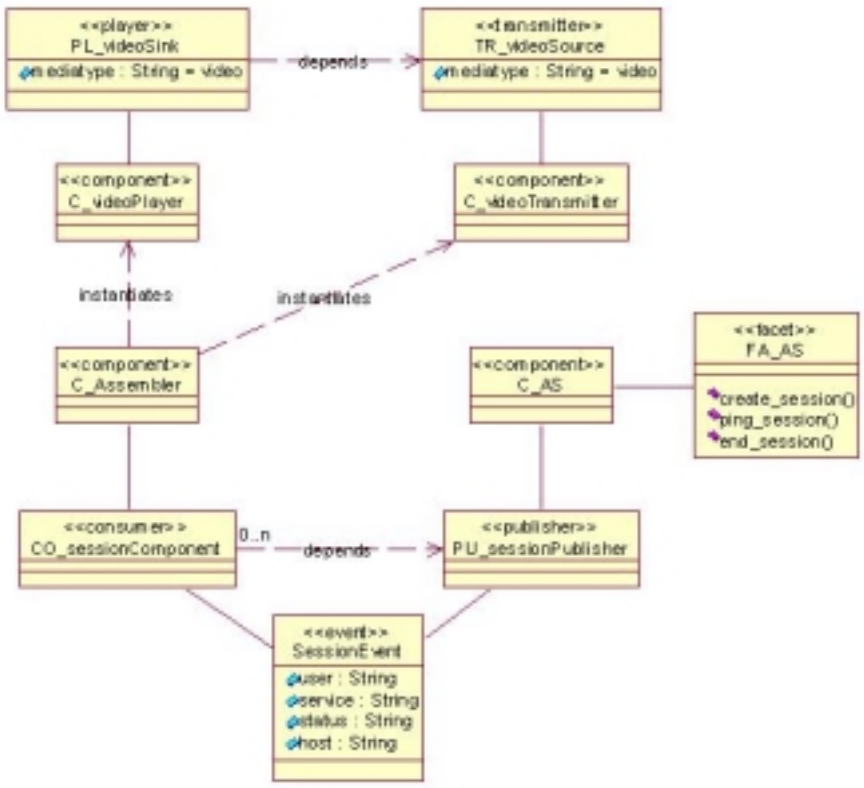

Fig. 5.16: Modelo de análise para componentes.

Especificação de Contêineres

Os contêineres CM-tel também são especificados por classes expressas em UML. Na fase de análise são especificados os componentes hospedados e as suas dependências em relação a outros componentes da aplicação. A Figura 5.17 ilustra dois contêineres abrigando os componentes

C_videoPlayer e *C_videoTransmitter*, onde cada componente possui um contêiner dedicado ao tipo de componente.

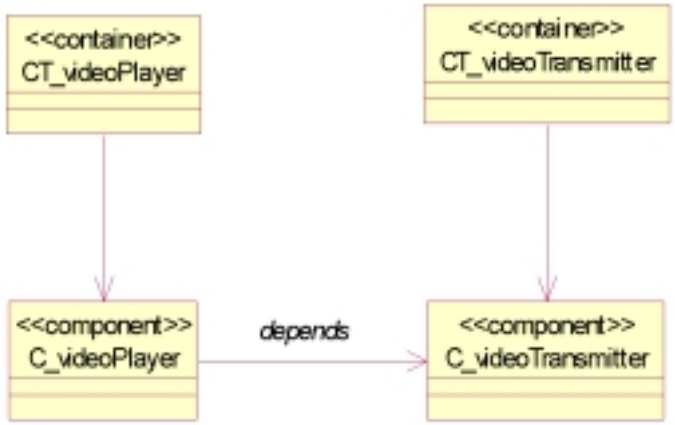

Fig. 5.17: Modelo de análise para contêineres dos componentes C_videoTransmitter e C_videoPlayer.

Dependendo da aplicação, contêineres podem definir agentes móveis e uma porta para acesso às propriedades de instalação (Figura 3.18).

5.7.2 Modelo de Projeto

Na fase de projeto da aplicação, o desenvolvedor adiciona um nível maior de detalhes para as classes especificadas na fase de análise para os componentes, mais precisamente, o detalhamento dos métodos definidos pelas facetas (tipos de retorno e parâmetros de entrada e saída, pré/pós-condições e invariantes). Deve-se notar que os demais métodos não necessitam ser especificados pois os mesmos serão gerados a partir do estereótipo e atributos da porta (por exemplo, métodos do tipo *connect* e *disconnect* apresentados na Figura 3.15). A Figura 5.18 ilustra o modelo de projeto para o componente C_AS.

Adicionalmente, as interações entre os componentes são detalhadas utilizando-se diagramas de interação UML (diagramas de seqüência e de colaboração). Estes diagramas representam a instanciação de componentes e as conexões de suas portas, bem como a invocação dos métodos da aplicação. Por exemplo, a Figura 5.19 ilustra o diagrama de seqüência que mostra a interação do componente responsável pela etapa de configuração da aplicação (*C_Assembler*) com os demais componentes especificados na Figura 5.16.

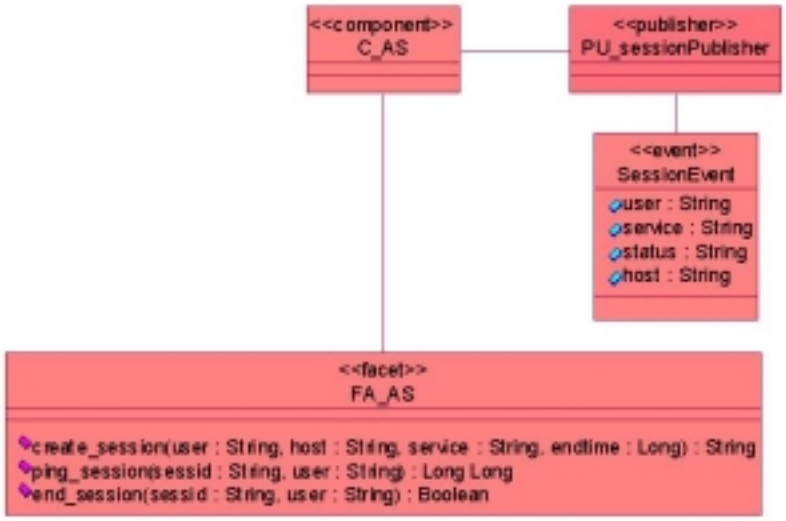

Fig. 5.18: Modelo de projeto para o componente C_AS.

O diagrama de seqüência ilustra as seguintes interações:

- é desencadeado o processo de montagem quando o componente de sessão (*C_AS*), responsável pela gerência do serviço, emite um evento de início de sessão (*start*) ao serviço solicitado, através de sua porta publicadora de eventos, notificando desta forma o componente *C_Assembler*;
- o componente *C_Assembler* cria uma instância do componente produtor de vídeo (*C_videoTransmitter*) e uma instância do componente apresentador de vídeo (*C_videoPlayer*), invocando a operação *create* na fábrica dos respectivos componentes;
- o componente *C_Assembler* invoca a operação *connect* na porta produtora de vídeo do componente *C_videoTransmitter*, conectando a porta apresentadora de vídeo do componente *C_videoPlayer*;
- é encerrada a sessão de serviço, quando o componente de sessão (*C_AS*) emite um evento de término de sessão (*end*) através de sua porta publicadora de eventos, notificando desta forma o componente *C_Assembler*;
- o componente *C_Assembler* invoca a operação *disconnect* na porta produtora de vídeo do componente *C_videoTransmitter* e a operação *remove* na fábrica do componente *C_videoPlayer* (*C_videoPlayerHome*), destruindo-o;

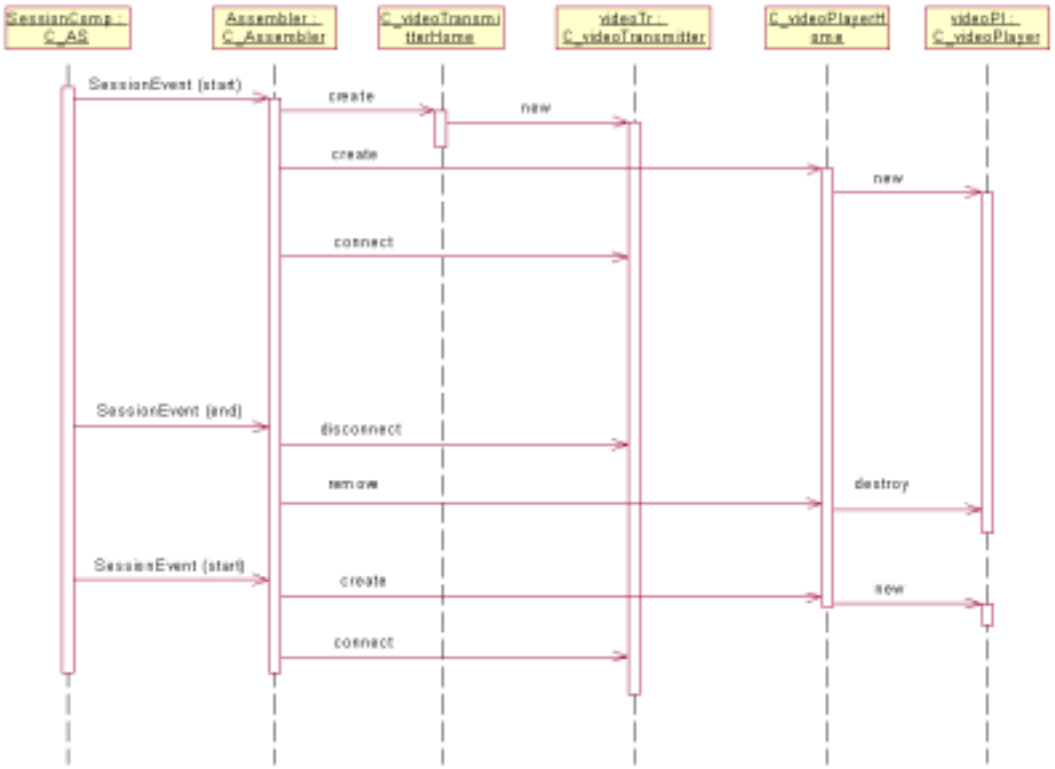

Fig. 5.19: Diagrama de seqüência para o componente C_Assembler

- na seqüência, quando o componente *C_AS* recebe uma nova requisição de acesso ao mesmo serviço, é iniciada uma nova sessão deste serviço com a emissão de um evento *start* para o componente *C_Assembler*;
- o componente *C_Assembler* cria uma nova instância do componente apresentador de vídeo (*C_videoPlayer*), conectando sua porta apresentadora de vídeo como no caso anterior, reiniciando assim a apresentação das imagens de vídeo.

Na fase de projeto da aplicação, não há necessidade de nenhuma adição à especificação de contêineres. Deve ser observado que as pré e pós-condições não aparecem no diagrama de classes acima, porém são editados em interfaces específicas da ferramenta CASE.

Uma vez completados os diagramas UML, os componentes e contêineres são exportados no formato em XMI pela própria ferramenta CASE. Para este exemplo, foi utilizada a ferramenta ROSE [124] da IBM Corporation. A ferramenta CCMtel-Builder gera as especificações XML segundo o modelo de projeto CM-tel a partir das especificações XMI (transformação especificação-projeto). A Figura 5.20 ilustra as especificações XML geradas pela ferramenta para os componentes da Figura 5.16.

Pode-se observar, nesta figura, que as pré e pós-condições definidas anteriormente por meio da ferramenta CASE, são visualizadas na especificação dos componentes em XML. As especificações dos contêineres são apresentadas na Figura 5.21.

```
<?xml version="1.0" encoding="UTF−8"?>
<component xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/comp.xsd"
type="C_videoPlayer" repid="IDL:C_videoPlayer:1.0">
<ports>
<player name="PL_videoSink" mediatype="video"/>
</ports>
</component>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<component xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/comp.xsd"
type="C_videoTransmitter" repid="IDL:C_videoTransmitter:1.0">
<ports>
<transmitter name="TR_videoSource" mediatype="video"/>
</ports>
</component>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<component xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/comp.xsd"
type="C_Assembler" repid="IDL:C_Assembler:1.0">
<ports>
<consumer name="CO_sessionComponent">
<event type="SessionEvent">
<parameter type="string" name="user"/>
<parameter type="string" name="service"/>
<parameter type="string" name="status"/>
<parameter type="string" name="host"/>
</event>
</consumer>
</ports>
</component>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<component xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/comp.xsd"
type="C_AS" repid="IDL:C_AS:1.0">
<ports>
<facet name="FA_AS" repid="FA_AS">
<operation name="create_session">
<syntax>String create_session (string user, string host, string service, long endtime)
</syntax>
<precondition>A session is established and a session ID returned</precondition>
<poscondition/>
</operation>
<operation name="ping_session">
<syntax>Long Long ping_session (string sessid, string user)
</syntax>
<precondition>1. The time remaining for the session is returned
2. The timeout timer  is resetted</precondition>
<poscondition/>
</operation>
<operation name="end_session">
<syntax>Boolean end_session (string sessid, string user)
</syntax>
<precondition>1.The session ends and true is returned</precondition>
<poscondition/>
</operation>
</facet>
<publisher name="PU_sessionPublisher">
<event type="SessionEvent">
<parameter type="string" name="user"/>
<parameter type="string" name="service"/>
<parameter type="string" name="status"/>
<parameter type="string" name="host"/>
</event>
</publisher>
</ports>
</component>
```

Fig. 5.20: Especificação dos componentes em XML.

5.7.3 Modelo de Implementação

Na fase de implementação da aplicação, o desenvolvedor realiza a geração, empacotamento e distribuição dos componentes, agentes móveis e contêineres da sua aplicação.

Geração e Empacotamento de Código

Ainda utilizando a ferramenta CCMtel-Builder, a geração automática de código é realizada. São gerados arquivos contendo código fonte (Java e IDL) para componentes e código Java para contêineres (Fig. 5.22), a partir de suas especificações em XML (transformação projeto-implementação física).

O único código acrescentado pelo desenvolvedor é aquele que implementa a lógica da aplicação. Este deve proporcionar uma implementação das funcionalidades da aplicação, por meio de portas

```
<?xml version="1.0" encoding="UTF−8"?>
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/cont.xsd"
type="CT_videoPlayer">
<components>
<componenttype type="C_videoPlayer">
<dependency type="C_videoTransmitter"/>
<dependency type="C_videoTransmitter"/>
</componenttype>
</components>
<agents/>
</container>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/cont.xsd"
type="CT_videoTransmitter">
<components>
<componenttype type="C_videoTransmitter"/>
</components>
<agents/>
</container>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/cont.xsd"
type="CT_Assembler">
<components>
<componenttype type="C_Assemler">
<dependency type="C_AS"/>
</componenttype>
</components>
<agents/>
</container>
```

```
<?xml version="1.0" encoding="UTF−8"?>
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/ccmBuilder/schema/cont.xsd"
type="CT_AS">
<components>
<componenttype type="C_AS"/>
</components>
<agents/>
</container>
```

Fig. 5.21: Especificação dos contêineres em XML.

facetas e de receptáculos, além das operações de *callback* que são invocadas pelo contêiner durante o ciclo de vida de uma instância, por exemplo, quando o componente é instalado ou removido. No exemplo do componente C_AS, o código acrescentado pelo desenvolvedor corresponde aos métodos *create_session, ping_session* e *end_session* (Fig. 5.18).

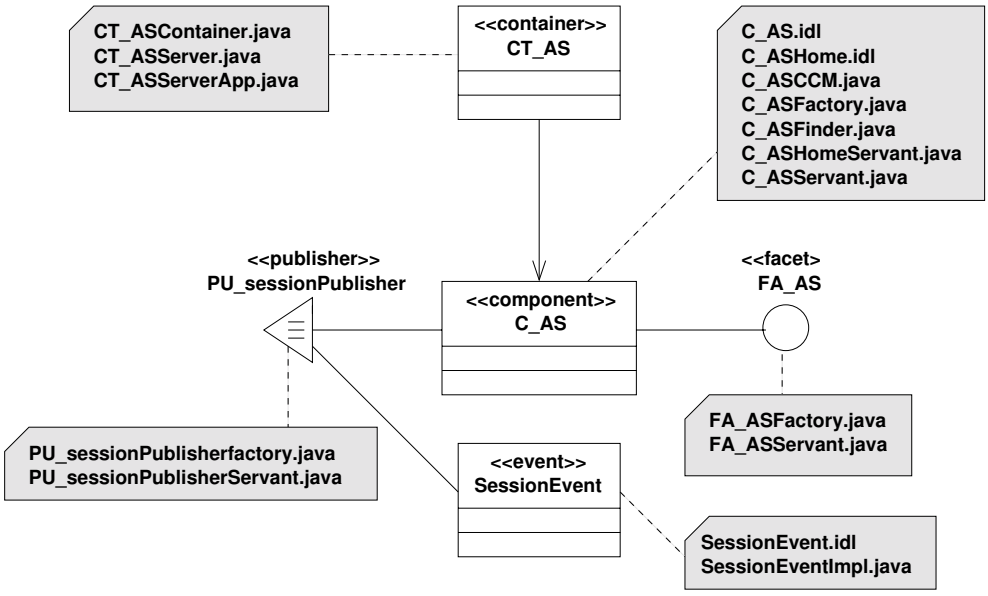

Fig. 5.22: Arquivos gerados para o componente e contêiner CT_AS.

A seguir, todo o processo de compilação dos arquivos fonte em Java e em IDL e de empaco-

tamento destes arquivos para instalação é conduzido por meio da ferramenta Ant. As diretivas de compilação e empacotamento (arquivo build.xml) são geradas pela ferramenta CCMtel-Builder. O resultado do processo é um arquivo no formato JAR contendo todas as partes do componente empacotadas, ou seja, os serventes que implementam a interface equivalente e as portas do componente, os *stubs* e *skeletons* para estes serventes, a fábrica do componente e classes para manipular as portas de configuração e localizar instâncias deste componente em seus contêineres.

Da mesma forma são gerados arquivos no formato JAR, contendo as classes que implementam os contêineres. Estas classes implementam os serventes para o localizador de fábricas (*HomeFinder*) e de fábricas de agentes móveis se especificadas, bem como classes para manipular as propriedades de configuração do contêiner.

Estes arquivos em formato JAR facilitam o reuso de componentes e de contêineres por outras aplicações – o desenvolvedor pode simplesmente reusar cada componente ou contêiner usando o correspondente arquivo JAR. Outra vantagem inerente a componentes é a possibilidade de especializar as instâncias de componentes e contêineres por meio de propriedades, ou mesmo reinstalar contêineres em diferentes nós da rede. No REAL, componentes e contêineres para captura e apresentação de vídeo são reutilizados pelos diferentes modos de interação.

Distribuição do Código

Depois que o desenvolvedor criou ou reutilizou os componentes e contêineres de sua aplicação, inicia-se o processo de distribuição do código. A ferramenta CCMtel-Builder fornece suporte para distribuição dos componentes e contêineres já empacotados. A partir de uma especificação em XML de um descritor de distribuição fornecida pelo desenvolvedor da aplicação, a ferramenta gera arquivos de lote (formatos *bat* para Windows e *sh* para Linux) e HTML para instalação de contêineres como processo e *applets* Java, respectivamente. A Figura 5.23 ilustra o descritor de distribuição para o contêiner de um dos componentes escolhido para exemplo (*C_videoPlayer*). A Figura 5.24 ilustra o correspondente arquivo de instalação no formato *bat*.

Os descritores de distribuição definem os parâmetros de instanciação para os contêineres, por exemplo, localização do servidor de nomes, parâmetros de QoS para portas *stream*, políticas do POA, dentre outros. No caso de instalação de contêineres como processos Java, os parâmetros de instalação são passados para a máquina virtual Java. Neste caso os arquivos JAR dos componentes e contêineres devem estar acessíveis ao *classloader* que carregou o contêiner. No caso de instalação de contêineres como *applets* Java, os parâmetros de instalação são passados como parâmetros do *applet*. Neste caso, os arquivos JAR devem estar disponibilizados em um servidor HTTP, tipicamente o mesmo que abriga o arquivo HTML de instalação. Todos os outros aspectos deste descritor foram apresentados na Seção 4.2.2.

```
<?xml version="1.0" ?>
<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema–instance"
xsi:noNamespaceSchemaLocation="file://C:/CorbaCOS/CCMBuilder/schema/depl.xsd"
  class = "CT_VideoPlayer"
  file = "vp">

<HTML
   codebase = "./jars"
   archive = "comps.jar,propss.jar,avss.jar"
   width = "160"
   height = "160"/>


<ORB
  org.omg.CORBA.ORBClass = "com.sun.corba.se.internal.POA.POAORB"
  org.omg.CORBA.ORBSingletonClass = "com.sun.corba.se.internal.POA.POAORB"/>

<NameService
  NameServiceURL = "http://piranguinha.p4.cenpra.gov.br:8080/Tnameserv"/>

<HomeFinder
  HomeFinderName = "VideoPlayer#1"/>

<StreamPorts>
   <QoS
      video_framerate = "25"
      video_colourDepth = "24"
      video_colourModel = "1"
      video_resolution = "160X120"
       video_quality = "0.5"/>

   <MediaPlayer
      Player = "JMF"/>
</StreamPorts>

<!–– POA polices ––>
<POAPolicies
      thread = "ORB_CTRL_MODEL"
      lifespan  = "TRANSIENT"/>

</deployment>
```

Fig. 5.23: Descritor de distribuição para o contêiner do componente C_videoPlayer.

```
java  –D"org.omg.CORBA.ORBClass"="com.sun.corba.se.internal.POA.POAORB"
     –D"org.omg.CORBA.ORBSingletonClass"="com.sun.corba.se.internal.POA.POAORB"
     –D"NameServiceURL"="http://www.piranguinha.p4.cenpra.gov.br:8080/Tnameserv"
     –D"HomeFinderName"="VideoPlayer#1"
     –D"thread"="ORB_CTRL_MODEL"
     –D"lifespan"="TRANSIENT"
     –D"video_framerate"="25"
     –D"video_colourDepth"="24"
     –D"video_colourModel"="1"
     –D"video_resolution"="160X120"
     –D"video_quality"="0.5"
     –D"Player"="JMF" CT_VideoPlayerServer
```

Fig. 5.24: Arquivo de instalação no formato *bat*.

Após a instalação dos contêineres inicia-se a fase de configuração da aplicação. Nesta fase, os componentes são instanciados (pelos respectivos *homes*) seguindo-se a conexão das portas complementares destes componentes. Um componente especial, o componente de configuração de serviço da aplicação (neste exemplo, o componente *C_Assembler*), executa esta fase inicial da aplicação.

Aplicações complexas podem demandar outros tipos de artefatos de *software*. Por exemplo, a sessão de acesso do REAL utiliza *servlets* e páginas dinâmicas JSP abrigadas em contêineres J2EE. A integração destes diferentes artefatos é propiciada pela própria linguagem Java. Por exemplo, *servlets* da sessão de acesso do REAL invocam operações em facetas do componente de sessão, que por sua vez notifica, por meio de eventos, os componentes da sessão de serviço. Estas notificações decorrem em função de ações executadas pelo usuário (tal como encerramento de sessão) ou eventos internos (tal como tempo de reserva expirado).

5.8 Considerações Finais

Este capítulo contemplou a implementação do protótipo de um laboratório virtual (versão atual do REAL) desenvolvido segundo o modelo CM-tel, no caso mapeado para as tecnologias CORBA e Java. Para esta implementação, os desenvolvedores do REAL utilizaram a plataforma CCM-tel.

Avaliado em relação ao protótipo anterior (versão 2 do REAL), que foi desenvolvido utilizando-se a plataforma de *middleware* de objetos distribuídos CORBA (orientada a objetos distribuídos), constatamos a adequação do modelo CM-tel e da sua plataforma de suporte ao desenvolvimento orientado a componentes de aplicações telemáticas e ubíquas. As principais características do desenvolvimento deste protótipo são apresentadas na seqüência:

- ênfase na especificação da aplicação em UML, deixando a cargo da plataforma CCM-tel a geração do código da aplicação;
- por ser uma aplicação complexa, o laboratório virtual REAL permitiu validar todas as características do modelo CM-tel e da plataforma CCM-tel;
- maior facilidade no desenvolvimento em relação à versão anterior;
- possibilidade de reutilização dos componentes e contêineres desenvolvidos;
- possibilidade de reutilização do projeto de desenvolvimento da aplicação em diferentes plataformas para diversas tecnologias;
- utilização de agentes móveis (para gerência de QoS) sem incorrer em elevados *overheads* como aqueles propiciados pelas plataformas comerciais de agentes tais como *Voyager* e *Grasshoper*;
- comprovação da potencialidade da integração dos paradigmas componente-agente para o desenvolvimento de aplicações distribuídas.

Capítulo 6

Conclusões e Trabalhos Futuros

Este capítulo está estruturado em três seções. Na Seção 6.1 é apresentada uma retrospectiva das contribuições. Na Seção 6.2 são discutidos os desdobramentos do trabalho proposto. Finalmente, na Seção 6.3 são propostas algumas linhas de ação como possíveis extensões e trabalhos futuros.

6.1 Retrospectiva das Contribuições

Em termos gerais, estabelecemos como objetivo para este trabalho de tese contribuir para a concepção e desenvolvimento das aplicações telemáticas e ubíquas. Neste sentido, as principais contribuições realizadas foram:

1. Criação de um modelo de componentes neutro, definido por meio de um perfil UML para o desenvolvimento orientado a componentes de aplicações telemáticas e ubíquas (CM-tel). Este modelo viabiliza inclusive a construção de plataformas leves para o modelo (por exemplo, para execução em dispositivos móveis sem fio).

2. Uma arquitetura neutra de *software* para plataformas de suporte ao modelo CM-tel baseada em transformações XML e sua implementação em uma primeira plataforma mapeada para as tecnologias CORBA e Java (CCM-tel). Uma segunda plataforma mapeada para as tecnologias de serviços Web, Java Micro Edition e computação móvel sem fio (WSCM-tel) foi desenvolvida no âmbito de uma tese de mestrado [32].

3. Validação das plataformas (e, por conseguinte, do modelo) pela implementação do protótipo de uma aplicação telemática e ubíqua de grande porte, o laboratório virtual REAL. Um dos modos de navegação, o modo avançado, foi desenvolvido no âmbito de uma tese de mestrado, enquanto que os outros dois, os modos básico e assistido, foram desenvolvidos no âmbito de sete trabalhos de iniciação científica.

6.1.1 O Modelo de Componentes CM-tel

Inicialmente, procedemos com uma análise dos modelos abertos de componentes, principalmente os modelos CCM e EJB, visando avaliar sua aplicação no desenvolvimento de aplicações telemáticas e ubíquas. Rapidamente, chegamos à conclusão de que tais modelos não suportam três requisitos centrais demandados por estas categorias de aplicações: suporte a comunicação tipo *stream*, suporte a adaptação dinâmica ao ambiente de execução e possibilidades de execução em dispositivos de baixo poder computacional conectados por redes sem fio (*wireless*). Duas possibilidades foram avaliadas: estender um modelo de componentes existente ou desenvolver um novo modelo de componentes. A opção pelo desenvolvimento de um novo modelo foi motivada pela necessidade de desvincular o modelo de uma tecnologia específica, tendência esta confirmada recentemente pelo padrão MDA do OMG.

Para a proposição de tal modelo, apresentado no Capítulo 3, pesquisamos as características essenciais de modelos de componentes, por exemplo, especificação de componentes, geração automática de código e ambientes para execução de componentes. Constatamos que as soluções propostas para o suporte a componentes são fortemente dependentes da tecnologia associada ao modelo de componentes. Assim sendo, propusemos um conjunto de soluções independentes de tecnologia para especificação, projeto, geração de código e suporte a execução de componentes. Estas soluções, centradas em tecnologias neutras como UML e XML, podem servir de base para a criação de modelos de componentes de *software* para outros domínios.

Para o suporte a ubiqüidade avaliamos duas estratégias: reflexão computacional e mobilidade de código. Optamos por mobilidade de código devido a nossa experiência anterior em utilização de agentes móveis e por ser uma estratégia não vinculada diretamente a uma tecnologia. Uma questão importante e atual é a integração componentes-agentes móveis. No modelo proposto, esta integração se dá pela unificação das infra-estruturas de suporte a componentes e agentes em torno de um mesmo contêiner.

6.1.2 Plataforma de Componentes

O objetivo principal para o desenvolvimento das plataformas CCM-tel e WSCM-tel foi avaliar a arquitetura de *software* associada ao modelo CM-tel, notadamente a sua máquina de geração de código baseada em transformações XML. Em adição, procuramos avaliar a viabilidade de instalação e execução de componentes e contêineres CM-tel tanto em computadores tradicionais, como em dispositivos móveis de baixo poder computacional. Finalmente, procuramos avaliar a infra-estrutura de *software* fornecida para o desenvolvimento de aplicações aderentes a este modelo.

A construção da plataforma CCM-tel iniciou-se com a implementação em Java das quatro es-

pecificações de serviços CORBA: propriedades, eventos, gerência de fluxos de áudio/vídeo e agentes móveis. Estas implementações estão descritas em [50][51][106].

Uma versão inicial da plataforma CCM-tel contemplou uma ferramenta para geração de código que foi implementada no âmbito de uma tese de mestrado [2] e apresentada em [26]. Esta versão não utiliza transformações CCM-tel (descritas na Seção 4.2.3) para geração de código, mas um esquema de substituição de marcações em esqueletos de código dirigida por um analisador XML. Esta plataforma também utilizou uma sessão de acesso aderente à arquitetura de serviço TINA apresentada em [33] e desenvolvida no âmbito da tese de mestrado [125].

As principais contribuições relacionadas ao modelo CM-tel e a nova versão da plataforma CCM-tel foram apresentadas nos Capítulo 3 e Capítulo 4. Esta versão da plataforma utiliza geração de código baseada em transformações XML e *templates* codificados como folhas de estilo XSL, bem como possibilita a utilização de agentes móveis nas aplicações. Esta nova versão da plataforma, bem como outros aspectos apresentados no contexto deste trabalho, foram publicados em [30][34][25].

A plataforma WSCM-tel é detalhada na dissertação de mestrado de um dos elementos do nosso grupo [32].

6.1.3 O Laboratório Virtual REAL

Uma primeira versão do laboratório virtual REAL, implementada por meio de objetos distribuídos, foi apresentada na feira CIENTEC [44] promovida pela Unicamp em 2001. A evolução do laboratório virtual REAL, desenvolvido com componentes e contêineres CM-tel e utilizando a versão mais recente da plataforma CCM-tel, está descrita no Capítulo 5 e também pode ser encontrada em outras referências [39][40][34][30].

O processo de desenvolvimento do laboratório virtual REAL utiliza especificações de componentes e contêineres segundo o perfil UML descrito no Capítulo 3, ou seja, é especificado de forma totalmente independente de tecnologia. Esta implementação, que teve a participação de alunos de mestrado [32][41][47] e de iniciação científica [126][127][45][128][46], permitiu avaliar os benefícios e a flexibilidade obtida no desenvolvimento baseado em um modelo de componentes neutro, bem como o mapeamento deste modelo para uma tecnologia escolhida, por exemplo, com o emprego da plataforma de *software* (CCM-tel). Neste desenvolvimento foram reutilizados e personalizados vários componentes e contêineres (reuso de código). Outro benefício importante, o reuso do projeto, está sendo avaliado pela implementação do modo de interação básico do laboratório virtual em dispositivos móveis. Neste caso, os componentes e contêineres especificados em UML são implementados na tecnologia J2ME, pelo emprego da plataforma WSCM-tel.

Alguns experimentos relativos à adaptação ao ambiente de execução por meio de agentes móveis, funcionalidade requerida pelas aplicações ubíquas, foram realizados no âmbito do REAL. Por exem-

plo, no experimento apresentado na Seção 5.6 e publicado em outra referência [25], um agente móvel inspeciona dois parâmetros de qualidade de serviço (taxa de transmissão e taxa de perda de pacotes) disponibilizados pelas portas produtora e consumidora de vídeo e atua no sentido de ajustar estes parâmetros em níveis pré-estabelecidos e, conseqüentemente, aprimorar a qualidade dos fluxos de vídeo transmitidos em função do desempenho da rede utilizada. Um estudo mais aprofundado de adaptação de QoS será conduzido no âmbito do projeto GIGA descrito na Seção 6.2.

6.1.4 Avaliação das Contribuições

O número de trabalhos publicados em revistas (duas revistas do IEEE), congressos nacionais e internacionais contribuiu para a avaliação das contribuições deste trabalho. Recentemente, constatou-se a atualidade dos aspectos abordados nesta tese com o reconhecimento pelo OMG da necessidade de definir padrões para *streams*, qualidade de serviço, perfis UML para domínios específicos e serviços web para colaboração de empresas. Para este fim, OMG publicou uma série de RFPs que estão ainda em andamento [48][49].

Outro aspecto importante foram as três teses de mestrado diretamente vinculadas ao trabalho e os trabalhos concluídos de iniciação científica. Em adição, três alunos de iniciação científica ingressaram no programa de mestrado e continuam trabalhando no nosso grupo.

6.2 Desdobramentos do Trabalho

A experiência adquirida e os resultados proporcionados por este trabalho de tese têm sido aplicados em quatro projetos multi-institucionais submetidos recentemente em editais de chamada e aceitos: QUARESMA/CNPq [129], VIMOS/CNPq [130], GigaBOT/RNP [131] e TIDIA/Fapesp [132].

O projeto QUARESMA (Qualidade de Serviço em Redes - Segurança, Mobilidade e Aplicações) contempla pesquisa e desenvolvimento em diversos temas relacionados à qualidade de serviço em redes avançadas. A nossa participação neste projeto consistiu na pesquisa para o suporte a tecnologias de *middleware* com ênfase no desenvolvimento orientado a componentes, e na implementação de uma aplicação - versão inicial do laboratório virtual REAL.

O projeto VIMOS (Vídeo, Mobilidade e Segurança) tem como objetivo estudar, conceber e avaliar o desempenho de um conjunto de mecanismos de infra-estrutura capazes de lidar com aspectos tais como redes de alta banda, comunicação sem fio e suporte a uma variedade de comunicações multimídia para a transmissão de fluxos multimídia com qualidade e segurança. Por exemplo, a transmissão de áudio e vídeo de tempo real. A nossa participação neste projeto visa prover soluções que atendam às restrições impostas pelos dispositivos sem fio e suporte ao desenvolvimento de aplicações

distribuídas para equipamentos móveis de baixa capacidade computacional, baixa autonomia e de baixa capacidade de comunicação, tais como computadores de mão, telefones celulares e dispositivos embarcados.

O projeto GigaBOT (Laboratórios de Acesso Remoto sobre Redes Avançadas) tem linhas de pesquisa em dois subtemas. O primeiro está inserido no contexto de suporte e desenvolvimento de aplicações multimídia de tempo real em redes avançadas. O segundo consiste no desenvolvimento de sistemas integrados de gerência de redes e aplicações. A nossa participação neste projeto consiste do fornecimento de uma plataforma para o desenvolvimento orientado a componentes de aplicações multimídia tempo real e no suporte à implementação de um laboratório de acesso remoto a equipamentos robóticos.

O laboratório e-Labora do programa de pesquisa e desenvolvimento TIDIA/Fapesp congrega as capacidades requeridas pelo projeto e-Learning com o objetivo de definir e implementar um ambiente padronizado e modular, visando a qualificação e/ou especialização profissional no âmbito do estado de São Paulo. Neste ambiente, componentes de *software* educacionais podem ser facilmente combinados para criar programas de aprendizagem não-presenciais de alta qualidade. A nossa participação neste projeto consiste em disponibilizar laboratórios de acesso remoto integrados ao ambiente de aprendizagem, permitindo a realização remota de experimentos práticos em diferentes equipamentos de aprendizagem.

Durante o desenvolvimento deste trabalho constatamos que tecnologias como CORBA e Java são melhor posicionadas no lado servidor, o que deixa para o lado cliente o uso de facilidades nativas tais como navegadores e capturadores e apresentadores de mídia. Além disso, a comunicação por meio da Internet deve ser restrita a protocolos Internet tais como HTTP e XML. Estas constatações serviram de base para o mapeamento do modelo CM-tel para a tecnologia de serviços Web.

6.3 Trabalhos Futuros

O desenvolvimento desta tese possibilitou vislumbrar algumas linhas de pesquisa que julgamos promissoras, por exemplo:

1. Novas formas de integração componente-agente: neste trabalho, agentes e componentes interagem pelo compartilhamento de um ambiente comum (contêiner). Outra possibilidade de integração é tornar o agente um "componente móvel", onde seria impossível distinguir um componente de um agente. Nesta linha, é também importante o suporte à introspecção de forma independente de tecnologia, onde um conjunto de interfaces de introspecção (ou reflexivas) fornecem informações sobre os componentes, agentes e contêineres de forma a viabilizar a interação com estes elementos em tempo de execução sem a necessidade de conhecimento

prévio (em tempo de compilação) dos elementos. Acreditamos que este tema de pesquisa é importante pois unificaria dois paradigmas de desenvolvimento de *software* até então distintos.

2. Suporte a aplicações sensíveis ao contexto: nesta linha, componentes e agentes teriam a capacidade de se adaptar em função de seu "contexto de execução", determinado em função da localização, ambiente de execução, usuário da aplicação, etc. Esta característica é importante devido a grande profusão de redes sem fio e de dispositivos móveis. Particularmente, na área de robótica estas aplicações são importantes porque permitem o desenvolvimento de sistemas robóticos que se adaptam a um contexto no qual o robô está imerso. Por exemplo: a existência de outros robôs no ambiente, a presença de dispositivos de monitoramento (câmeras e microfones) no ambiente e o perfil do usuário que está utilizando os robôs.

3. Suporte a novos requisitos não-funcionais, por exemplo, requisitos de tempo real. Nesta linha de pesquisa, a interação entre componentes teria que atender requisitos de tempo real tal como *deadlines* para o cumprimento de interações síncronas, garantias de atraso e *jitter* máximos para fluxos de mídia contínua e garantias de periodicidade para determinadas interações. Este suporte traria os benefícios e facilidades proporcionados peloo desenvolvimento orientado a componentes para as aplicações de tempo real.

4. Suporte a componentes para aplicações comerciais. Para trazer componentes para estas aplicações são necessárias metodologias de desenvolvimento de *software* orientadas a componentes que contemplem novas tendências e padrões tais como UML 2.0, MDA e serviços Web. Metodologias de testes, segurança e avaliação de qualidade de *software* são igualmente importantes para o desenvolvimento na indústria.

5. Realização de novos mapeamentos do modelo CM-tel para modelos de componentes comerciais; por exemplo, EJB. Desta forma, utilizando o modelo CM-tel seria possível especificar um projeto independente de tecnologia e, a partir de transformações definidas por este modelo, gerar automaticamente um modelo em uma tecnologia comercial (como o EJB). A seguir são empregadas ferramentas comerciais de geração de código, como por exemplo, Eclipse [58].

6. Fornecimento de bibliotecas de componentes para aplicações telemáticas e ubíquas.

Referências Bibliográficas

[1] J. Daniels. *UML Components*. Syntropy Limited, 2001.

[2] C. A. Miglinski. Uma Ferramenta para o Suporte ao Desenvolvimento de Software Orientado a Componente. Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Outubro 2003.

[3] Object Management Group. Common Object Request Broker Architecture and Specification. Technical Report formal/2002-05-08, OMG, December 2002. `http://www.omg.org/`.

[4] Object Management Group. OMG Home Page, November 2004. `http://www.omg.org/`.

[5] Sun Microsystems. Java Remote Method Invocation. Technical report, November 2004. `http://java.sun.com/rmi`.

[6] M. Kirtland. *Projetando Soluções Baseadas em Componentes*. Editora Campus, 2000.

[7] ISO/IEC. Open Distributed Processing Reference Model - Part 1: Overview. International Standard ISO/IEC 10746-1, International Standard Organization, 1995.

[8] ISO/IEC. Open Distributed Processing Reference Model - Part 3: Architecture. International Standard ISO/IEC 10746-3, International Standard Organization, 1995.

[9] H. Berndt, E. Darmois, F. Duduy, and Y. Inoue. *The TINA Book*. Prentice Hall Europe, 1999.

[10] F. Costa and F. Kon. Novas Tecnologias de Middleware: Rumo à Flexibilização e ao Dinamismo. In *20 Simpósio Brasileiro de Redes de Computadores, SBRC02*, pages 1–61, Buzios, RJ, Maio 2002. SBC. Mini-curso.

[11] G. T. Heineman and W. T. Council. *Component-Based Software Engineering - Putting the Pieces Together*. Addison-Wesley, 2001.

[12] R. Marvie, P. Merle, and J.M.Geib. Towards a Dynamic CORBA Component Platform. In *2nd IEEE International Symposium on Distributed Object Applications, DOA00*, pages 305–314, Antwerpen, Belgium, September 2000.

[13] F. Bachman. Technical Concepts of Component-Based Software Engineering. Technical Report CMU/SEI-2000-TR-008, Software Engineering Institute, May 2000.

[14] T. Meijler and O. Nierstrasz. Beyond Objects: Components. In *Proceedings of the Third International Workshop on Component-Oriented Programming, WCOP98*, 1998.

[15] C. Szyperski, D. Gruntz, and S. Murer. *Component Software - Beyond Object-Oriented Programming*. Addison Wesley Longman Ltd, second edition, 2002.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[17] Object Management Group. CORBA Component Model. Technical Report ptc/2001-11-02, OMG, 2002. `http://www.omg.org/`.

[18] Sun Microsystems. Enterprise JavaBeans Specification, Version 2.1. Technical report, July 2002. `http://java.sun.com/products/ejb`.

[19] R. Marvie and P. Merle. CORBA Componente Model: Discussion and Use with OpenCCM. Technical report, Laboratoire d'Informatique Fondamentale de Lille, 2001.

[20] N. Wang, D. Schmidt, M. Kircher, and K. Parameswaran. Applying Reflective Techniques to Optimize a QoS-enabled CORBA Component Implementation. In *24th Annual International Computer Software and Applications Conference (COMPSAC 2000).*, Tapei, Taiwan, October 2000.

[21] Laboratorie d'Informatique Foundamentalle de Lille (LIFL). OpenCCM website. Technical report, 2003. `http://openccm.objectweb.org/`.

[22] A. Tripathi. Challenges Designing Next-generation Middleware Systems. *Communications of the ACM*, 45(6), June 2002.

[23] Object Management Group. Audio/Video Streams, Version 1.0. Technical Report formal/2000-01-03, OMG, January 2000. `http://www.omg.org/`.

[24] TILAB. Java Agent Development Framework (JADE) v.3.2. Technical report, Telecom Lab, 2004. `http://jade.tilab.com`.

[25] E. G. Guimarães, E. Cardozo, M. F. Magalhães, W. P. Gomes, R. P. Pinto, and L. F. Faina. CCM-tel- uma Plataforma para Aplicações Telemáticas e Ubíquas. In *22 Simpósio Brasileiro de Redes de Computadores (SBRC04)*, Gramado, RS, Maio 2004. SBC.

[26] E. G. Guimarães, E. Cardozo, M. Magalhães, M. Bergerman, A. T. Maffeis, J. L. Pereira, B. G. Russo, C. A. Miglinsk, and R. P. Pinto. Desenvolvimento de Software Orientado a Componentes para Novos Serviços de Telecomunicações. In *19 Simpósio Brasileiro de Redes de Computadores (SBRC01)*, volume I, Florianópolis, SC, Maio 2001. SBC.

[27] D. Carlson. Modeling XML Vocabularies with UML: Parts I, II, III. Technical report, O'Reilly, 2001. `http://www.xml.com/pub/a/2001/08/22/uml.html`.

[28] Object Management Group. Unified Modeling Language (UML) Version 1.5. Technical Report formal/03-03-01, OMG, 2003. `http://www.omg.org/`.

[29] World Wide Web Consortium (W3C). Extensible Markup Language (XML) Specification. Technical report, 2001. `http://www.w3c.org/XML/`.

[30] E. G. Guimarães, A. T. Maffeis, R. P. Pinto, C. A. Miglinski, E. Cardozo, M. Bergerman, and M. F. Magalhães. REAL: A Virtual Laboratory Built from Software Components. *Proceedings of the IEEE*, 91(3):440–448, March 2003.

[31] World Wide Web Consortium (W3C). Extensible Stylesheet Language Transformation (XSLT). Technical report, january 2003. `http://www.w3c.org/xslt/`.

[32] W. P. Gomes. Uma Plataforma de Desenvolvimento de Software Baseado em Componentes para Dispositivos Móveis. Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Campinas, SP, Brasil, 2005.

[33] R. P. Pinto, L. F. Faina, A. T. Maffeis, E. G. Guimarães, C. A. Miglinsk, and E. Cardozo. Uma Arquitetura para Disponibilização e Gerência de Serviços na Internet. In *20 Simpósio Brasileiro de Redes de Computadores (SBRC02)*, volume I, pages 243–258, Buzios, RJ, Maio 2002. SBC.

[34] R. P. Pinto, E. G. Guimarães, E. Cardozo, and M. F. Magalhães. Incorporação de Qualidade de Serviços em Aplicações Telemáticas. In *21 Simpósio Brasileiro de Redes de Computadores (SBRC03)*, volume I, Natal, RN, Maio 2003. SBC.

[35] V. A. M. Souza and E. Cardozo. Estudo e Implementação do Modo Assistido de Interação do Laboratório Virtual REAL. Technical report, UNICAMP/Centro de Pesquisas Renato Archer (CenPRA), janeiro 2004. Proc Fapesp 02/08141-8.

[36] World Wide Web Consortium (W3C). Simple Object Access Protocol (SOAP) Specification. Technical report, 2001. `http://www.w3c.org/TR/SOAP/`.

[37] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Specification. Technical report, 2001. `http://www.w3c.org/TR/WSDL/`.

[38] T.Bellwood, L. Clement, and C. von Riegen (eds.). UDDI Version 3.0.1. Technical report, OASIS Standard Consortium, 2004. `http://www.uddi.org/pubs/`.

[39] E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, M. Bergerman, E. Cardozo, and M. F. Magalhães. REAL: A Virtual Laboratory for Mobile Robot Experiments. In *First IFAC Conference on Telematics Applications in Automation and Robotics*, volume I, pages 209–214, Weingarten, Germany, July 2001.

[40] E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, E. Cardozo, and M. Bergerman M. F. Magalhães. REAL: A Virtual Laboratory for Mobile Robot Experiments. *IEEE Transactions on Education*, 46(1):37–42, February 2003.

[41] A. T. Maffeis. Implementação do Modo de Navegação Avançado do Laboratório Virtual REAL (Título Provável). Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Campinas, SP, Brasil, Em andamento.

[42] J. L. Pereira, E. G. Guimarães, and M. Bergerman. Sistemas Distribuídos Multimídia Aplicados à Robótica. Technical report, Centro de Pesquisas Renato Archer (CenPRA), abril 2001. Proc. Fapesp/IC (99/09922-9): 01-12-1999 a 30-04-2001.

[43] B. G. Russo, E. G. Guimarães, and E. Cardozo. Sistemas Distribuídos Multimídia Aplicados à Robótica. Technical report, UNICAMP/Centro de Pesquisas Renato Archer (CenPRA), julho 2001. Proc. CNPq/Pibic: 01-08-2000 a 30-07-2001.

[44] E. Gomes (editor). Revista da CIENTEC, Agosto 2001.

[45] V. A. S. M. Souza and E. G. Guimarães. Estudo e Implementação de Componentes dos Modos de Navegação Básico e Observador do Laboratório Virtual REAL. In *III Jornada de Iniciação Científica do CenPRA (JICC-2002)*, Campinas, SP, novembro 2002. CenPRA.

[46] R. F. Sassi and E. G. Guimarães. Projeto e Implementação Orientado a Componentes do Modo de Navegação Observador do REAL. In *V Jornada de Iniciação Científica do CenPRA (JICC-2004)*, Campinas, SP, novembro 2004. CenPRA.

[47] J. L. Pereira. Desenvolvimento de Laboratórios de Acesso Remoto Sobre Redes de Alto Desempenho. Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Campinas, SP, Brasil, Em Andamento. Título Provável.

[48] Object Management Group. Streams for CORBA Components, request for proposals. Omg mars/03-01-16, OMG, 2 February 2003. `http://www.omg.org/`.

[49] Object Management Group. Quality of Service for CORBA Components, request for proposal. Omg mars/03-01-18, OMG, 2 February 2003. `http://www.omg.org/`.

[50] E. G. Guimarães, M. Bergerman, E. Cardozo, and M. F. Magalhães. Um Framework de Transmissão de Áudio e Vídeo para Novos Serviços de Telecomunicações. In *18 Simpósio Brasileiro de Redes de Computadores (SBRC00)*, pages 505–518, Belo Horizonte, MG, maio 2000. SBC.

[51] E. G. Guimarães, E. Cardozo, M. Bergerman, and M. F. Magalhães. Um Framework de Transmissão de Áudio e Vídeo para Laboratórios de Acesso Remoto. In *XIII Congresso Brasileiro de Automática (CBA00)*, Florianópolis, SC, Setembro 2000.

[52] OpenEJB. OpenEJB website. Technical report, OpenEJB Group, 2003. `http://www.openejb.org/`.

[53] M. deMiguel. QoS-aware Component Frameworks. In *Proc. of the Tenth International Workshop on Quality of Service (IWQos 2002)*, Miami Beach, USA, May 2002. SBC.

[54] D. C. Schmidt. CIAO - Component Integrated ACE ORB. Technical report, Vanderbilt and Washington Universities, 2003. `http://www.cs.wustl.edu/~schmidt/CIAO.html`.

[55] SantosLab. Cadena website. Technical report, Kansas State University, 2003. `http://cadena.projects.cis.ksu.edu/`.

[56] D. C. Schmidt. TAO, the ACE ORB. Technical report, 2003. `http://www.cs.wustl.edu/~schmidt/TAO.html`.

[57] Object Management Group. XML Metadata Interchange (XMI) Version 1.2. Technical Report formal/2002-01-01, OMG, January, 2002. `http://www.omg.org/`.

[58] IBM. Eclipse website. Technical report, IBM, 2003. `http://www.eclipse.org/`.

[59] Object Management Group. MDA Guide Version 1.0.1. Technical Report omg/03-06-01, OMG, 2003. `http://www.omg.org/`.

[60] T. V. Batista and T. R. Nascimento. A Tool to Convert a PIM into a CORBA IDL Specification. In *MDA Implementers' Workshop*, Orlando, Fl, USA, May 2003.

[61] D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley Publishing, Inc., 2003.

[62] S. Gobel, C. Pohl, S. Rottger, , and S. Zschaler. The COMQUAD Component Model - Enabling Dynamic Selection of Implementations by Weaving Non-functional Aspects. In *International Conference on Aspect-Oriented Software Development (AOSD-04)*, Lancaster, UK, March 2004. ACM.

[63] JBoss Group. JBoss website. Technical report, 2003. `http://www.jboss.org`.

[64] OpenORB. The OpenORB Project. Technical report, Community OpenORB., `http://openorb.souceforge.net/`, 2004.

[65] M. Clarke. An Efficient Component Model for the Construction of Adaptive Middleware. In *ACM Int'l Conf. Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[66] Microsoft Corp. COM Home Page. Technical report, Microsoft Corp., `http://www.microsoft.com/com/`, 2004.

[67] N. Parlavantzas, G. Coulson, and G. Blair. An Extensible Binding Framework for Component-Based Middleware. In *6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003*, Brisbane, Australia, September 2003.

[68] E. Cariou. Communication Components. Technical report, ENST Bretagne and University of Rennes., `http://www-info.enst-bretagne.fr/medium/`, 2003.

[69] Simula. QuA: Quality of Service Aware Component Architecture. Technical report, Simula Research Laboratory., 2004.

[70] T. Ritter. A QoS Metamodel and its Realization in a CORBA Component Infrastructure. In *IEEE Conference on System Sciences*, Big Island, Hawaii, 2003. Proceedings of the Hawaii International.

[71] Fokus Institute. Qedo Open Source Platform website. Technical report, 2003. `http://www.qedo.org/`.

[72] F. Kon, F. Costa, G. Blair, and R. Campbell. The Case for Reflexive Middleware. *Communication of the ACM*, 45(6), 2002.

[73] S. Alda. Support of Collaborative Structural Design Processes through the Integration of Peer-to-Peer and MultiAgent Architectures. Technical report, University of Bonn, 2004. `http://www.informatik.uni-bonn.de/~alda/`.

[74] S. Alda. Strategies for a Component-based Self-Adaptability Model in Peer-to-Peer Architectures. Technical report, University of Bonn, 2004. `http://www.informatik.uni-bonn.de/~alda/`.

[75] M. Amoretti, S. Bottazzi, M. Reggiani, and S. Caselli. Designing Telerobotic Systems as Distributed CORBA-based Applications. In *Distributed Objects Architectures and Applications (DOA2003)*, November 2003.

[76] D. Fabri. Robot Control Designer Education on the Web. In *IEEE International Conference on Robotics and Automation (ICRA2004)*, New Orleans, LA, May 2004.

[77] A. Khamis, D. M. Rivero, F. Rodríguez, and M. Salichs. Pattern-based Architecture for Building Mobile Robotics Remote Laboratories. In *IEEE International Conference on Robotics and Automation (ICRA2003)*, September 2003.

[78] IECAT. Innovative Educational Concepts for Autonomous and Teleoperated Systema (IECAT). Technical report, `http://www.ars.fh-weingarten.de/iecat/`, 2003.

[79] Sun Microsystems. Java 2 Enterprise Edition Specification. Technical report, 2001. `http://java.sun.com/j2ee`.

[80] I. Jacobson, G.Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley Longman Inc, 1999.

[81] E. G. Guimarães, A. Lopes, J. A. Coello, and M. F. Magalhães. A Distribuição do Ambiente para Desenvolvimento de Software de Tempo Real STER. In *Seminário Franco Brasileiro em Sistemas Informáticos Distribuídos*, Florianópolis, SC, Setembro 1989.

[82] M. Volter, A. Schmid, and E. Wolff. *Server Component Patterns*. John Wiley & Sons,Inc., 2002.

[83] J. Cheesman and J. Daniels. *UML Components - A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.

[84] I. Jacobson, G.Booch, and J. Rumbaugh. *The Unified Modelling Language User Guide*. Addison Wesley, 1998.

[85] B. Meyer. The Significance of Components. *Software Development Online*, November 1999. `http://www.sdmagazine.com/`.

[86] C. Farias. *Architectural Design of Groupware Systems: a Component-Based Approach*. Tese de doutorado, University of Twente, 2002.

[87] L. Bass, P. Clemens, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[88] B. Meyer. Contracts for Components. *Software Development Online*, July 2000. `http://www.sdmagazine.com/`.

[89] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2nd edition, 1997.

[90] A. Beugnard, J. Jézéquel, N. Plouseau, and D. Watkins. Making Components Contract Aware. *IEEE Computer*, 32(7):38–45, July 1999.

[91] C. Szyperski. Components and Contracts. *Software Development Online*, May 2000. `http://www.sdmagazine.com/`.

[92] R. Kramer. iContract - The Java Design by Contract Tool. In *TOOLS 26: Technology of Object-Oriented Languages and Systems, IEEE Computer Society Press.*, pages 295–307, Los Alamitos, Ca., 1998.

[93] S. Omohundro and D. Stoutamire. Sather 1.1 Language Specification. Technical report, International Computer Science Institute, 1996. `http://www.icsi.berkeley.edu/Sather/`.

[94] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley, 1999.

[95] J. Hopkins. Component Primer. *Communications of the ACM*, 43(10):27–30, October 2000.

[96] J. Siegel. *CORBA 3: Fundamentals and Programming*. John Wiley & Sons,Inc., 2nd edition, 2000.

[97] Object Management Group. Object Constraint Language Specification. Technical Report formal/03-03-13, OMG, 2003. `http://www.omg.org/`.

[98] I. Pyarali, C. O'Ryan, and D. Schmidt. A Pattern Language for Efficiente, Predictable, Scalable, and Flexible Dispatching Mechanisms for Distributed Object Computing Middleware. In *IEEE/IFIP International Symposium on Object-Oriented Real Time Distributed Computing.*, Newport Beach, CA, 2000.

[99] C. O'Ryan, D. Schmidt, and D. Levine. Applying a Scalable CORBA Event Service to Large-scale Distributed Interactive Simulations. In IEEE, editor, *5th Workshop on Object-oriented Real-time Dependable Systems*, Monterey, CA, November 1999.

[100] T. Harrison, D. Levine, and D. Schmidt. The Design and Performance of a Real-Time CORBA Event Service. In ACM, editor, *OOPSLA-97*, Atlanta, GA, October 1997.

[101] World Wide Web Consortium (W3C). XML Schema Part1: Structures. Technical report, january 2003. `http://www.w3c.org/TR/`.

[102] World Wide Web Consortium (W3C). XML Schema Part2: Data Types. Technical report, january 2003. `http://www.w3c.org/TR/`.

[103] D. Carlson. *Modeling XML Applications with UML.* Addison-Wesley, 2001.

[104] Sun Microsystems. Java Media Framework API Guide. Technical report, November 1999. `http://java.sun.com/jmf`.

[105] R. C. Bosnardo. Um Sistema de Vídeo-conferência para Educação a Distância Baseado em Padrões Abertos. Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Julho 2001.

[106] L. F. Faina, R. P. Pinto, E. G. Guimarães, and E. Cardozo. Mobile Agents for Supporting Ubiquity in Telecommunication Services. In J. Nogueira A. Loureiro, editor, *Second Latin American Network Operation and Management Symposium (LNNOMS)*, pages 50–61, Belo Horizonte, MG, August 2001.

[107] Object Management Group. Property Service, Version 1.0. Technical Report formal/2000-06-22, OMG, April 2000. `http://www.omg.org/`.

[108] Object Management Group. Event Service, Version 1.1. Technical Report formal/2001-03-01, OMG, March 2001. `http://www.omg.org/`.

[109] L. R. Queiroz. Um Laboratório Virtual de Robótica e Visão Computacional. Tese de mestrado, Instituto de Computação, UNICAMP, Novembro 1998.

[110] L. R. Queiroz, S. S. Bueno, and A. Elfes. Educação à Distância em Robótica e Visão Computacional. *Revista Brasileira de Informática na Educação*, (3):17–26, Setembro 1998.

[111] A. Elfes L. R. Queiroz, S. S. Bueno. Design and Operation of the Brazilian Robotics and Computer Vision Virtual Laboratory. In *6th International Congress on Informatics in Education*, Havana, Cuba, 1998.

[112] E. G. Guimarães and M. F. Magalhães. Capacitação Tecnológica em Sistemas Distribuídos Abertos (relatório técnico final). Technical report, Fundação Centro Tecnológico para Informática (FCTI), Junho 1999. Projeto de Cooperação FCTI-UNICAMP.

[113] E. G. Guimarães and M. F. Magalhães. Sistemas Distribuídos Abertos para Novos Serviços Telemídia em Redes de Alta Velocidade (relatório técnico final). Technical report, Fundação Centro Tecnológico para Informática (FCTI), Setembro 2001. Projeto de Cooperação FCTI-UNICAMP.

[114] E. G. Guimarães and M. F. Magalhães. Tecnologias Orientadas a Componentes para o Desenvolvimento de Aplicações Telemáticas. Technical report, Centro de Pesquisas Renato Archer (CenPRA), Outubro 2003-2005. Plano Trabalho do Projeto de Cooperação CenPRA-UNICAMP.

[115] G. McKee. The Development of Internet-based Laboratory Environments for Teaching Robotics and Artificial Intelligence. *Proc. IEEE Int. Conf. Robotics and Automation*, 3:2695–2700, 2002.

[116] S. Jia, Y. Hada, G. Ye, and K. Takase. Distributed Telecare Robotic Systems Using CORBA as a Communication Architecture. *Proc. IEEE Int. Conf. Robotics and Automation*, 2:2202–2207, 2002.

[117] K. Han, Y. Kim, and S. Hsia. Internet Control of Personal Robot between KAIST and UC Davis. *Proc. IEEE Int. Conf. Robotics and Automation*, 2:2184–2189, 2002.

[118] Apache. Apache Tomcat. Technical report, The Apache Software Foundation, january 2004. `http://www.apache.org/jakarta/`.

[119] MySQL. MySQL Programmer´s Guide. Technical report, 2004. `http://www.mysql.org/`.

[120] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[121] Apple. Darwin Relfector Programmer´s Guide. Technical report, Apple Corp., 2004. `http://developer.apple.com/darwin/`.

[122] F. Baschieri andP. Bellavista and A. Corradi. Mobile Agents for QoS Tailoring, Control and Adaptation over the Internet: the ubiQoS Video on Demand Service. In *Proc. of the 2002 Symposium on Applications and the Internet (SAINT)*, Nara City, Nara, Japan, Jan 2002.

[123] C. Koliver and J.M. Farines. Um Controlador Nebuloso para Adaptação de QoS. In *19 Simpósio Brasileiro de Redes de Computadores*, Florianópolis, SC, Maio 2001. SBC.

[124] IBM. Rational Software. Technical report, IBM Corporation, january 2004. `http://www.rational.com/`.

[125] R. P. Pinto. Sessão de Acesso TINA com Suporte à Adaptação de Serviços Através de Agentes Móveis. Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Julho 2001.

[126] J. L. Pereira, M. Moço, B. G. Russo, E. G. Guimarães, M. Bergerman, and E. Cardozo. Sistemas Distribuídos Multimídia Aplicados à Robótica. In *II Jornada de Iniciação Científica do CenPRA (JICC-2000)*, Campinas, SP, setembro 2000. CenPRA.

[127] B. G. Russo, E. G. Guimarães, and E. Cardozo. Sistemas Distribuídos Multimídia Aplicados à Robótica. In *III Jornada de Iniciação Científica do CenPRA (JICC-2001)*, Campinas, SP, setembro 2001. CenPRA.

[128] V. A. S. M. Souza, R. F. Sassi, and E. G. Guimarães. Desenvolvimento Orientado a Componentes da Sessão de Comunicação do REAL. In *IV Jornada de Iniciação Científica do CenPRA (JICC-2003)*, Campinas, SP, novembro 2003. CenPRA.

[129] O. Duarte. Projeto QUARESMA - CNPq - Edital Redes Avançadas - Chamada CNPq 10/2001. Technical report, CNPq, 2001. `http://www.gta.ufrj.br/quaresma/`.

[130] N. Fonseca. Projeto VIMOS - Chamada Conjunta MCT/SEPIN-CNPq-FINEP 01/02 - Programa de Apoio á Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação. Technical report, CNPq, 2002. `http://www.dcc.unicamp.br/~nfonseca/vimos/`.

[131] RNP. Projeto GIGA, 2004. `http://www.rnp.br/pd/giga`.

[132] Fapesp. Tecnologia da Informação no Desenvolvimento da Internet Avançada (TIDIA), 2004. `http://www.tidia.fapesp.br/`.