Desenvolvimento de Software Orientado a Componentes Para Novos Serviços de Telecomunicações

Eliane G. Guimarães*, Antonio T. Maffeis
James L. Pereira, Bruno G. Russo, Marcel Bergerman
Instituto Nacional de Tecnologia da Informação - ITI
CP 6162 - Campinas-SP 13089-970
{eliane,maffeis,jameslaw,bgrusso,marcel}@ia.cti.br

Eleri Cardozo, Mauricio F. Magalhães
Carlos A. Miglinski, Rossano P. Pinto
DCA-FEEC - UNICAMP
CP 6101 - Campinas-SP 13083-970
{eleri,mauricio,mig,rossano}@dca.fee.unicamp.br

Sumário

Este artigo apresenta uma estratégia de desenvolvimento de software para novos serviços de telecomunicações. Tais serviços são altamente complexos e requerem um processo de desenvolvimento rápido, barato e confiável. Devido a estes fatores, o desenvolvimento de software para serviços de telecomunicações emprega cada vez mais sistemas de hardware e software "de prateleira". As soluções especializadas tenderão a desaparecer devido ao seu alto custo e tempo de desenvolvimento. A estratégia de desenvolvimento proposta neste artigo é orientada a componentes de software e utiliza modernas tecnologias tais como CORBA, XML e UML. Resumidamente, componentes consistem de software desenvolvido segundo um modelo bem definido e com alto grau de abstração, reuso e configuração. Após a descrição da estratégia de desenvolvimento, o artigo apresenta um serviço de laboratório virtual construído segundo a estratégia proposta.

Palavras-chave: Internet: protocolos, serviços e aplicações; Comunicação Multimídia; XML; CORBA; Telerobótica.

*Aluna de Doutorado da Faculdade de Engenharia Elétrica e de Computação, UNICAMP.

Abstract

This paper presents a software development strategy for new telecommunication services. Such services are highly complex and demand a fast, cheap and reliable development process. Due to these factors, the development of software for new telecommunication services employs more and more off-the-shelf software and hardware systems. Customized solutions tend to disappear due to their high costs and development time. The strategy proposed in this article is component-oriented and employs modern technologies such as CORBA, XML and UML. Briefly, components consist of software built according to a well defined model and with high degree of abstraction, reuse and configuration. After the description of the development strategy, the article presents a virtual laboratory service built according to the proposed strategy.

Keywords: Internet: protocols, services and applications; Multimedia Communication; XML; CORBA; Telerobotics.

1 Introdução

O desenvolvimento de modernos serviços de telecomunicações teve um grande avanço na década de oitenta com a padronização das redes inteligentes de telefonia (IN: *Intelligent Network*). A tecnologia concebida para IN foi o primeiro passo no sentido de desvincular o serviço do hardware de comutação que o suporta. Em outras palavras, o serviço passou a ser cada vez mais baseado em software e, idealmente, independente do hardware de comutação. Serviços concebidos para redes IN residem integralmente na rede do provedor, ou seja, o terminal possui capacidade apenas de sinalização, não de processamento de parte da lógica do serviço.

Na década de noventa, com a explosão da Internet movida pela *World Wide Web* (WWW), surge uma nova classe de serviços. Estes serviços, outrora centrados exclusivamente na difusão de informação, estão evoluindo para serviços de telecomunicações. Esta nova classe de serviço é viável na atualidade pelo constante aumento das velocidades de comutação, tanto no acesso quanto no núcleo da rede. Serviços tradicionais da Internet se contrapõem aos serviços IN no sentido de estarem totalmente concentrados na periferia da rede (nos *hosts*), ou seja, a rede não participa do processamento do serviço. Atualmente, um meio-termo entre estes dois extremos se faz necessário [1]. As razões para isto incluem:

1. novos serviços de telecomunicações terão na Internet uma importante rede de acesso dada a sua ubiqüidade;
2. *backbones* possuem taxas suficientes para comportar serviços que manipulam mídia contínua em tempo real;
3. novos terminais conectados à Internet possuem capacidade limitada de processamento, como é o caso de telefones celulares e *notepads*.

Face a este cenário, os novos serviços de telecomunicações deverão ter seu processamento distribuído entre o terminal e o provedor (este último fornecendo a capacidade de processamento adicional que o serviço necessita). Exemplo típico desta categoria de serviços inclui acesso à Internet através de telefones celulares, onde, apesar de sua baixa capacidade computacional, o terminal dispõe de um navegador capaz de acessar a Internet com auxílio da infra-estrutura existente no provedor do serviço. Outro fator importante para os novos serviços é o seu *time to market* cada vez mais reduzido. Em um ambiente competitivo, os serviços devem ser disponibilizados assim que as necessidades de mercado sejam identificadas. Finalmente, os novos serviços devem conceder ao usuário capacidade de configuração e gerência. Por exemplo, um serviço pode ser configurado para operar a partir de vários tipos de terminais, apresentando diferentes comportamentos para diferentes terminais.

Este artigo apresenta uma estratégia de desenvolvimento de software tendo os novos serviços de telecomunicações como aplicações potenciais. A estratégia consiste na definição de um modelo de componentes de software; uma arquitetura de suporte para este modelo; e um conjunto de diretrizes para a incorporação do modelo e arquitetura nos processos já estabelecidos de desenvolvimento de software.

O artigo está organizado na seguinte forma. A seção 2 apresenta os conceitos relacionados ao desenvolvimento de software orientado a componentes. A seção 3 apresenta a estratégia de desenvolvimento proposta. A seção 4 apresenta uma arquitetura de suporte a sistemas baseados em componentes aderente à estratégia de desenvolvimento proposta. A seção 5 ilustra a aplicação do modelo no projeto REAL (*REmotely Accessible Laboratory*), um laboratório virtual de robótica implementado como um serviço telemático. Finalmente, a seção 6 apresenta as conclusões e novas pesquisas em curso.

2 Frameworks e 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 [2]. Um ponto crucial para o caso de software de natureza distribuída é a incorporação ao modelo do sistema de fatores pertinentes à distribuição. No ciclo clássico, 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, sendo posteriormente sintetizada na fase de implementação com o auxílio de um *middleware* tipo CORBA[1], DCOM[2] ou Java RMI[3].

É importante observar que mesmo os modernos processos de desenvolvimento de software [2] são deficientes para o desenvolvimento de sistemas distribuídos. Esta deficiência se deve à ausência de estratégias que permitam construir sistemas com alto índice de reuso e geração automática de código (distribuído). Nestas estratégias, complexas funções de

[1]Common Object Request Broker Architecture.
[2]Distributed Component Object Model.
[3]Remote Method Invocation.

distribuição tais como comunicação, segurança, configuração, notificação e transação são incorporadas ao sistema a partir de modelos e especificações, e não através de codificação.

Recentemente, novos conceitos visando aprimorar os processos de desenvolvimento de software têm surgido [3]. Dentre estes, os mais importantes são os conceitos de *frameworks* de classes, padrões de projeto (*design patterns*), componentes e *frameworks* de componentes. Estes conceitos são descritos de forma resumida a seguir. As referências [4] e [5] tratam extensivamente destes temas.

Frameworks de Classes e Padrões de Projeto

Em desenvolvimento de software, *framework* é um conceito bastante genérico e empregado em várias situações. Adotamos aqui a definição de Szyperski [4]: *um framework é um conjunto de classes, algumas das quais abstratas, que permite a reutilização de projeto para um determinado domínio*. Portanto, *frameworks* permitem o reuso de projetos, não de objetos individuais como em técnicas de desenvolvimento orientado a objeto.

Embora implícito no conceito fundamental de *framework*, características como extensão, reuso, adaptação, especialização e modularidade asseguram que *frameworks* podem ser adaptados a novos desenvolvimentos. *Frameworks* adotam diferentes abordagens para reuso, comumente denominadas *white box*, *black box*, ou *gray box* (combinação das duas primeiras). Na abordagem *white box*, as classes que compõem o *framework* são conhecidas, podendo ser especializadas através de herança. Já na abordagem *black box*, o *framework* não expõe a estrutura interna de suas classes, o que força o reuso através de composição (com classes externas). A abordagem *gray box* mescla as duas anteriores, onde o *framework* expõe algumas classes para reuso via herança, ocultando outras para reuso via composição.

Um exemplo de *framework* de classes pode ser encontrado em [6] onde os autores descrevem a implementação da especificação *A/V Streams* do OMG[4]. Esta especificação consiste de um *framework* de classes que foi estendido (via herança) para adaptá-lo às necessidades da sessão de comunicação de um serviço de telecomunicação.

Um padrão de projeto é um documento estruturado que descreve um problema a ser solucionado, uma solução e um contexto em que esta solução se enquadra. Por exemplo, o padrão de projeto *Observador* [5] estipula como implementar uma dependência do tipo um-para-muitos entre objetos de tal forma que quando um objeto tem seu estado alterado, todos os objetos dependentes são notificados da alteração. Padrões de projeto são os elementos micro-arquiteturais para a construção de *framework* de classes.

Componentes de Software

A noção de componente de software é um refinamento do modelo de objeto, sendo considerado uma extensão natural deste modelo [7]. Existem na literatura dezenas de definições de componentes de software. Adotaremos uma combinação das definições de Szyperski [4] e Meyer [7]: *um componente de software é uma unidade indivisível de composição e instalação que inclui uma especificação precisa de suas funcionalidades e*

[4]Object Management Group - http://www.omg.org

dependências. Por unidade de composição entende-se que um componente pode se compor de forma simples com outros componentes, formando assim uma unidade maior. Por unidade de instalação entende-se que um componente pode ser utilizado por terceiros, unicamente com base em sua especificação, e sem qualquer conhecimento sobre seu desenvolvimento ou intervenção de seu desenvolvedor.

As funcionalidades oferecidas pelo componente são acessadas através de interfaces que o componente expõe. A utilização destas interfaces é regida por especificações denominadas *contratos*. Um contrato descreve as condições para a utilização da interface, tanto do lado do cliente que a utiliza, quanto do lado do componente que a implementa.

Para garantir a composição e a instalação de componentes, estes devem ser implementados segundo um modelo bem definido. Exemplo de tais modelos incluem Enterprise Java Beans (SUN Microsystems), Active X (Microsoft), e CORBA Components (OMG) [8].

Frameworks de Componentes

Novamente, várias definições existem para *frameworks* de componentes. Adota-se também a definição de Szyperski [4]: *Um framework de componentes é uma arquitetura com um conjunto de interfaces*[5] *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.

Certos modelos de componentes denominam *containers* os *frameworks* que suportam seus componentes. A figura 1 estabelece as relações entre componentes, padrão de projeto, *frameworks* de classes e *frameworks* de componentes. A figura ilustra um componente implementado através do reuso de um *framework* de classes (que por sua vez contém a implementação de um padrão de projeto), sendo o componente suportado por um *framework* de componentes baseado em CORBA.

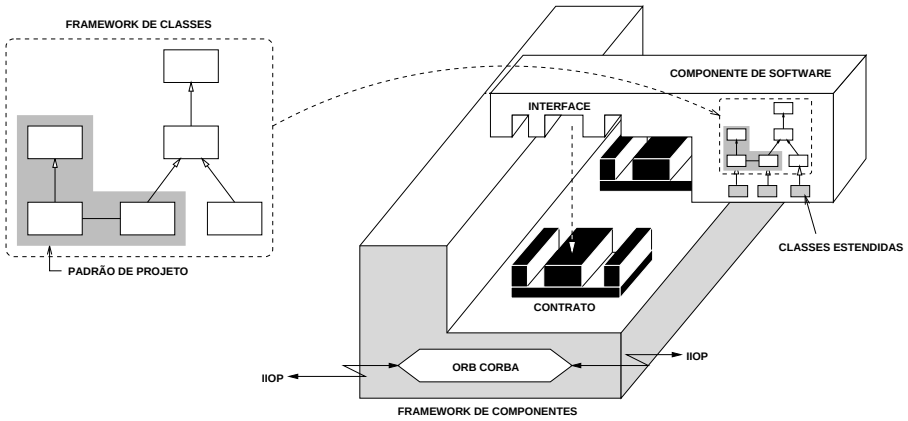

Figura 1: *Frameworks* e componentes de software.

[5]Especificadas através de contratos.

3 Uma Estratégia de Desenvolvimento de Software

O foco principal deste artigo é uma estratégia de desenvolvimento de software para novos serviços de telecomunicações. A idéia básica é *componentizar* o desenvolvimento de serviços de telecomunicações [9]. Resumidamente, um serviço é implementado através da integração de componentes de software. Uma analogia pertinente é o desenvolvimento de serviços para as redes inteligentes (IN) onde blocos funcionais (SIB[6]) são agregados para a realização de um serviço. A interconexão de SIBs é garantida pela aderência destes ao modelo de chamada estabelecido para a IN. Atualmente, o conceito de chamada em telefonia evoluiu para o conceito de sessão em serviços avançados sobre redes de dados. TINA[7] decompõe um serviço em três sessões: acesso, serviço e comunicação [10].

Da mesma forma que SIBs aderem a um modelo de chamada, os componentes do serviço devem aderir a um modelo de sessão. Cada sessão do serviço é implementada através da integração de componentes cuja interconexão é garantida por um modelo de componentes e uma arquitetura de suporte. Propostas para um modelo e uma arquitetura serão apresentadas, respectivamente, nesta e na próxima seção. O reuso de toda uma arquitetura distribuída e geração automática de código colaboram para um desenvolvimento rápido e de qualidade.

É importante observar que a estratégia não propõe um novo processo de desenvolvimento de software, mas a incorporação aos processos já existentes de um modelo e de uma arquitetura de componentes que simplifica a distribuição do sistema. A estratégia proposta possui algumas propriedades importantes:

- incorpora, via reuso, configuração e geração automática de código, uma arquitetura de software que contempla muitas das funções pertinentes à distribuição, tais como comunicação, segurança, configuração, notificação e transação;
- deixa a cargo do projetista apenas decisões relativas ao "empacotamento" (*packaging*) e à distribuição espacial dos componentes (*placement*);
- pode ser incorporada a qualquer processo de desenvolvimento de software orientado a objeto;
- favorece os procedimentos de teste do sistema (componentes são testados individualmente e sua integração com outros componentes é garantida pelo *framework* de componentes).

A estratégia aqui proposta consiste de três elementos:

1. um modelo de componentes;
2. uma arquitetura de componentes;
3. um conjunto (mínimo) de procedimentos para a incorporação do modelo aos processos de desenvolvimento de software.

[6]Service Information Block.
[7]Telecommunication Information Network Architecture - http://www.tinac.com

Modelo de Componentes de Software

Idealmente, um novo sistema pode ser construído através da interconexão de componentes de software oriundos de desenvolvimentos anteriores ou adquiridos no mercado. Entretanto, na prática, a interconexão de componentes requer regras de interação muito bem estabelecidas, onde apenas os componentes que seguem estas regras são passíveis de interconexão. Estas regras são definidas por um modelo de componentes que fornece as bases para a instalação e interação entre componentes. Este modelo prescreve as características dos componentes, indicando, por exemplo, as suas funcionalidades, interfaces, campos de aplicação, dependências e qualidade de serviço. O modelo impõe restrições de projeto para o desenvolvedor de componentes. Estas restrições são policiadas pelo *framework* de componentes que fornece suporte aos componentes aderentes ao modelo.

Um ponto fundamental em um modelo de componentes é a estrutura do componente. Em nosso modelo, os componentes possuem a estrutura ilustrada na figura 2.

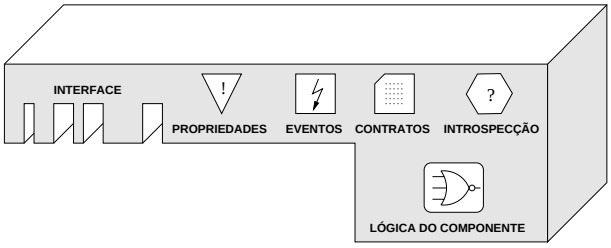

Figura 2: Estrutura de um componente de software.

Neste modelo, um componente é constituído pelos seguintes elementos:

- propriedades: definem um conjunto de "variáveis de estado" normalmente relacionadas à configuração e ao estado corrente do componente;
- eventos: são mensagens de notificação geradas por um componente e recebidas por zero ou mais componentes (utilizadas usualmente para o informe de status e falhas, bem como para sincronizar o processamento entre componentes);
- lógica do componente: consiste em um conjunto de métodos que implementam as funcionalidades precípuas do componente;
- interface: é o ponto de acesso às funcionalidades do componente;
- contratos: descrevem as condições para a utilização do componente através de sua interface;
- introspecção: permite inspecionar, em tempo de execução, a estrutura visível do componente (as propriedades que define, os eventos que produz e consome, os métodos associados à sua lógica e as suas interfaces).

No modelo proposto, os componentes definem um conjunto de propriedades que podem ser inspecionadas e alteradas por outros componentes (por exemplo, para fins de

reconfiguração dinâmica de componentes). Este conjunto é implementado através de um objeto denominado `PropertySet` conforme ilustra o diagrama UML[8] da figura 3. Propriedades são encapsuladas em objetos da classe `PropertyObject`. Estes objetos armazenam o nome da propriedade, seu valor corrente, bem como um conjunto de subscritores (*listeners*). Subscritores são objetos que se cadastram para serem notificados quando ocorrem alterações em determinadas propriedades. A subscrição/notificação segue o padrão de projeto *Observador* [5] de tal forma que quando uma propriedade armazenada em um objeto `PropertyObject` é alterada, todos os subscritores são notificados desta alteração.

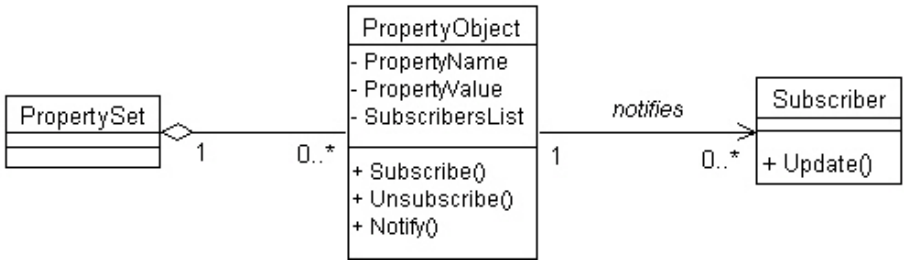

Figura 3: Estrutura de suporte à propriedades para o modelo de componentes.

O modelo de componentes emprega um serviço de notificação de eventos do tipo *push*, onde produtores de eventos tomam a iniciativa de notificar consumidores quando da ocorrência de eventos. Dois mecanismos de notificação de eventos são definidos: emissor-consumidor e publicador-consumidor (figura 4). No primeiro, é estabelecida uma relação um-para-um entre um produtor e um consumidor de eventos. No segundo, é estabelecida uma relação um-para-muitos onde eventos gerados por um produtor são distribuídos para vários consumidores. No modelo publicador-consumidor um canal de eventos conecta produtores e consumidores. O canal de eventos intermedia a subscrição de consumidores, bem como a difusão de eventos gerados por produtores (ou seja, um produtor envia um evento para o canal e este se encarrega de distribuir cópias para cada consumidor conectado). O modelo de subscrição/notificação implementado pelo canal de eventos também segue o padrão de projeto *Observador* [5].

Arquitetura de Componentes

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* [4]. Os *frameworks* de componentes são estruturados em camadas "horizontais" denominadas *tiers*. Uma arquitetura com N camadas horizontais é denominada arquitetura *N-tier*. Diferentes *tiers* ocupam-se de diferentes graus de interação, com diferentes graus de especialidade. Os *frameworks* de componentes localizados em determinado *tier* integram os *frameworks* de componentes (ou componentes) localizados em *tiers* mais específicos, e para tal utilizam recursos dos *tiers* mais genéricos.

[8]Unified Modeling Language. http://www.omg.org/uml

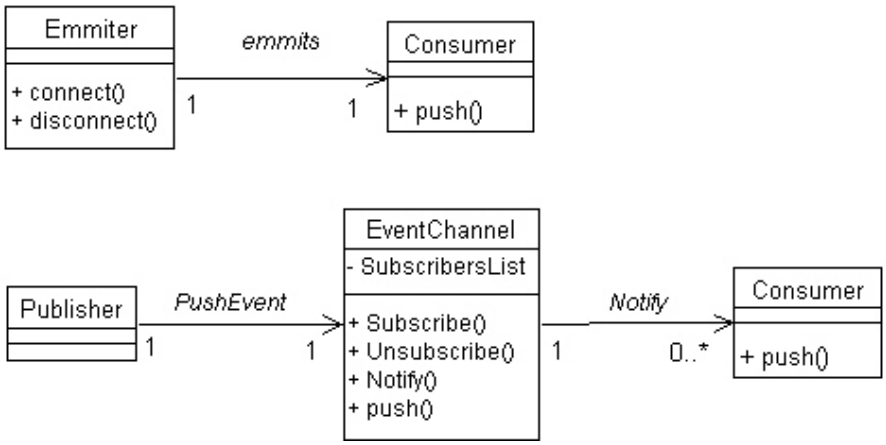

Figura 4: Mecanismos de difusão de eventos para o modelo de componentes: emissor-consumidor (acima) e publicador-consumidor (abaixo).

A figura 5 ilustra uma arquitetura *3-tier*. O primeiro *tier* consiste de um conjunto de componentes; o segundo consiste de *frameworks* de componentes responsáveis pela interconexão destes componentes; o terceiro consiste de um *framework* de *frameworks* de componentes que interconecta os *frameworks* presentes no segundo *tier*. Note que este padrão arquitetural é recorrente. Por exemplo, pode-se integrar várias arquiteturas *3-tier* em uma arquitetura *4-tier*. O *framework* localizado no último *tier* é denominado *framework de integração*.

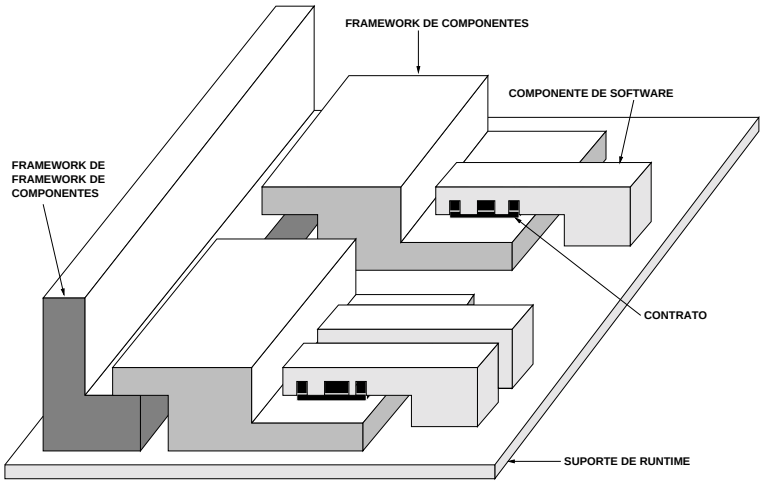

Figura 5: Arquitetura *3-tier* para suporte ao modelo de componentes.

Incorporação do Modelo aos Processos de Desenvolvimento de Software

O modelo proposto pode ser incorporado na fase de análise aos processos de desenvolvimento de software orientados a objeto. Para tal, componentes são modelados através de classes (por exemplo, expressas em UML. A figura 6 apresenta um componente denomi-

nado `Component` modelado por uma classe de mesmo nome. Uma classe base denominada `ComponentBase` fornece as funções comuns aos componentes, por exemplo, funções de ciclo de vida, serialização, mobilidade, segurança, etc. Componentes possuem classes agregadas para fins de manipulação de eventos, propriedades, introspecção e lógica do componente. Estas classes agregadas subdividem-se em duas categorias: de base (independente do componente) e específica (dependente) do componente. Exceto as classes que realizam a lógica do componente, as demais classes agregadas podem ser geradas a partir de uma descrição formal do componente.

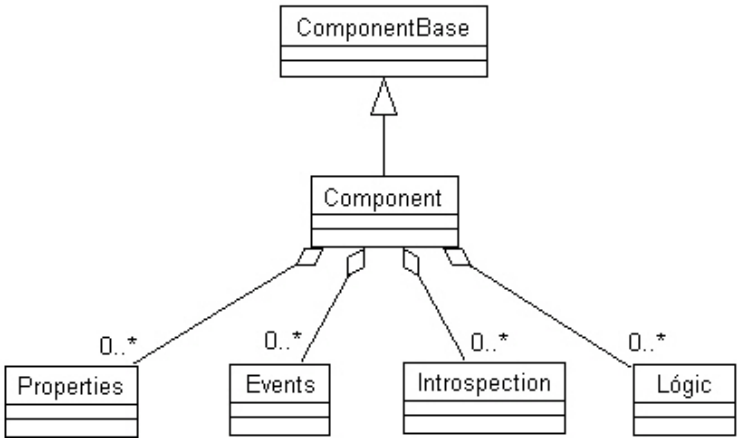

Figura 6: Representação na notação UML de um componente em modelos de análise.

4 Uma Arquitetura de Suporte a Componentes

Com base na arquitetura de componentes apresentada na figura 5, implementamos uma arquitetura *3-tier* baseada nas tecnologias CORBA, Java e XML (*eXtensible Markup Language*). A tecnologia CORBA é empregada para acesso remoto aos componentes, ou seja, manipulação de propriedades, propagação de eventos e chamada de métodos específicos do componente. Nesta arquitetura, as interfaces do componente são descritas em IDL (*Interface Definition Language*). Os Serviços de Propriedades e de Eventos do OMG são utilizados, respectivamente, para a manipulação remota de propriedades e difusão de eventos no mecanismo publicador-consumidor.

Java é utilizada como plataforma de *runtime*. XML é empregada para a especificação de contratos. Para tal, um DTD (*Document Type Definition*) determina a forma de declarar as características do componente (eventos, propriedades e métodos). A especificação de um contrato consiste em um conjunto de *tags* inspirados no padrão *CORBA Components* [8]. Foram acrescidos novos *tags* para contemplar as interfaces de manipulação de fluxo contínuo (como áudio e vídeo). A partir desta especificação, um *parser* XML gera automaticamente código IDL e Java. Este código possui todas as funções de distribuição embutidas, o que reduz significativamente a complexidade associada à codificação e à instalação de componentes.

O *parser* implementado é uma especialização de um parser XML denominado *XML Parser for Java*[9] (`xml4j`) compatível com a especificação 1.0 da linguagem XML. A figura 7 ilustra a mecânica de produção de componentes e *frameworks* de componentes. O código Java gerado pelo *parser* XML contém classes para a manipulação remota de propriedades e eventos definidos pelo componente, bem como as suas funções de introspecção[10]. Para completar o componente, o desenvolvedor acrescenta a este código a implementação dos métodos que realizam a sua lógica. O código destes métodos é o único que o projetista do componente necessita desenvolver, sendo totalmente isento de funções de distribuição.

O *framework* de componentes contém classes Java responsáveis pela distribuição dos componentes conectados. Estas classes são geradas pelo compilador IDL a partir das interfaces IDL que especificam os métodos do componente; as propriedades e eventos que o componente manipula, conforme geradas pelo *parser* XML; e os serviços CORBA de suporte (por exemplo, os Serviços de Propriedades e de Eventos).

Frameworks e seus componentes são "empacotados" em servidores CORBA e distribuídos em determinados nós da rede. A decisão de como empacotar e distribuir é de responsabilidade do projetista do serviço. Como regra geral, os componentes com funções de interfaceamento com usuário e a apresentação de mídia residem no terminal do usuário. Os demais componentes residem nos servidores do provedor do serviço.

O *framework* de integração consiste de um ORB (*Object Request Broker*) e um conjunto de serviços CORBA. No momento quatro serviços padronizados pelo OMG são empregados: Nomes, Propriedades, Eventos, e *A/V Streams* (fluxo multimídia) [6]. Exceto o Serviço de Nomes, os demais foram implementados pelos autores. Futuramente, pretende-se incorporar os Serviços de Segurança e de Transações, também do OMG.

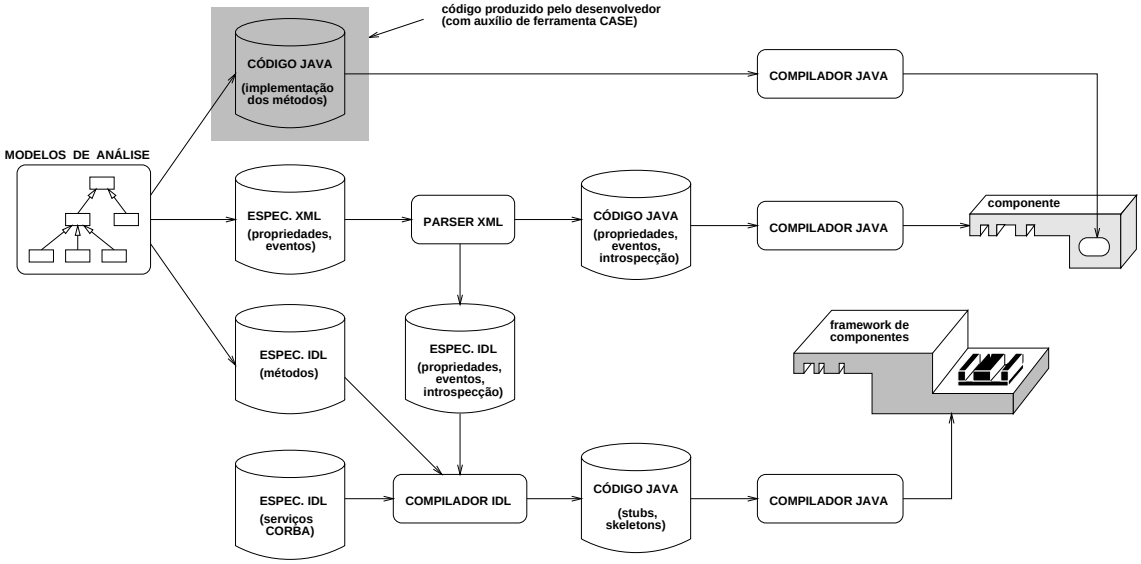

Figura 7: Ciclo de geração de componentes e *frameworks* de componentes.

[9]Disponível em http://www.alphaworks.ibm.com/tech/xml4j
[10]Atualmente, uma única função de introspecção retorna a especificação XML do componente.

5 Exemplo de Aplicação: REAL

Uma visão geral do projeto REAL (*REmotely Accessible Laboratory*) pode ser encontrada em [6], [11] e [12]. REAL é um laboratório de robótica com acesso remoto pela Internet, em desenvolvimento no ITI em cooperação com a UNICAMP. Este laboratório permite que usuários utilizem remotamente um robô móvel como se estivessem presentes fisicamente no laboratório. O laboratório é constituído dos seguintes componentes: um robô móvel XR4000; um sistema de aquisição de imagens a bordo do robô móvel e sua transmissão contínua ao usuário remoto; e uma interface homem-máquina para programação do robô e recebimento dos dados de sua operação.

O projeto REAL considera laboratórios virtuais como novos serviços de telecomunicações e não como aplicações WWW tradicionais. A razão para tal é a semelhança entre laboratórios virtuais e serviços de telecomunicações nos seguintes aspectos: i) os papéis de provedor e usuário do serviço são bem definidos; ii) requerem um rígido controle de subscrição, acesso e uso (e, em alguns casos, tarifação); iii) demandam sessão de comunicação com suporte a mídia contínua; iv) aspectos de segurança e privacidade devem ser amplamente considerados.

O projeto REAL utiliza a estratégia de desenvolvimento apresentada neste artigo. Componentes e *frameworks* utilizados em outros projetos foram incorporados ao REAL, assim como os desenvolvidos para o REAL poderão vir a ser reutilizados em outros projetos. Por exemplo, para a sessão de acesso, utiliza-se um *framework* aderente à Arquitetura de Serviço TINA [13], enquanto para a sessão de comunicação utiliza-se um *framework* aderente ao padrão *A/V Streams* do OMG [6]. Estes dois *frameworks* foram implementados em Java e incorporados ao REAL através de especialização.

Neste artigo, restringiremo-nos aos componentes que implementam a lógica do serviço. A lógica do serviço permite dois modos de interação com o robô: básico e avançado. No modo básico, o usuário seleciona, via interface gráfica, um alvo para onde o robô deve se dirigir. Utilizando um algoritmo próprio de navegação é determinada a melhor rota para se atingir o alvo, iniciando-se a seguir o movimento do robô ao longo da rota. O sistema de navegação reporta periodicamente à interface (através de eventos) a posição corrente do robô na rota. No modo avançado, o usuário submete um algoritmo de navegação (escrito na linguagem C) que é compilado remotamente e, em seguida, transferido e executado no computador de bordo do robô.

Neste artigo, vamos nos ater apenas ao subsistema de operação em modo avançado. Neste modo de operação, o usuário submete um programa de navegação ao REAL para compilação. A compilação e ligação com as bibliotecas de navegação do robô é feita em servidor no domínio do REAL. O resultado da compilação é armazenado no servidor HTTP do REAL e apresentado em seguida no *browser* do usuário. Caso a compilação seja livre de erros, o código executável é armazenado no servidor de navegação do robô. Posteriormente, o usuário pode iniciar, remotamente, a execução desse código. Durante a execução da missão, o usuário acompanha os movimentos do robô através de duas câmeras, uma posicionada no teto do ambiente e outra a bordo do robô. O vídeo proveniente destas duas câmeras é apresentado também no *browser* do usuário. A captura, transferência e

apresentação do vídeo é responsabilidade da sessão de comunicação. O transporte de fluxo contínuo de vídeo se dá através do protocolo RTP (*Real Time Protocol*) sobre UDP (*User Datagram Protocol*).

O subsistema de operação no modo avançado tem sua lógica implementada em dois componentes de software, um operando no lado do usuário remoto e outro no domínio do REAL. O componente que opera no lado do usuário remoto é implementado através de um *applet* Java que executa na máquina virtual do *browser* e implementa uma interface gráfica de interação com o cliente (figura 8). O componente que opera no domínio do REAL é implementado através de um processo Java que recebe e processa requisições do cliente.

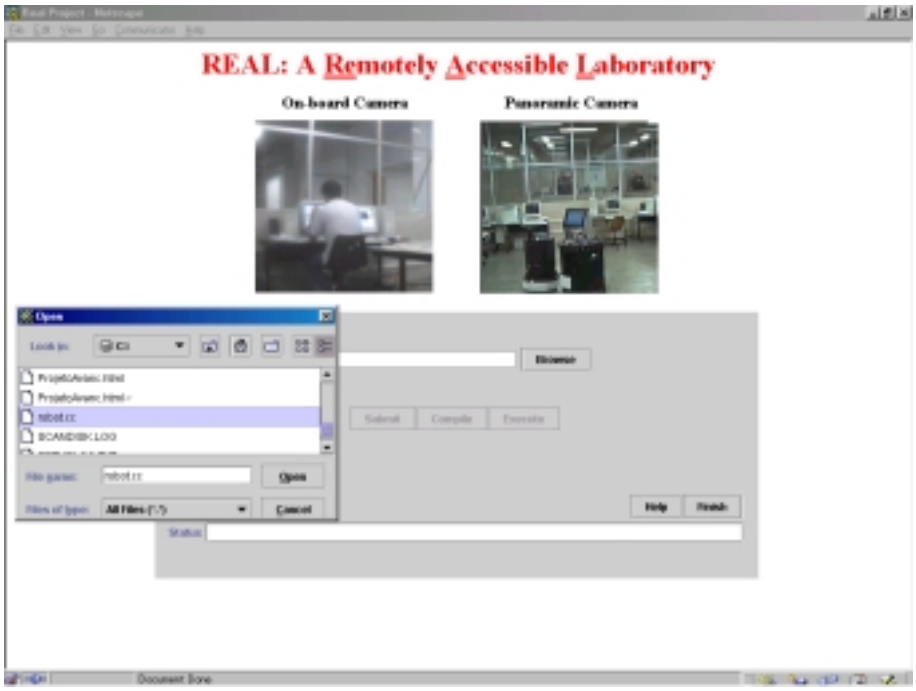

Figura 8: Uma das interfaces do modo de operação avançado.

O componente no lado cliente é especificado através de XML conforme ilustra a figura 9(a). Este componente, denominado `RA_ClientComponent` não define nenhuma propriedade (via *tag* `<properties>`), consome eventos do tipo `RA_Event` (*tag* `<consumes>`) e exporta o método `push` (*tag* `<provides>`). Eventos são utilizados para sincronização (término de compilação, por exemplo) e exceções diversas (violação de segurança, término do tempo reservado à sessão de acesso, etc.).

O componente no lado servidor é especificado através de XML conforme ilustra a figura 9(b). Este componente, denominado `RA_ServerComponent` define as propriedades `ServerVersion` (apenas de leitura) e `ServerMode`. O componente produz eventos do tipo `RA_Event` (*tag* `<emits>`) e exporta métodos para gerência de arquivos.

A figura 10 apresenta um modelo dos componentes do modo avançado de operação na notação UML. As classes `RA_ClientComponent` e `RA_ServerComponent` modelam os componentes de mesmo nome e derivam de `ComponentBase`, uma classe base para todos

```
<?xml version="1.0"?>
<!DOCTYPE corbacomponent SYSTEM "corbacomponent.dtd">
<corbacomponent>
   <componentfeatures name="RA_ClientComponent"
                      repid="IDL:RA_Client:1.0">
   <ports>
     <provides
        providesname="push"
        repid="IDL:RA_Client:1.0">
        <operation name="push" />
     </provides>
     <consumes
       consumesname=RA_ClientConsumer"
       eventtype="RA_Event>
     </consumes>
   </ports>
</corbacomponent>
```

(a)

```
<?xml version="1.0"?>
<!DOCTYPE corbacomponent SYSTEM "corbacomponent.dtd">
 <corbacomponent>
   <componentfeatures name="RA_ServerComponent"
                      repid="IDL:RA_Server:1.0">
  <ports>
     <provides
       providesname="FileMgmt"
       repid="IDL:RA_Server:1.0">
       <operation name="transferFile" />
       <operation name="compileFile" />
       <operation name="runFile" />
       <operation name="cancelRunFile" />
       <operation name="getBinaryFiles" />
       <operation name="getSourceFiles" />
     </provides>
     <emitts
        emitsname="RA_Emitter"
        eventtype="RA_Event">
     </emits>
  </ports>
  <properties>
     <simple mame="ServerVersion" type="string" rights="readonly">
        <defaultvalue>"1.0"</defaultvalue>
     </simple>
     <simple name="ServerMode" type="long">
        <defaultvalue>0</defaultvalue>
     </simple>
   </properties>
</corbacomponent>
```

(b)

Figura 9: Especificações XML dos componentes RA_ClientComponent (a) e RA_ServerComponent (b).

os componentes. As classes `RA_ClientComponent` e `RA_ServerComponent` agregam outras classes geradas pelo *parser* XML, classes estas que permitem manipular eventos e propriedades conforme definidos pelos componentes.

No caso de eventos, ao processar o *tag* `<emits>` o parser XML gera a classe `RA_Event` a partir do *tag* `<eventtype>`. Esta classe deriva da classe `EventBase`, uma classe base para eventos. Outra classe gerada é `RA_EventEmitter` que contém os métodos `connect` e `disconnect` que permitem a um consumidor de eventos conectar-se a este (desconectar-se deste) emissor de eventos.

Analogamente, o processamento do *tag* `<consumes>` (declarado na especificação do componente `RA_ClientComponent`) pelo *parser* gera a interface `RA_EventConsumer` que declara o método `push`. Este método é utilizado pelo emissor para notificar a ocorrência de um evento (no caso, uma nova posição do robô).

A manipulação de propriedades ocorre através de duas classes, uma de base (`PropertySet`), e outra gerada pelo *parser* XML (`PropertyManipulator`). A classe `PropertySet` é utilizada pelo componente que define propriedades. Esta classe permite acessar objetos que armazenam as propriedades e, a partir destes, gerenciá-las[11]. A classe `PropertyManipulator` define métodos do tipo `get` e `set` utilizados por um componente para ler e alterar as propriedades declaradas por outros componentes. Métodos do tipo `set` não são gerados quando a propriedade for apenas para leitura (como o caso da propriedade `ServerVersion` que contém o atributo `rights` com valor `readonly`).

[11] Por exemplo, adicionar um subscritor que é notificado quando a propriedade tem seu valor alterado.

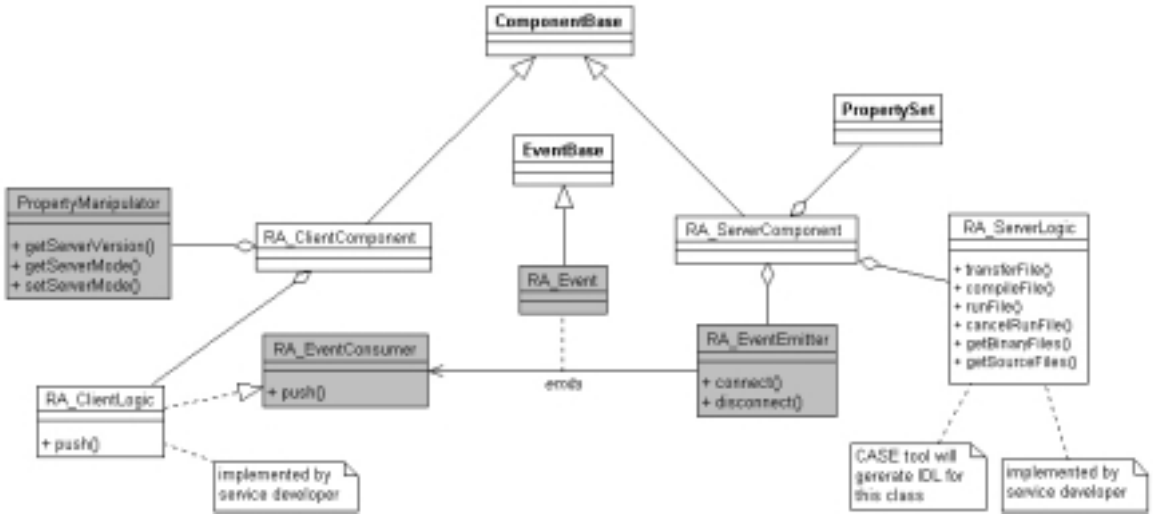

Figura 10: Modelo para o modo avançado de operação do REAL. Classes sombreadas são geradas pelo *parser* XML. Classes com nome em negrito são classes de base.

Finalmente, para o caso de métodos, a declaração XML informa os métodos que o componente provê e sua localização no repositório de interfaces do CORBA, onde encontram-se armazenadas as definições dos métodos (protótipos). Estes métodos devem ser implementados pelo desenvolvedor do serviço (classes `RA_ServerLogic` e `RA_ClientLogic` da figura 10). Entretanto, conforme já enfatizado, nenhuma consideração sobre distribuição necessita ser levada em conta na implementação destes métodos.

6 Conclusões

Este artigo apresentou uma estratégia de desenvolvimento de software orientado a componentes para novos serviços de telecomunicações. Foi proposto um modelo de componentes e uma arquitetura de componentes que promovem o reuso de software distribuído na forma de *frameworks* e componentes. Outro aspecto importante está na geração de código Java e IDL para componentes especificados em XML. A proposta de desenvolvimento de serviços apresentada alinha-se com tendências atuais neste campo que claramente apontam para a utilização de modernas técnicas de desenvolvimento de software distribuído, utilização de produtos "de prateleira", padrões *de facto* e capacidade de integração com sistemas legados. O projeto REAL, um serviço de laboratório virtual acessível através da Internet, vem demonstrando a utilidade da proposta aqui apresentada.

Como trabalhos futuros, um ambiente de criação de serviços tendo o *parser* XML como seu elemento inicial está sendo considerado. Da mesma forma, planeja-se utilizar o modelo e a arquitetura propostos em projetos nas linhas de qualidade de serviço e gerência proativa de redes de computadores.

Agradecimentos

Este projeto tem o suporte das seguintes agências: FAPESP (procs. 97/13384-7 e 99/09922-9), CNPq (procs. 300723/93-8 e 108615/00-6), FINEP (proc. 1588/96). Os autores são gratos também aos revisores do artigo pelas valiosas sugestões.

Referências

[1] Vanecek, G., Mihai, N., Vidovic, N., Vrsalovic, D., *Enabling Hybrid Services in Emerging Data Networks*, IEEE Communications Magazine, Vol. 37, N. 7, Jul 1999.

[2] Jacobson, I., et. al., *The Unified Software Development Process*, Addison Wesley, 1999.

[3] *Component Based Enterprise Framework*, Communications of the ACM, Vol. 43, No 10, Out 2000.

[4] Szyperski, C., *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, 1998.

[5] Gama, E. et. al., *Design Pattern : Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[6] Guimarães, E., Cardozo, E., Bergerman, M, Magalhães, M., *Um Framework para Transmissão de Áudio e Vídeo para Novos Serviços de Telecomunicações*, 18º SBRC, Belo Horizonte, MG, Maio 2000.

[7] Meyer, B, *What to Compose*, Software Development Online, Beyond Objects Column, março de 2000, http://www.sdmagazine.com/features/uml/beyondobjects/

[8] Object Management Group, *CORBA Components - Volume 1*, http://www.omg.org.

[9] Bakker, J., Batteram, H., *Design and Evaluation of the Distributed Software Component Framework for Distributed Communication Architectures*, 2nd ACM Int. Enterprise Distributed Object Computing Workshop, San Diego, USA, Nov 1998.

[10] Berndt, H., et. al, *The TINA Book*, Prentice-Hall Europe, 1999.

[11] Guimarães, E., Cardozo, E., Bergerman, M, Magalhães, M., *Um Framework de Transmissão de Áudio e Vídeo para Laboratórios de Acesso Remoto*, XIII Congresso Brasileiro de Automática, Florianópolis, SC, Set 2000.

[12] Guimarães, E., Bergerman, M., Pereira, J., Russo, B., Maffeis A., Cardozo, E., Magalhães. M., *REAL: A Virtual Laboratory for Mobile Robot Experiments*, First IFAC Conference on Telelematics Applications in Automation and Robotics, Weingarten, Alemanha, Jul 2001.

[13] Oliveira, E.J., Faina, L.F., Cardozo, E., *Sessões de Acesso e Serviço TINA Baseadas em Web*, 17º SBRC, Salvador, Brasil, Maio 1999.