CCM-tel - uma Plataforma para Aplicações Telemáticas e Ubíquas

Eliane G. Guimarães[1], Eleri Cardozo[2], Mauricio F. Magalhães[2], Wander P. Gomes[2], Rossano P. Pinto[2], Luis F. Faina[3]

[1]Centro de Pesquisas Renato Archer - CenPRA
Rodovia D. Pedro I, Km 143,6
13082-120 Campinas, SP

[2]Faculdade de Engenharia Elétrica e de Computação - FEEC
Universidade Estadual de Campinas - Unicamp
13083-970 Campinas, SP

[3]Faculdade de Computação - FACOM
Universidade Federal de Uberlândia - UFU
38400-902 Uberlândia, MG

eliane.guimaraes@cenpra.gov.br

Abstract. *This paper presents a middleware platform able to fulfill an important set of requirements imposed by the emerging applications, for instance, support for multimedia communication with quality of service, mobile code, and dynamic adaptation to the environment. The platform favors a component-based software development according to a novel component model. This component model is neutral in terms of technology and completely specified in UML (Unified Modeling Language). An example of a telematic application in the field of virtual laboratories employing multimedia communication and mobile code illustrates the capabilities of the platform.*

Resumo. *Este artigo apresenta uma plataforma de middleware capaz de atender um importante conjunto de requisitos impostos pelas aplicações emergentes, por exemplo, suporte para comunicação multimídia com garantias de qualidade de serviço (QoS), código móvel e adaptação dinâmica ao ambiente. A plataforma favorece o desenvolvimento de software baseado em componentes de acordo com um novo modelo de componentes. Este modelo de componentes é neutro em termos de tecnologia e especificado inteiramente em UML (Unified Modeling Language). Um exemplo de aplicação telemática na linha de laboratórios virtuais que emprega comunicação multimídia distribuída e código móvel ilustra as funcionalidades da plataforma.*

1. Introdução

Plataformas de *middleware* convencionais oferecem uma infra-estrutura para o suporte a distribuição de objetos através de uma rede de comunicação e consistem de um conjunto de funcionalidades tais como, linguagens de definição de interfaces (IDL- *Interface Definition Language*), ORBs (*Object Request Brokers*), facilidades para a geração e

manutenção de código e diversos serviços de suporte às aplicações tais como, serviços de nomes, segurança e transação.

CORBA (*Common Object Request Broker Architecture*), Java RMI (*Remote Method Invocation*) e DCOM (*Distributed Component Object Model*) são tecnologias representativas de tais infra-estruturas. Embora largamente empregadas na indústria e academia, estas plataformas de *middleware* apresentam algumas desvantagens. Primeiramente, a aplicação é responsável pelo correto uso dos serviços de suporte. Por exemplo, uma aplicação que emprega mobilidade de código possui todas as chamadas para o serviço de agentes móveis embutidas em seu próprio código. Isto demanda do desenvolvedor um profundo conhecimento sobre o serviço (uso, limitações, desempenho, etc.). Em segundo lugar, a facilidade de geração automática de código oferecida por estas plataformas é restrita a *stubs* e *skeletons*. Em terceiro lugar, estas plataformas não oferecem nenhum suporte para a configuração dinâmica e para a distribuição das aplicações. Finalmente, o reuso de código é limitado, uma vez que a granularidade dos objetos distribuídos é baixa. Estas desvantagens restringem o uso das plataformas de *middleware* existentes no desenvolvimento de aplicações emergentes tais como, aplicações telemáticas, aplicações móveis e aplicações multimídia distribuídas. Estas aplicações demandam novas tecnologias de desenvolvimento de *software*. Adicionalmente, a mobilidade tem sido uma característica cada vez mais importante, disponibilizada pelas redes sem fio e terminais portáteis tais como, computadores de mão (*palmtops*) e telefones celulares. Tripathi [Tripathi, 2002], apresenta os quatro principais requisitos impostos por estas aplicações: suporte para a integração de componentes em oposição à programação; suporte para novos paradigmas de computação distribuída, principalmente aqueles baseados em código móvel e em tecnologias de agentes móveis; suporte para ambientes sensíveis a contexto; e, suporte para interações multimídia em tempo real.

O suporte para os requisitos listados acima é um dos principais desafios impostos para o projeto de novas plataformas de *middleware*. Plataformas de *middleware* suportando componentes de *software* são um passo importante em direção a um amplo suporte aos requisitos listados acima. Componentes de *software* são considerados uma extensão natural do modelo de objetos distribuídos [Heineman and Council, 2001]. Os modelos de componentes existentes tais como, EJB (*Enterprise Java Beans*))[1]da SUN, CCM (*CORBA Component Model*))[2] do OMG (*Object Management Group*) e COM+[3] da Microsoft suportam os requisitos de transação, segurança e persistência, sendo mais direcionados para aplicações no domínio de negócios.

Infelizmente, os requisitos relacionados a qualidade de serviço, mobilidade de código e adaptação da aplicação a ambientes dinâmicos não são suportados por estes modelos de componentes. A falta de suporte a estes requisitos restringe o uso destes modelos de componentes comerciais no desenvolvimento de muitas aplicações modernas. Adicionalmente, estes modelos 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 na utilização das respectivas tecnologias vinculadas.

---

[1] `http://java.sun.com/products/ejb/`

[2] `http://www.omg.org/technology/`

[3] `http://www.microsoft.com/com/tech/COMPlus.asp/`

As deficiências apontadas acima motivaram-nos a propor um novo modelo de componentes que atenda os principais requisitos impostos pelas novas aplicações. A chave deste modelo é a sua independência de tecnologia. Em vez de especificar um modelo de componentes para uma tecnologia específica como usual, adotamos uma abordagem diferente. Esta abordagem consiste em especificar um modelo de componentes em dois passos. O primeiro passo, define um modelo genérico, estável e neutro em termos de tecnologia (linguagens de programação, plataformas de *middleware*, sistemas operacionais, protocolos de rede, etc.). Este modelo neutro é descrito inteiramente em UML (*Unified Modeling Language*). O segundo passo, especifica um mapeamento do modelo de componentes proposto para uma tecnologia alvo tal como, CORBA, Java RMI ou DCOM. Um dos benefícios desta abordagem é a flexibilidade obtida, uma vez que a especificação do modelo de componentes bem como dos componentes da aplicação permanecem estáveis, enquanto que a implementação destes em uma tecnologia pode evoluir ou ser substituída de forma independente. Este modelo de componentes pode interoperar com outros modelos existentes se os mecanismos de comunicação forem os mesmos. Por exemplo, o modelo proposto mapeado para CORBA pode interoperar com o modelo CCM que também utiliza CORBA como mecanismo de comunicação.

Este artigo apresenta na seção 2 um novo modelo de componentes denominado CM-tel. A seção 3 descreve CCM-tel, uma plataforma de suporte para este modelo desenvolvida sobre as tecnologias CORBA e Java. A seção 4 apresenta o suporte oferecido pela plataforma para monitoramento e controle de qualidade de serviço (QoS) em aplicações multimídia distribuídas. A seção 5 descreve uma aplicação para monitoramento e controle de QoS baseada em agentes móveis utilizando os recursos propiciados pela plataforma CCM-tel. Finalmente, a seção 6 tece considerações finais sobre o trabalho.

2. O Modelo de Componentes CM-tel

Modelos de componentes como EJB e CCM proporcionam somente dois tipos de interfaces de componentes definidas no modelo de referência para processamento distribuído aberto RM-ODP [ISO-IEC, 1995]: interfaces operacionais e de sinal. A terceira interface prescrita pelo RM-ODP é a interface de fluxo contínuo. Esta interface permite aos componentes manipular mídia contínua (fluxos de áudio e vídeo). Por exemplo, se o modelo de componentes suporta esta interface, uma sessão multimídia pode ser estabelecida com uma única operação conectando um componente consumidor de mídia a um componente produtor de mídia. Modelos de componentes que não suportam esta interface impedem as aplicações de especificar, no nível de componentes, as interações de tempo real baseadas em comunicação multimídia. Como conseqüência, a incorporação de comunicação multimídia dentro das aplicações segue uma solução desvinculada do modelo de componentes.

O modelo CM-tel [Guimarães et al., 2003b] compartilha algumas características comuns aos modelos de componentes existentes, principalmente com CCM. Entretanto, algumas diferenças são dignas de nota. CM-tel é neutro em termos de especificação e tecnologia, suporta de forma integrada as três interfaces prescritas pelo RM-ODP, inclusive interfaces de fluxo contínuo, e permite que atributos de QoS sejam especificados em alto nível para estas interfaces. CM-tel é um modelo de componentes bem mais simples do que CCM, o que favorece o mapeamento e a construção de plataformas de *middleware*

leves que suportam o modelo. Por exemplo, uma plataforma CM-tel pode executar em um dispositivo móvel com baixo poder de processamento.

Um ponto fundamental no modelo de componentes CM-tel é a estrutura do componente de *software*. A estrutura de um componente CM-tel é ilustrada na figura 1.

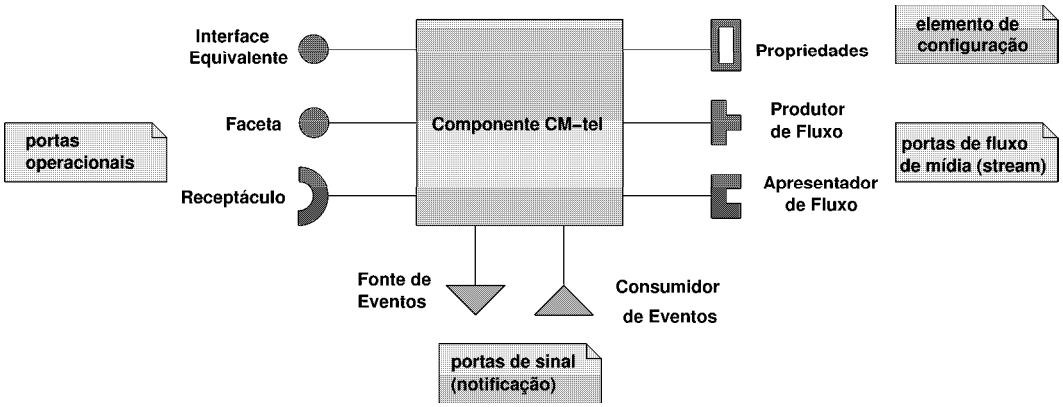

Figura 1: Estrutura de um componente de software CM-tel.

Componentes CM-tel são entidades de implementação e composição que podem ser instaladas de forma independente. Para permitir a composição, os componentes expõem as suas interfaces para os seus clientes. Estas interfaces são referenciadas como portas e executam papéis complementares tais como, *cliente/servidor* e *publicador/subscritor*. As portas complementares são conectadas durante a fase de configuração da aplicação ou dinamicamente em tempo de execução. Seis tipos de portas são definidas pelo modelo:

- facetas: interfaces operacionais que suportam operações síncronas;
- receptáculos: guardam as referências (facetas, componentes, objetos) requeridas pelo componente;
- fontes de eventos: interfaces de sinal capazes de produzir mensagens de notificação assíncronas (eventos);
- consumidores de eventos: interfaces de sinal capazes de consumir eventos quando conectadas às fontes de eventos;
- produtores de fluxos: interfaces capazes de gerar e transmitir fluxos de mídia contínua (áudio e vídeo);
- apresentadores de fluxos: interfaces capazes de receber e apresentar fluxos de mídia quando conectadas aos produtores de fluxos.

Facetas definem métodos contendo a lógica da aplicação. As demais portas definem métodos tipo *connect/disconnect* empregados na interconexão de portas complementares. Em adição às facetas, duas interfaces operacionais são suportadas:

- interface equivalente: interface que identifica unicamente uma instância de um componente e permite o acesso às suas demais interfaces;
- propriedades: interfaces que proporcionam acesso para um conjunto de pares atributo-valor que armazenam informações sobre o estado e configuração do componente.

Um componente CM-tel é especificado em UML através de um conjunto de estereótipos (ou perfil UML) que associam classes UML às interfaces do componente descritas acima. A figura 2 ilustra a especificação UML para componentes CM-tel. O diagrama UML da figura 2 é traduzido em um *schema* XML (*Extensible Makup Language*)[4] de acordo com regras de equivalência UML-XML propostas em [Carlson, 2001]. Ferramentas como HyperModel[5] geram automaticamente *schemas* XML a partir de perfis UML. Este *schema* XML especifica componentes CM-tel em XML.

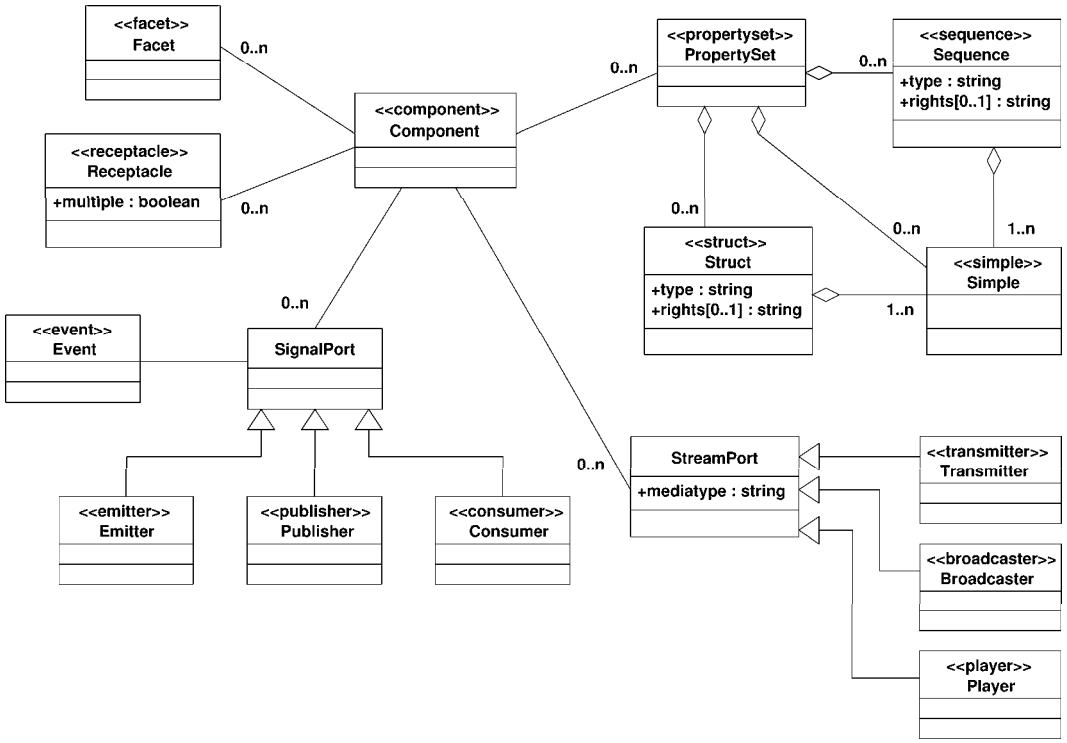

Figura 2: Especificação UML para componentes de software CM-tel.

Agentes móveis são segmentos de código capazes de se mover de uma localização para outra na rede, baseado em decisão própria. Agentes móveis e componentes podem interagir através das funcionalidades propiciadas pelas portas dos componentes ou métodos de interação propiciados pelo agente. O modelo CM-tel também define um estereótipo UML para classes que implementam agentes móveis.

Um *container* representa o ambiente de execução para um conjunto de componentes e agentes móveis. *Containers* proporcionam os recursos necessários a execução dos componentes e agentes móveis nele instalados, por exemplo, recursos para processamento, comunicação e armazenamento. Tal como componentes, *containers* CM-tel são formalmente descritos através de um perfil UML com um *schema* XML equivalente. A especificação define os tipos de componentes e de agentes móveis que são instanciados no *container*. Para cada componente, as suas dependências em relação a outros componentes também são especificadas. Da mesma forma, para cada agente é especificado o tipo da agência, instalada no *container* e responsável pela gerência do ciclo de vida do agente

[4] `http://www.w3c.org/XML/`
[5] `http://www.hypermodel.com`

móvel.

Containers são instalados em um conjunto de nós da rede de acordo com especificações denominadas descritores de distribuição. Um descritor de distribuição define as características e os atributos necessários para configurar o *container* durante a sua instalação. Um conjunto importante de atributos de distribuição para serviços telemáticos consiste nos parâmetros de qualidade de serviço para as interfaces de fluxo contínuo. Os descritores de distribuição são especificados em XML.

3. A Plataforma CCM-tel

Dado que o modelo CM-tel é neutro, plataformas de suporte para este modelo devem mapear o modelo para uma dada tecnologia. Tais mapeamentos são conduzidos em duas etapas. A primeira etapa consiste em transformar a especificação UML de componentes e *containers* para uma representação intermediária à geração de código, no caso, uma representação XML. A segunda etapa transforma a representação XML em código de componentes e *containers* em uma dada tecnologia.

Estas duas etapas são especificadas através de transformações na linguagem XSLT (*Extensible Stylesheet Language Transformation*)[6]. 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 em XML as regras de transformação de documentos XML em outros documentos (por exemplo, documentos HTML).

Na primeira etapa de transformação, as representações de componentes e *containers* em UML são convertidas com o auxílio de ferramentas de projeto de *software* para uma representação XML equivalente segundo o padrão XMI (*XML Metadata Interchange*). Folhas de estilo XSLT processam as transformações do documento XMI para documentos XML com sintaxe regida pelos *schemas* XML definidos para componentes e *containers* a partir do respectivo perfil UML como o ilustrado na figura 2. Estas transformações são totalmente independentes de tecnologia.

Na segunda etapa, as especificações XML de componentes e *containers* produzidas na primeira etapa sofrem várias transformações, cada qual gerando um documento contendo código para uma dada tecnologia. Estas transformações são, portanto, dependentes da tecnologia alvo. O código é então compilado e empacotado por uma máquina de construção de código.

A plataforma CCM-tel implementa o modelo CM-tel para as tecnologias CORBA e Java fornecendo, portanto, um mapeamento do modelo CM-tel para estas tecnologias. Esta plataforma consiste de uma máquina de transformação XSLT que realiza as duas etapas necessárias à geração de código, um conjunto de serviços CORBA, um conjunto de classes utilitárias para suporte à gerência de componentes (classes comuns) e uma máquina de construção de código.

Os serviços CORBA suportam a implementação das portas e propriedades dos componentes, bem como dos agentes móveis. Além do serviço de nomes, os serviços de

[6] `http://www.w3c.org/xslt/`

eventos, de gerência e controle de fluxo de áudio e vídeo (A/V Streams), de propriedades e de agentes móveis (MAF) são disponibilizados.

A máquina de transformação emprega o processador XSLT Xalan-Java[7] enquanto a máquina de construção de código emprega a ferramenta Ant[8], ambos fornecidos pela *Apache Software Foundation*. A plataforma gera arquivos Java e IDL que implementam componentes e *containers*. Todo o código necessário para a manipulação dos serviços CORBA, infra-estrutura de composição de componentes e gerência e comunicação entre componentes e agentes é gerado pela plataforma. A este código gerado o desenvolvedor acrescenta o código que implementa os métodos das facetas e os agentes móveis. O processo de montagem conduzido pela ferramenta Ant consiste na compilação dos arquivos IDL e Java, empacotamento no formato JAR e assinatura do código para distribuição através de *applets* Java. Adicionalmente, a plataforma CCM-tel gera *shell scripts* e arquivos HTML para a instalação de *containers*, bem como arquivos XML de construção com diretivas de compilação e montagem para a ferramenta Ant.

A figura 3 ilustra os elementos que compõem a plataforma CCM-tel.

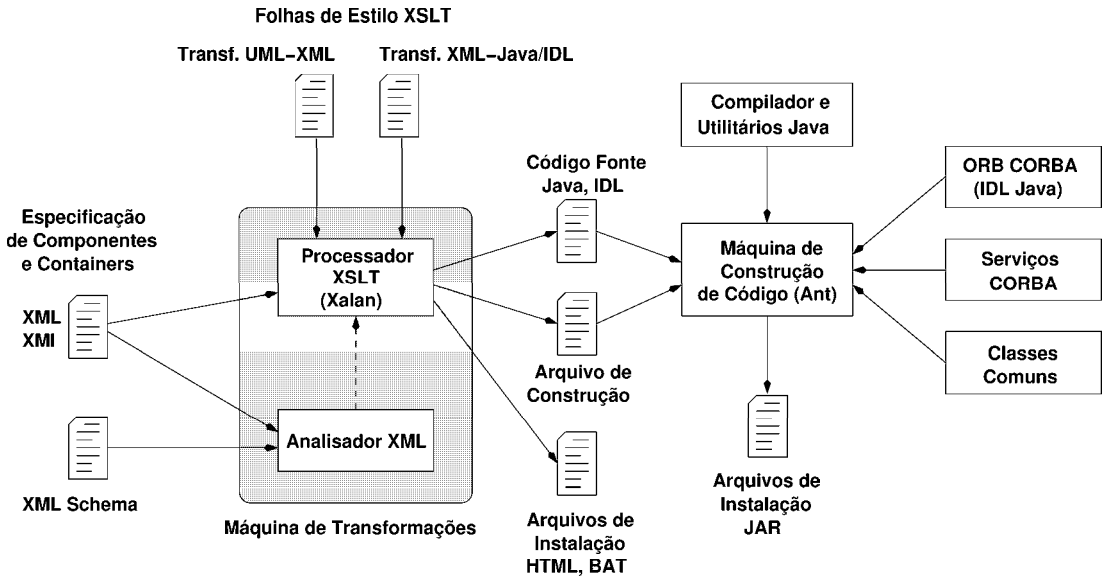

Figura 3: Elementos da plataforma CCM-tel.

A arquitetura de um *container* CCM-tel disponibiliza fábricas de componentes e agências (fábricas de agentes) para os tipos de componentes e agentes móveis especificados para o *container*. O *container* possui ainda um objeto localizador de fábricas através do qual pode-se obter as referências das fábricas de componentes e agentes e, a partir destas, dos componentes e agentes por elas gerenciados. Artefatos da arquitetura CORBA como o ORB, o adaptador de objetos portável (POA) e interceptadores portáveis são instanciados pelo *container* e disponibilizados para os componentes e agentes abrigados. Uma infra-estrutura de acesso aos serviços CORBA é incorporada ao *container* e utilizada pelos objetos que implementam componentes e agentes. Finalmente, um conjunto de propriedades de distribuição é disponibilizada pelo *container*. Através destas

[7] `http://xml.apache.org/xalan-j/`
[8] `http://jakarta.apache.org/ant/`

propriedades, um componente ou agente interno ou externo ao *container* pode alterar os parâmetros definidos originalmente pelo descritor de distribuição. A alteração destas propriedades pode afetar os componentes e agentes existentes ou que venham a ser instanciados no *container*.

Interação Componentes-Agentes Móveis

Containers CCM-tel podem gerenciar agentes móveis através de agências. Agências utilizam a Facilidade de Agentes Móveis do CORBA (MAF) para esta gerência. Agências expõem uma interface com operações para: instanciação de agentes móveis; migração de agentes para outra agência; acesso aos agentes gerenciados pela agência; controle de execução de agentes (suspensão e retomada de execução); e, destruição de agentes.

A plataforma CCM-tel implementa um sistema de agentes leve e de baixo *overhead* na linguagem Java e compatível com a especificação MAF. Este sistema de agentes emprega serialização e carregamento dinâmico de classes (local ou via protocolo HTTP), ambos nativos da linguagem Java. Agentes CCM-tel devem implementar duas interfaces Java: *Runnable* e *Agent*. A interface *Runnable* permite aos agentes executar como *threads* Java. A interface *Agent* define cinco métodos de *callback* que permitem a uma agência controlar seus agentes.

Agentes móveis podem empregar todos os recursos da tecnologia CORBA, tal como acessar os serviços mantidos pelo ORB (por exemplo, o serviço de nomes), acessar o ORB como cliente ou registrar alguns de seus objetos no adaptador de objetos portável como servidores CORBA. Esta capacidade dos agentes permite que estes 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 do componente ou invocar métodos definidos pelas interfaces. Da mesma forma, componentes podem localizar e interagir com agentes móveis. Adicionalmente, os agentes móveis podem executar novas funções ausentes no código original do *container*, possibilitando estendê-lo e adaptá-lo em tempo de execução; podem efetuar ações de gerência de *containers*, por exemplo, identificando os componentes hospedados e suas conexões; e, reconfigurar *containers* alterando suas propriedades.

4. Suporte à Qualidade de Serviço

O suporte à qualidade de serviço (QoS) para aplicações multimídia distribuídas consiste de duas atividades: reserva de recursos e monitoramento e controle de QoS.

A plataforma CCM-tel implementa reserva de recursos através de interceptadores portáveis CORBA. Interceptadores são segmentos de código que efetuam pré ou pós-processamento de requisições de métodos encaminhadas através do ORB. Um interceptador portável, *IQoS*, efetua pós-processamento das requisições que estabelecem conexão entre portas de fluxo contínuo. *IQoS* extrai os parâmetros de QoS da requisição (qualidade de áudio ou vídeo) e o endereço IP de destino do fluxo, converte-os em recursos de rede (banda), e interage com um *Bandwidth Broker* para proceder a reserva de recursos no nível de rede. Caso a rede não suporte negociação de QoS, os fluxos de mídia contínua

serão encaminhados com a política de melhor-esforço. Este mecanismo de reserva de recursos foi descrito em detalhes em [Pinto et al., 2003].

A segunda atividade, monitoramento e controle de QoS, será detalhada neste artigo. Esta atividade consiste em monitorar os 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, pode-se monitorar a taxa de bits em uma porta apresentadora de vídeo e, caso esta taxa exceda a taxa reservada, efetuar uma diminuição da qualidade do vídeo na porta transmissora (aumentando-se a taxa de compressão ou diminuindo-se a resolução espacial do quadro, por exemplo).

A plataforma CCM-tel não provê um mecanismo interno para monitoramento e controle de QoS, mas sim facilidades para a implementação de tal mecanismo no nível da aplicação. Esta decisão deve-se ao fato de que estes mecanismos são altamente dependentes da aplicação e do ambiente de rede na 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. A plataforma CCM-tel disponibiliza um conjunto de propriedades para monitoramento e controle de QoS que são descritas na seqüência.

Ao especificar um componente com portas produtora e apresentadora de fluxo contínuo, o projetista 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 de manipulação de QoS, cujos valores são obtidos do gerente de sessão (*Session Manager* disponibilizado pelo *Java Media Framework*[9], 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* e fração de bytes perdidos - propriedade *RtcpFractionLost*.

Os valores das propriedades acima, obtidos do gerente de sessão, são atualizados a cada 2 segundos. Os dois primeiros parâmetros são obtidos localmente e correspondem à taxa de bits ou pacotes gerados pela porta produtora ou consumidos pela porta apresentadora. Os três parâmetros restantes são obtidos de relatórios enviados através do protocolo RTCP (*Real Time Control Protocol*). Estes relatórios são trocados entre produtores e apresentadores de mídia contínua que empregam o protocolo RTP (*Real Time Protocol*).

Os parâmetros de instalação presentes no descritor de distribuição são armazenados como propriedades de distribuição no *container*. Dentre estes parâmetros, os parâmetros de QoS são de especial interesse. Uma aplicação pode alterar as propriedades para controle de QoS no nível de aplicação, por exemplo, qualidade do vídeo ou taxa de amostragem do áudio. A alteração destas propriedades do *container* afeta as conexões futuras de portas de fluxo contínuo para os componentes abrigados neste *container*. Para as conexões já estabelecidas, pode-se atualizar seus parâmetros de QoS para os novos parâmetros procedendo-se a parada e reinício do fluxo através dos métodos *stop* e *start* disponibilizados nas portas de fluxo contínuo.

[9] `http://java.sun.com/`

5. Exemplo de Aplicação

A aplicação de monitoramento e controle de QoS apresenta um monitor de QoS implementado através de código móvel. A idéia é desenvolver um agente móvel que monitora certos parâmetros de QoS *in loco* e, com base em um conjunto de regras simples de decisão, atua no sentido de aprimorar a qualidade do vídeo em função do desempenho da rede. O emprego de agentes móveis no monitoramento e controle de QoS em contraste com uma aplicação centralizada se justifica pelos pontos a seguir [Baschieri et al., 2002]:

- 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[10]. 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, bem como capacidades de adaptação e inferência são atrativas para problemas envolvendo gerência de recursos, monitoramento de sistemas e tomada de decisão.

O Monitor de QoS é a entidade de *software* responsável pelo monitoramento e controle de QoS para determinados fluxos estabelecidos pela aplicação (nesta aplicação, para os fluxos de vídeo). O monitor recebe como entrada um conjunto de fluxos de vídeo. Para cada fluxo, ele delega a tarefa de monitoramento e controle de QoS para uma instância do agente móvel (Agente de QoS). Este agente reporta-se periodicamente ao monitor que, com base na informação reportada pode notificar o usuário, renegociar a qualidade de serviço, ou ainda re-estabelecer o fluxo com novos parâmetros de QoS (por exemplo, tamanho da janela e taxa de quadros). O 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 *container* os parântros de QoS no nível de aplicação e os converte em recursos de rede (no caso, banda) necessários para o fluxo. Caso a rede ofereça reserva de recursos através de *Bandwidth Broker*, esta taxa foi negociada previamente através do interceptador *IQoS*. 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.

O Agente de QoS é um agente móvel que executa um ciclo de cinco fases:

1. monitoramento: o agente obtém os parâmetros de QoS através da leitura das propriedades de monitoramento definidas para as portas produtoras e apresentadoras de fluxo;
2. decisão: o agente decide se uma atuação no sentido de alterar a qualidade do fluxo se faz necessária;
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;

[10]Mais precisamente, o *overhead* está na serialização e de-serialização do estado do agente.

5. reportagem: o agente reporta para o monitor de QoS informações sobre a qualidade da conexão.

A fase de monitoramento é conduzida em todos os locais (*containers*) que 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). Nesta fase, 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 na porta apresentadora 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, estabelecemos 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 através da regra de controle $Q_{N+1} = Q_N + \delta$. O agente altera a propriedade *video_quality* do elemento de configuração do *container*, pára o fluxo e o reinicia a seguir através dos 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 forem 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 reporta 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 maior que 30% e *jitter* maior que 500 ms.

Implementação e Resultados

A figura 4 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 [Guimarães et al., 2003a, Guimarães et al., 2003b]. REAL disponibiliza equipamentos robóticos a partir de acesso remoto através da Internet.

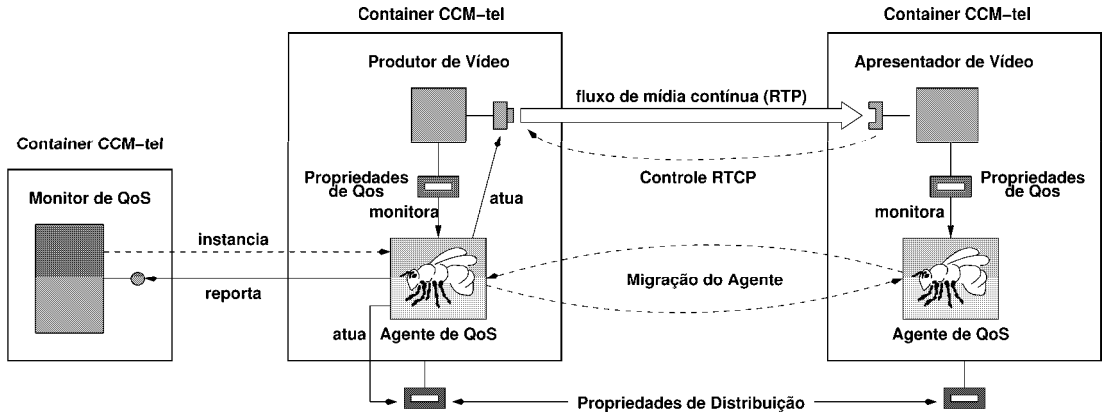

Figura 4: Monitoramento e controle de QoS através de agentes móveis.

A sessão de comunicação emprega dois fluxos de vídeo. Um é produzido por uma câmera panorâmica focada no ambiente de locomoção de um robô autônomo e outro por uma câmera embarcada no próprio robô. Cada fluxo é produzido por um componente dotado de porta transmissora de mídia e apresentado por outro componente dotado de porta apresentadora de mídia. Os componentes definem propriedades de monitoramento de QoS para as suas portas de fluxo contínuo.

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 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 *containers*). A figura mostra o ganho sendo adaptado, 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 6 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.

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 [Koliver and Farines, 2001]) 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.

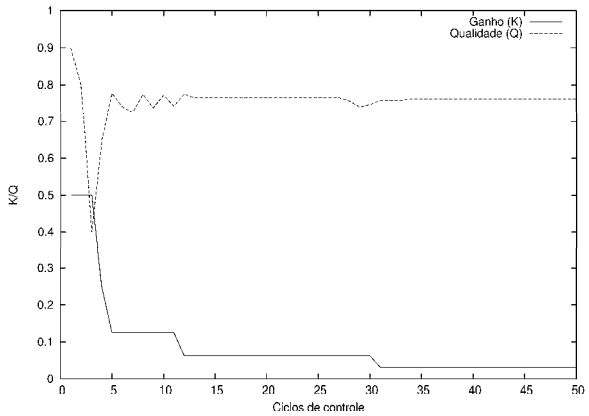

Figura 5: Variação do ganho e qualidade em função do número de ciclos.

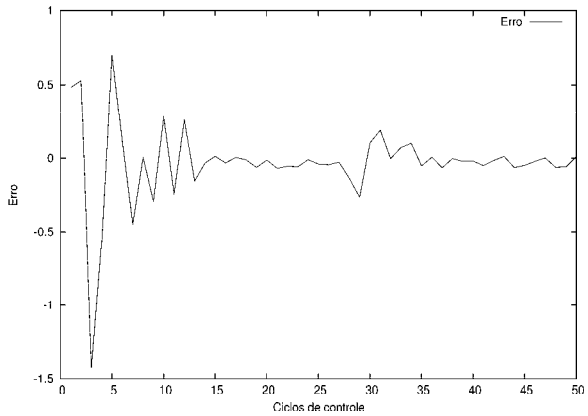

Figura 6: Variação do erro em função do número de ciclos.

6. Conclusões

Este artigo apresentou um modelo de componentes neutro (CM-tel) e uma plataforma de *middleware* (CCM-tel) que implementa um mapeamento deste modelo para a tecnologia Java/CORBA. Esta plataforma é capaz de suprir um importante conjunto de requisitos impostos pelas aplicações emergentes, notadamente, comunicação multimídia com qualidade de serviço e adaptação dinâmica ao ambiente via mobilidade de código. Por tais características, acreditamos que esta plataforma é um avanço face as plataformas existentes.

A plataforma CCM-tel integra dois paradigmas de desenvolvimento de *software* distribuído: componentes e agentes móveis. O suporte integrado a componentes e código móvel propicia grande flexibilidade no desenvolvimento de aplicações. Agentes móveis podem desempenhar funções de gerência, adaptação e "customização", enquanto componentes podem desempenhar funções de comunicação multimídia e notificação assíncrona de eventos. Neste artigo, agentes móveis foram empregados para monitoramento e controle de qualidade de serviço em aplicações que empregam comunicação multimídia distribuída. Mostrou-se que, para o ambiente de teste empregado, através de regras simples

de controle pode-se obter uma sessão de comunicação estável mesmo em redes que operam em regime de melhor-esforço. Futuramente, pretendemos avaliar o comportamento deste mecanismo de monitoramento e controle de QoS na Internet pública.

A plataforma CCM-tel está sendo aprimorada com incorporação de funções de segurança. Adicionalmente, um mapeamento do modelo CM-tel para a tecnologia J2ME (Java2 Micro Edition) está em desenvolvimento com o objetivo de utilizar este modelo em dispositivos móveis sem fio.

Agradecimentos

Esta pesquisa foi suportada em parte pelo Projeto Quaresma com recursos do CNPq - Programa de Redes Avançadas e desenvolvida no âmbito do convênio de cooperação CenPRA-FEEC/Unicamp.

References

Baschieri, F., Bellavista, P., and Corradi, A. (2002). 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.

Carlson, D. (2001). *"Modeling XML Applications with UML"*. Addison-Weslye.

Guimarães, E., Maffeis, A., Pereira, J., Russo, B., Bergerman, M., Cardozo, E., and Magalhães, M. (2003a). "REAL: A Virtual Laboratory for Mobile Robot Experiments". *IEEE Transactions on Education*, 46(1):37–42.

Guimarães, E., Maffeis, A., Pinto, R., Miglinski, C., Cardozo, E., Bergerman, M., and Magalhães, M. (2003b). "REAL - A Virtual Laboratory Built From Software Components". *Proceedings of the IEEE*, 91(3):440–448.

Heineman, G. and Council, W. (2001). *Component-Based Software Engineering - Putting the Pieces Together*. Addison-Wesley.

ISO-IEC (1995). Open distributed processing reference model, part 1: Overview. ISO/IEC 10746-1, International Standard Organization.

Koliver, C. and Farines, J. (2001). "Um controlador Nebuloso para Adaptação de QoS". In *19 Simpósio Brasileiro de Redes de Computadores*, Florianópolis, SC. SBC.

Pinto, R., Guimarães, E., Cardozo, E., and Magalhães, M. (2003). "Incorporação de Qualidade de Serviços em Aplicações Telemáticas". In *21 Simpósio Brasileiro de Redes de Computadores*, Natal, RN. SBC.

Tripathi, A. (2002). Challenges designing next-generation middleware systems. *Communications of The ACM*, 45(6).