Wander Euclides Carneiro Pimentel Gomes

Uma Plataforma de Desenvolvimento de Software Baseado em Componentes para Dispositivos Móveis

Faculdade de Engenharia Elétrica e de Computação

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Eleri Cardozo

Campinas, SP
2005

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

Gomes, Wander Euclides Carneiro Pimentel Gomes
G585p Uma plataforma de desenvolvimento de software baseado em componentes para dispositivos móveis / Wander Euclides Carneiro Pimentel Gomes. – Campinas, SP: [s.n.], 2005.

Orientador: Eleri Cardozo.
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas de comunicação sem fio. 2. Java (linguagem de programação de computador). 3. XML (linguagem de marcação de documentos). 4. Programação orientada a objetos. 5. Programação (Computadores). 6. Sistemas multimídia. 7. Redes de computação. 8. Processamento eletrônico de dados - Processamento Distribuído. I. Cardozo, Eleri. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: A software development platform based on components for mobile devices.
Palavras-chave em Inglês: Wireless communication systems, JAVA (Computer program language) XML (Extensible Markup Language (document markup language)), Object-oriented programming (Computer science), Computer programming, Multimedia Systems, Computer networks e Distributed computer systems in eletronic data processing.
Área de Concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Edmundo Roberto Mauro Madeira, Maurício Ferreira Magalhães e Marco Aurélio Amaral Henriques.
Data da defesa: 22/02/2005

Wander Euclides Carneiro Pimentel Gomes

Uma Plataforma de Desenvolvimento de Software Baseado em Componentes para Dispositivos Móveis

Faculdade de Engenharia Elétrica e de Computação

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 22/02/2005

Prof. Dr. Eleri Cardozo - UNICAMP
Prof. Dr. Edmundo Roberto Mauro Madeira - IC/UNICAMP
Prof. Dr. Maurício Ferreira Magalhães - UNICAMP
Prof. Dr. Marco Aurélio Amaral Henriques - UNICAMP

Campinas, SP
2005

Resumo

Este trabalho apresenta uma infra-estrutura para auxílio ao desenvolvimento de *software* baseado em componentes para dispositivos móveis com baixo poder de processamento e armazenamento. Esta infra-estrutura baseia-se em um modelo de componentes neutro em termos de tecnologia e especificado inteiramente em UML *(Unified Modeling Language)*. A plataforma utiliza *Web Services* para comunicação síncrona entre os componentes. Um Serviço de Notificação baseado em documentos XML *(Extensible Markup Language)* foi desenvolvido para suporte a notificação assíncrona entre componentes. Um exemplo de aplicação na área de gerência de redes ilustra as funcionalidades da infra-estrutura.

Palavras-chave: Dispositivos Móveis, Componentes de *Software*, Serviços *Web*, Plataformas de *Middleware*.

Abstract

This work presents an infrastructure for supporting component-based software development for mobile devices with limited resources regarding processing power and storage. This infrastructure is based on a neutral component model in terms of technology and specified entirely in UML (Unified Modeling Language). The platform employs Web Services for synchronous communication between components. A Notification Service based on XML (Extensible Markup Language) documents was developed in order to support asynchronous communication among the components. An example of application in the field of network management illustrates the functionalities of the infrastructure.

Keywords: Mobile Devices, Software Components, Web Services, Middleware Platforms.

Dedico esse trabalho à minha querida esposa Lu.
Sem o seu amor, carinho e paciência, nada disso seria possível.

Agradecimentos

Ao meu orientador, Prof. Eleri Cardozo, agradeço a grande atenção e dedicação.

A minha família, pelo carinho e apoio durante essa caminhada.

Aos colegas de pós-graduação, pelo companheirismo e ajuda.

Ao CNPq, pelo apoio financeiro.

Sumário

Lista de Figuras

Glossário

BREW	Binary Runtime Environment for Wireless
CBD	Component-based Development
CCM	CORBA Component Model
CCM-tel	CORBA CM-tel
CIDL	Component Implementation Definition Language
CIF	Component Implementation Framework
CLDC	Connected Limited Device Configuration
CM-tel	Component Model for telematic applications
CORBA	Common Object Request Broker Architecture
COTS	Components off the shelf
DCOM	Distributed Component Object Model
EJB	Enterprise Java Beans
HTML	HyperText Markup Language
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2 Platform Micro Edition
J2SE	Java 2 Platform Standard Edition
JCP	Java Community Process

JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JPEG	Joint Photographic Experts Group
kSOAP	kilobyte SOAP
MAF	Mobile Agents Facilities
MECM-tel	Micro Edition Component Model for telematic applications
MIDP	Mobile Information Device Profile
OMG	Object Management Group
ORB	Object Request Broker
OTA	Over the Air
PDA	Personal Digital Assistant
POA	Portable Object Adapter
QoS	Quality of Service
RAM	Random Access Memory
RM-ODP	Reference Model - Open Distributed Processing
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus

W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WSP	Web Services for Palm
WSTKMD	Web Services Toolkit for Mobile Devices
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation

Capítulo 1

Introdução

Este capítulo apresenta algumas motivações para o desenvolvimento de *software* para dispositivos móveis utilizando uma abordagem orientada a componentes, o que traz, entre outros benefícios, um maior reuso de código e a diminuição da complexidade de desenvolvimento e integração de aplicações distribuídas. Posteriormente, as contribuições são descritas e alguns trabalhos relacionados são discutidos.

1.1 Motivações

Os dispositivos móveis, principalmente aqueles com baixo poder de processamento, são utilizados por um número cada vez maior de usuários para os mais diversos fins. Isso acontece, em grande parte, devido à contínua redução do preço final desses aparelhos bem como à diminuição do tamanho, graças a redução das dimensões e do consumo de potência dos componentes eletrônicos utilizados.

O benefício da mobilidade, principalmente no caso dos telefones celulares, torna o dispositivo uma importante ferramenta de trabalho. Adicionalmente, as operadoras de telefonia móvel oferecem planos de pagamento cada vez mais atrativos e flexíveis para os mais diversos perfis de usuários. Outro aspecto a ser considerado é a concorrência entre empresas operadoras e entre fabricantes de aparelhos que torna os preços desses dispositivos e do serviço cada vez mais acessíveis.

O aumento de receita das operadoras, como em qualquer empresa com fins lucrativos, é uma necessidade real. Esse aumento é trazido pelo crescimento do tráfego na rede, o que é largamente estimulado pelas operadoras. É nesse ponto que entra o investimento em aplicações para dispositivos móveis, o que se traduz em um amplo potencial de geração de tráfego. Esse investimento é feito tanto pelas operadoras, interessadas no aumento de suas receitas, quanto pelos fabricantes de aparelhos, interessados na venda de dispositivos cada vez mais modernos que hospedem aplicações cada vez mais diversificadas.

Com todos esses estímulos e preços acessíveis, dispositivos com poder de processamento cada vez maior e menor custo são atualmente utilizados por milhões de usuários. Ao contrário dos primeiros modelos, os atuais podem manipular conteúdo multimídia, tirar fotografias e executar aplicativos.

Porém, justamente devido a essa proliferação, existem dezenas de modelos de aparelhos de diversos fabricantes utilizando diferentes tipos de processadores e sistemas operacionais. Isso torna necessária a existência de um denominador comum para o desenvolvimento de sistemas que executem em tais plataformas, pois caso contrário, a ausência de padrões mundialmente aceitos pode dificultar a evolução dessa tecnologia e sua adoção como forma de comunicação e entretenimento.

A Plataforma Java da *Sun Microsystems Inc* (1), através do J2ME *(Java 2 Platform Micro Edition)* (2), propõe justamente a portabilidade das aplicações e vem se tornado um padrão *de facto* no desenvolvimento de aplicações para dispositivos móveis. Essa plataforma é adotada e patrocinada por gigantes como IBM, Motorola, Palm Inc., Nokia, entre outros. Opções alternativas, porém na mesma direção, são o Qualcomm BREW (3) e o SuperWaba (4).

O poder de processamento crescente desses aparelhos viabiliza o desenvolvimento de *software* orientado a componentes juntamente com a distribuição desses elementos de *software* explorando a conectividade dos dispositivos. Isso torna possível que PDAs e telefones celulares sejam responsáveis por uma parcela maior do processamento da aplicação e, algumas vezes, tornando-se nós da rede como qualquer outro computador.

Porém, ainda existem sérias limitações quanto ao poder de processamento, quanto à capacidade de armazenamento e principalmente quanto à usabilidade, no que diz respeito ao tamanho da tela e à entrada de dados. Esses requisitos não-funcionais devem ser considerados tanto para o desenvolvimento de aplicações quanto na escolha da plataforma a ser utilizada.

1.2 Contribuições do Trabalho Proposto

Dentro do contexto descrito acima, o propósito do trabalho é apresentar uma plataforma de *middleware* para desenvolvimento de aplicações telemáticas em dispositivos móveis baseada no modelo de componentes CM-tel *(**C**omponent **M**odel for **t**elematic applications)* (5) (6). A plataforma auxilia a utilização de componentes aderentes ao modelo CM-tel em aplicações para dispositivos móveis. Esse modelo pode ser inteiramente descrito em UML *(Unified Modeling Language)*, sendo genérico e neutro em termos de tecnologia (linguagens de programação, sistemas operacionais, protocolos de rede, etc.).

Devido a essa característica do modelo de componentes em questão, é proposto um mapeamento para J2ME/*Web Services* o que permite explorar as vantagens da mobilidade e do baixo custo dos dispositivos móveis, se comparado a microcomputadores de mesa. Ou seja, é proposta uma infra-

estrutura para desenvolvimento de *software* orientado a componentes que transforma aparelhos, como telefones celulares e computadores de mão, em elementos principais das aplicações. Isso torna possível que os dispositivos assumam um papel de destaque na aplicação, ao contrário de meros clientes de um determinado servidor com maior poder de processamento.

Um mapeamento do modelo CM-tel para CORBA/Java, implementado na plataforma CCM-tel (CORBA CM-tel) (7), já validou a neutralidade em termos de tecnologia do modelo. Adicionalmente, um mapeamento para outra linguagem, no caso J2ME/*Web Services*, agrega valor à proposta mais ampla do modelo de componentes CM-tel, reafirmando sua independência de tecnologia e possibilitando o desenvolvedor especificar e implementar componentes CM-tel que sejam executados em dispositivos móveis. Dessa forma, é possível que aplicações mais complexas, tais como aplicações em gerência de rede ou laboratórios virtuais (8), possam fazer uso de componentes de *software* CM-tel sendo executados em dispositivos móveis. Essa possibilidade torna o sistema mais flexível, além de trazer os benefícios da mobilidade e do baixo custo. Como contribuições do trabalho proposto, destacam-se os seguintes pontos:

- transformação das especificações de componentes e *containers* CM-tel do formato XMI *(XML Metadata Interchange)* (9) para o formato XML. Diagramas de classes UML representando a especificação desses elementos do modelo são exportados para o formato XMI por meio de uma ferramenta de projeto como o ArgoUML (10);
- implementação da plataforma MECM-tel que constitui o mapeamento do modelo CM-tel para a tecnologia J2ME/ *Web Services*. Essa plataforma fornece a ferramenta MECM-tel Builder responsável pela geração automática de código com auxílio de uma interface gráfica e aplicação de transformações XSLT (*Extensible Stylesheet Language Transformations*);
- implementação de um Serviço de Notificação baseado em documentos XML para dar suporte à eventos gerados e consumidos pelos componentes;
- implementação de uma aplicação na área de gerência de redes para ilustrar as funcionalidades da plataforma MECM-tel;
- utilização de diagramas UML de colaboração para especificar a montagem da aplicação e diagramas UML de distribuição indicando os nós da rede onde serão instalados os componentes.

A plataforma CCM-tel possui como entrada documentos XML contendo as especificações dos componentes e *conteiners*, enquanto que a plataforma MECM-tel recebe como entrada diagramas UML no formato XMI. Isso torna possível que a especificação seja feita em um nível mais alto de abstração facilitando o seu entendimento.

1.3 Trabalhos Correlatos

O desenvolvimento de infra-estruturas para suporte a aplicações em dispositivos móveis é uma tendência. O aumento do poder de processamento desses aparelhos fornece aos programadores e arquitetos de *software* um leque de opções de projeto que antes não eram suportadas, como por exemplo aplicações baseadas em *Web Services* e em componentes de *software*.

No contexto do presente trabalho, o de aplicações baseadas em componentes de *software* distribuídos e *Web Services*, pode-se citar alguns trabalhos correlatos. Primeiramente, o projeto *iChilli* (11) constitui uma plataforma J2EE *(Java 2 Platform, Enterprise Edition)* (12) móvel que proporciona um ambiente compatível com EJB *(Enterprise Java Beans)* (13) para dispositivos móveis. A plataforma possui o objetivo de transferir tecnologias Java existentes e bem aceitas, tais como *Enterprise Java Beans*, para o mundo dos dispositivos móveis.

Essa plataforma J2EE móvel cobre o processo completo de desenvolvimento de aplicações Java *Enterprise*, o processo de *deployment* da aplicação em dispositivos móveis e finalmente monitora e gerencia essas aplicações. A proposta do *iChilli* difere dos demais sistemas J2EE pelo fato de proporcionar um ambiente distribuído para execução de *Enterprise Java Beans* em dispositivos móveis.

O *iChilli* também utiliza projetos de código aberto existentes como o Jakarta Avalon (14) para prover a infra-estrutura J2EE servidora e o projeto *OpenEJB* (15) para prover o ambiente de execução *(container)* EJB tanto no cliente como no servidor.

A motivação para citar essa plataforma como trabalho correlato está no fato da mesma proporcionar um ambiente de execução de componentes de *software* em dispositivos móveis possibilitando um maior reuso de código.

Na linha de *Web Services*, S. Berger (16) propõe que os dispositivos móveis podem não só consumir *Web Services* mas também prover tais serviços. Isso pode ser um poderoso modelo para facilitar a interação automática entre dispositivos de baixo poder de processamento. O trabalho também considera as dificuldades relativas à mobilidade dos dispositivos que hospedam *Web Services*, tais como descoberta de serviços, indisponibilidade de acesso e requisitos de segurança. Também são explorados os benefícios da aplicação da tecnologia de *Web Services* para facilitar a interação de programas entre dispositivos móveis e entre dispositivos móveis e a infra-estrutura.

O ponto em comum entre esse trabalho e o proposto nesta dissertação é justamente a interação entre componentes de *software* instalados em dispositivos móveis através de *Web Services*.

Outro trabalho correlato a ser citado é o SANKHYA Varadhi (17) que constitui uma solução de *middleware* e comunicação usada para integrar e desenvolver aplicações orientadas a objeto. Varadhi viabiliza a distribuição de *software* entre plataformas tais como Windows, Linux, Solaris e sistemas embarcados como telefones celulares e PDAs podendo as aplicações ser facilmente chaveadas para diferentes padrões, como CORBA e SOAP. SANKHYA Varadhi versão 1.1 implementa a especifi-

cação CORBA 2.2 (*Minimum* CORBA). O trabalho proposto também apresenta uma solução de middleware que auxilia o desenvolvimento de aplicações orientadas a objetos e que podem ser executadas em dispositivos móveis.

1.4 Organização do Texto

O capítulo 2 fornece uma visão geral do paradigma de componentes de *software* onde são discutidos os modelos de componentes EJB, CORBA *Component* e CM-tel. Adicionalmente, será abordado o desenvolvimento de software orientado a componentes.

O capítulo 3 trata do mapeamento do modelo CM-tel para tecnologias alvo tal como o CORBA. Os requisitos de um mapeamento para *Web Services* também são abordados nesse capítulo. Além disso, a análise e projeto de um mapeamento bem como a geração automática de código são discutidos.

O capítulo 4 aborda a plataforma MECM-tel *(Micro Edition CM-tel)* que constitui o mapeamento do modelo CM-tel para tecnologia J2ME/*Web Services*. Essa abordagem descreve o projeto da plataforma, sua implementação e um breve estudo das tecnologias envolvidas.

O capítulo 5 contém a descrição detalhada de um exemplo de aplicação na área de gerência de redes que utiliza componentes MECM-tel em dispositivos móveis. O cenário da aplicação e sua arquitetura são mostrados nesse capítulo juntamente com uma avaliação quanto à porcentagem aproximada de código gerado automaticamente e quanto ao desempenho e robustez da aplicação.

Finalmente, o capítulo 6 apresenta as conclusões do trabalho realizado na forma de uma breve retrospectiva do que foi feito, da análise dos resultados alcançados e de possíveis trabalhos futuros a serem desenvolvidos. Esses trabalhos podem ser tanto contribuições ao próprio modelo CM-tel quanto possíveis mapeamentos deste modelo para outras tecnologias.

Capítulo 2

Componentes de Software

Este capítulo apresenta uma visão geral do paradigma de componentes de *software* juntamente com uma breve descrição dos modelos de componentes *Enterprise JavaBeans* (EJB), *CORBA Component Model* (CCM) (18) e CM-tel. Além disso, será abordado o desenvolvimento de *software* baseado em componentes (CBD) e as vantagens da utilização dessa técnica. A plataforma CCM-tel (7), que constitui um exemplo de mapeamento do modelo de componentes CM-tel para tecnologia CORBA, também será descrita nesse capítulo.

2.1 Visão Geral de Componentes de Software

Sistemas baseados em componentes são aderentes ao princípio da divisão e conquista para gerenciamento da complexidade. Este princípio consiste em dividir o problema em várias partes menores, resolvê-las separadamente e então construir soluções a partir desses blocos mais simples. Esse conceito é largamente utilizado em engenharia de *software*, como por exemplo no desenvolvimento em camadas *(tiers)* e na própria orientação a objetos.

O conceito, em si, de componentes como blocos de construção para desenvolvimento de sistemas não pode ser considerado uma novidade. Porém, a evolução do poder computacional, da capacidade de armazenamento bem como das demais tecnologias envolvidas no processo de desenvolvimento de *software* permitiram a real utilização desse conceito na construção de sistemas complexos.

Como principais vantagens da utilização desse paradigma de componentes podem ser citados o aumento da produtividade no desenvolvimento, a facilidade de manutenção e o aumento da qualidade, trazendo, com isso, uma diminuição do custo. Esses benefícios são atingidos principalmente através do reuso de código, interligação explícita entre os componentes (baixo acoplamento), alta coesão e geração automática de código.

2.1.1 Definições de Componentes de Software

Existem inúmeras definições de componentes de *software* por parte de vários autores e, às vezes, até alguns abusos no uso do termo. Neste trabalho, será adotada a definição de Heineman e Council (19).

> "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"

Alguns termos dessa definição são esclarecidos a seguir:

- um modelo de componentes define padrões de interação e de composição para componentes;
- um padrão de interação define os requisitos que permitem elementos de *software* interagirem, direta ou indiretamente, com outros elementos de *software*;
- um padrão de composição define os requisitos que permitem a composição de elementos de *software* visando a criação de elementos maiores.

Outras duas definições de componentes de *software* são mencionadas. Segundo Szyperski (20), um componente é uma unidade de composição com interfaces especificadas contratualmente e dependências de contexto explícitas que pode ser instalado independentemente e exposto à composição por terceiros.

Adicionalmente, Cheesman e Daniels (21) adotam uma forma diferente de definição em que um componente assume várias formas refletindo algum aspecto durante o ciclo de desenvolvimento. Dessa forma, eles não definem o componente, e sim as várias formas que este pode tomar. Segundo essas definições, um componente possui:

- uma especificação de componente que descreve seu comportamento principalmente através da definição das interfaces;
- uma implementação do componente que consiste na realização da especificação do componente em uma determinada linguagem de programação;
- um padrão de componentes ao qual um componente deve obedecer para tornar possível a composição desses elementos de *software* para construção de aplicações;
- um empacotamento que contém os arquivos da implementação do componente e suas dependências;
- uma instalação que é realizada em determinado processador quando o arquivo que o empacota é transferido e processado neste processador.

2.2 Desenvolvimento Baseado em Componentes

Desenvolvimento baseado em componentes (*Component-based Development - CBD*) é uma importante abordagem de desenvolvimento de sistemas de *software* que tomou força nos últimos anos devido aos avanços tecnológicos principalmente de infra-estrutura, tais como o aumento do poder de processamento das máquinas.

CBD possui o foco no desenvolvimento de sistemas de larga escala integrando componentes pré-existentes. Através das melhorias em flexibilidade e manutenção dos sistemas, essa abordagem pode ser usada potencialmente para reduzir os custos do desenvolvimento de *software*, construir sistemas rapidamente e reduzir a espiral de manutenção associada ao suporte e à atualização de sistemas complexos. Como fundamentação para essa abordagem está o fato de que certas partes de um sistema de *software* reaparecem com suficiente regularidade de maneira que as partes comuns deveriam ser escritas uma só vez e reutilizadas.

CBD muda a ênfase do desenvolvimento da programação para a composição de sistemas de *software*. Desenvolver sistemas baseados em componentes está se tornando viável devido a alguns aspectos, tais como: aumento da variedade e qualidade de produtos de prateleira (*Components off the shelf - COTS*); pressões para redução de custos em desenvolvimento e em manutenção; surgimento de tecnologias de integração de componentes, como os ORBs; crescimento da quantidade de *softwares* nas empresas que podem ser reutilizados em novos sistemas.

Além de possibilitar reuso de código em maior granularidade, o que ocorre de forma insatisfatória no paradigma de objetos, podemos apontar como a principal vantagem de CBD a melhoria no gerenciamento de mudanças. Outras vantagens dessa abordagem de desenvolvimento são: diminuição do *"time-to-market"*, baixo acoplamento entre componentes e maior facilidade na alocação das equipes de desenvolvimento. Como desvantagens da adoção da abordagem CBD, podemos citar as dificuldades de se entender o que é realmente um componente, quais tecnologias estão envolvidas e como desenvolver componentes. A própria mudança de cultura na empresa quanto ao desenvolvimento de *software* se torna um empecilho a adoção de CBD.

Devido ao fato da real utilização de CBD ser relativamente atual, existem vários aspectos que ainda não se encontram suficientemente maduros. Como efetivamente armazenar e encontrar componentes adequados, como facilitar o entendimento de componentes e como garantir a qualidade dos componentes são tópicos de pesquisa bastante férteis. Talvez isso não seja considerado uma desvantagem, mas limita o desenvolvimento de software baseado em componentes.

Segundo Cheesman e Daniels (21), a principal vantagem da utilização de componentes é a facilidade de substituição e não o reuso de código, como muitos afirmam. Tal substituição pode ser feita tanto por uma implementação das funções do componente completamente diferente quanto por uma versão atualizada da mesma implementação. Isso coloca a ênfase na arquitetura, facilitando o

gerenciamento do sistema como um todo, ou seja, dos componentes envolvidos e suas mudanças de requisitos.

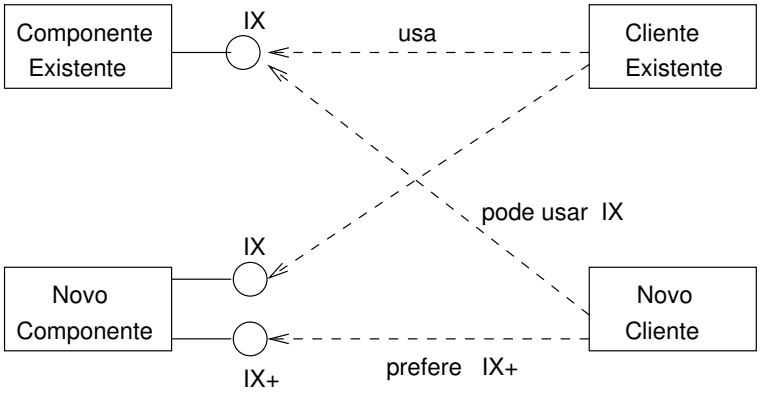

Fig. 2.1: Dependência de interfaces e substituição de componentes

Desenvolvimento baseado em componentes é diferente das abordagens anteriores quanto à separação entre a especificação do componente e sua implementação e quanto à divisão das especificações do componente em interfaces.

Essa divisão significa que as dependências entre componentes podem ser restringidas a interfaces individuais e não envolver toda a especificação dos componentes. Isso reduz o impacto das mudanças, pois um componente pode ser substituído por outro mesmo que este possua uma especificação diferente bastando que essa especificação contenha as mesmas interfaces requeridas pelo componente cliente, como mostra a figura 2.1. Tais características permitem que um componente seja atualizado ou substituído com mínimo impacto nos clientes.

2.3 O Modelo de Componentes Enterprise JavaBeans

A tecnologia *Enterprise JavaBeans (EJB)* (13) é uma arquitetura de componentes do lado servidor para a plataforma *Java 2 Platform Enterprise Edition (J2EE)* (12). EJB possibilita o desenvolvimento rápido e simplificado de aplicações distribuídas, transacionais, seguras e portáveis baseadas na tecnologia Java.

O modelo de componentes EJB é um modelo com especificação aberta fornecido pela *Sun Microsystems* para o desenvolvimento e instalação de aplicações Java pertencentes ao domínio de negócios.

Os *enterprise beans*, componentes J2EE que implementam a tecnologia EJB, são executados no *container* EJB que consiste em um ambiente de tempo de execução dentro do servidor J2EE, conforme mostrado na figura 2.2. Esses *containers* proporcionam, de forma transparente ao desenvolve-

dor de aplicações, serviços tais como suporte à transações e persistência. Um *enterprise bean* é um componente do lado servidor que encapsula a lógica de negócios de uma aplicação, ou seja, o código que satisfaz o objetivo da aplicação.

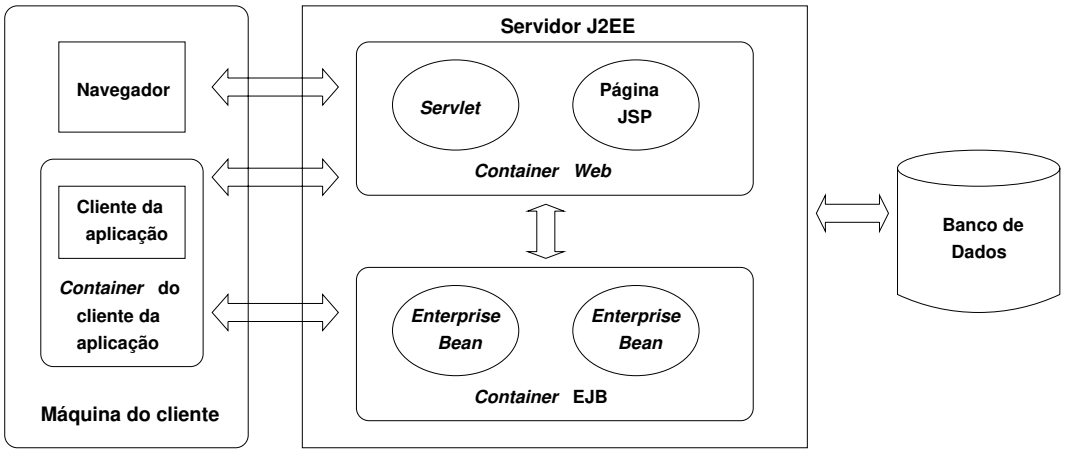

Fig. 2.2: Servidor J2EE e *containers*

A implementação do modelo EJB é dividida em duas partes: o *container* e o servidor EJB. A primeira gerencia o ciclo de vida do componente e constitui a parte central do modelo de componentes. A segunda, juntamente com o *container*, fornece todos os recursos necessários para que um componente distribuído possa interagir com outros componentes e provê o ambiente de execução para um ou mais *containers*.

2.3.1 O Container EJB

Os *enterprise beans* são executados em um ambiente especial chamado *container* EJB que hospeda e gerencia um *entreprise bean* da mesma forma que o *container Web* hospeda um *servlet* ou um *browser* HTML hospeda um *applet*. Um *enterprise bean* não pode ser executado fora de um *container* EJB.

O *container* EJB é responsável por fornecer serviços como transação, gerenciamento de recursos, escalabilidade, mobilidade, persistência e segurança para componentes EJB. Um *container* provê o encapsulamento de um ou mais componentes de uma aplicação e isola o *enterprise bean* do acesso direto por parte das aplicações cliente. Quando uma aplicação cliente invoca um método remoto de um componente EJB, o *container* primeiramente intercepta a invocação para garantir que persistência, transação e segurança sejam aplicadas devidamente a todas as operações invocadas pelo cliente do *enterprise bean*. A figura 2.3 mostra o gerenciamento de um *bean* em tempo de execução.

Um *enterprise bean* depende do *container* para os recursos que ele necessita, como por exemplo: acesso à conexão JDBC (*Java Database Connectivity*) ou a outro componente; obtenção de sua

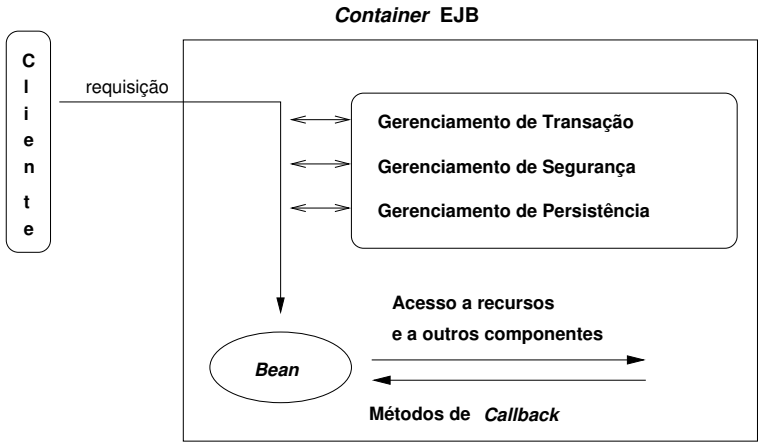

Fig. 2.3: Gerenciamento de um *bean* em tempo de execução

própria referência, identidade de um cliente ou acesso a propriedades. O *enterprise bean* interage com o seu *container* através de um dos três mecanismos:

- Métodos de *Callback*: cada *enterprise bean* implementa um sub-tipo da interface *Enterprise-Bean* que define vários métodos chamados de métodos de *callback*. O *container* invoca esses métodos para notificar o *bean* sobre eventos do seu ciclo de vida: sua ativação, persistência de estado, fim de transação, remoção do *bean* da memória, etc. Dessa forma, os métodos de callback permitem que o *bean* execute algumas tarefas imediatamente antes ou depois de algum desses eventos.
- *EJBContext*: cada *enterprise bean* obtém um objeto *EJBContext* que é uma referência direta para o *container*. A interface *EJBContext* provê métodos para interação com o *container* de forma que o *bean* pode requisitar informações a respeito de seu ambiente como por exemplo a identidade de um cliente, o status de uma transação ou obter referências remotas dele mesmo.
- *Java Naming and Directory Interface (JNDI)*: cada *enterprise bean* tem acesso automático a um serviço de nomes especial chamado *Environment Naming Context (ENC)* que por sua vez é gerenciado pelo *container* e acessado por *beans* usando JNDI. JNDI ENC permite que o *bean* utilize recursos como conexões JDBC, outros *enterprise beans* e propriedades específicas para aquele *bean*.

A interação entre o cliente e o componente EJB se dá através de duas interfaces geradas pelo *container*: interface *Home* e interface *Remote*, como mostrado na figura 2.4. As implementações dessas interfaces são chamadas de *EJBHome* e *EJBObject* respectivamente. No momento em que o cliente faz uma invocação a uma das operações dessas interfaces, o *container* a intercepta, personalizando-a ou adicionando serviços de gerência antes de repassá-la para o componente destino.

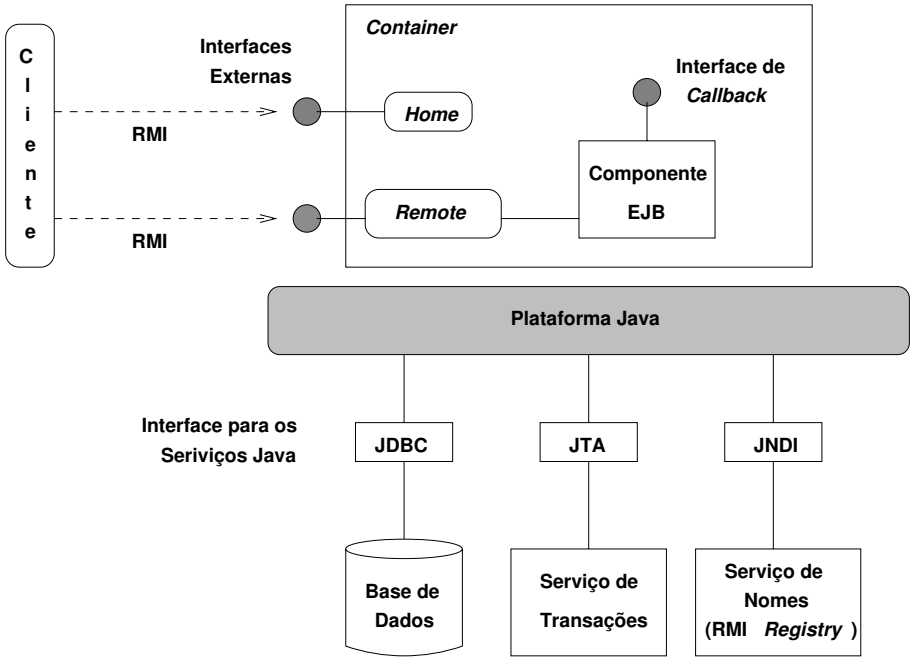

Fig. 2.4: O container no ambiente EJB

Através do *EJBHome*, clientes podem criar, encontrar e remover componentes EJB, enquanto o *EJBObject* provê o acesso às operações da lógica da aplicação do componente EJB.

O padrão *Open Distributed Processing Reference Model* (RM-ODP) (22) define três interfaces de componentes:

- **interfaces síncronas**: permitem interações síncronas entre dois componentes através de chamada de métodos remotos;
- **interfaces de notificação**: permitem interações assíncronas entre componentes através do envio de mensagens de notificação;
- **interfaces de fluxo contínuo**: permitem manipulação de mídias contínuas (fluxo de áudio e vídeo).

O modelo EJB não fornece as interfaces de sinal (notificação) e de *stream* (fluxo contínuo) propostas pelo RM-ODP, sendo todo o mecanismo de interação baseado em chamada de métodos remotos (estilo interação síncrona proposto pelo RM-ODP).

2.3.2 Tipos de Enterprise Beans

Os três tipos diferentes de *enterprise bean* são resumidos a seguir:

- Sessão *(Session)*: Executa uma tarefa para um cliente;

- Entidade *(Entity)*: Representa um objeto de entidade de negócios que existe em armazenamento persistente;
- Orientado a mensagem *(Message-Driven)*: Atua como um *listener* para a API *Java Message Service* (JMS) processando mensagens assincronamente.

Bean de Sessão

Um *bean* de sessão funciona como um representante de um único cliente no servidor J2EE. O cliente invoca as operações do *bean* de sessão para acessar uma aplicação implementada no servidor. Ou seja, o *bean* esconde de seu cliente a complexidade da lógica de negócios.

Existem dois tipos de *bean* de sessão: com informações de estado *(stateful)* e sem informações de estado *(stateless)*. Em um *bean* de sessão com informações de estado, as variáveis de instância representam o estado de uma única sessão cliente-*bean*. Uma vez que o cliente "conversa" com o *bean*, seu estado é freqüentemente denominado estado conversacional. Esse estado é mantido enquanto durar a sessão cliente-*bean*.

Quando um cliente invoca o método de um *bean* sem informação de estado, as variáveis de instância do *bean* podem conter um estado, mas apenas enquanto durar a chamada. Exceto durante a chamada do método, todas as instâncias de um *bean* sem informação de estado são equivalentes permitindo que o *container* atribua uma instância a qualquer cliente. Dessa forma, esses *beans* podem oferecer escalabilidade para aplicações que requerem um grande número de clientes.

Bean de Entidade

Um bean de entidade *(entity bean)* representa um objeto de negócios em um mecanismo de armazenamento persistente que, no J2EE SDK (*Software Development Kit*), corresponde a um banco de dados relacional. Cada *bean* de entidade possui uma tabela subjacente e cada instância do *bean* corresponde a uma tupla dessa tabela. Como exemplo de objetos de negócios, podem ser citados clientes, pedidos e produtos.

A persistência dos *beans* de entidade pode ser gerenciada pelo *container (container-managed persistence)* ou pelo próprio *bean* (*bean-managed persistence*). O *container* EJB manipula todos os acessos a banco de dados requeridos pelo *bean* de entidade acarretando uma maior portabilidade dos *beans*. Isso acontece devido ao fato do código do *bean* não estar vinculado a nenhum mecanismo de armazenamento persistente (banco de dados). Os *beans* de entidade e os *beans* de sessão diferem em vários aspectos. Os *beans* de entidade são persistentes, permitem acesso compartilhado, têm chaves primárias e podem participar de relacionamentos com outros *beans* de entidade.

Bean Orientado a Mensagem

Um *bean* orientado a mensagem *(message-driven bean)* permite aplicações J2EE processar mensagens assincronamente atuando como um *listener* de mensagens JMS. As mensagens tanto podem ser enviadas por qualquer componente J2EE (um cliente da aplicação, outro *bean* ou componente Web) quanto por uma aplicação JMS.

A principal diferença entre *beans* orientados a mensagens e os *beans* de sessão e de entidade é quanto ao acesso por parte dos clientes que não pode ser através de interfaces. Em diversos aspectos, um *bean* orientado a mensagens se parece com um *bean* de sessão sem informações de estado:

- instâncias de um *bean* orientado a mensagens não mantêm nenhum dado ou estado conversacional para um cliente específico;
- todas as instâncias de um *bean* orientado a mensagens são equivalentes, permitindo que um container EJB atribua uma mensagem a qualquer instância do *bean*;
- um único *bean* orientado a mensagens pode processar mensagens para múltiplos clientes.

2.3.3 Benefícios do Enterprise JavaBeans

Os *enterprise beans* simplificam o desenvolvimento de aplicações distribuídas de grande porte. Primeiramente, uma vez que o *container* EJB assume uma série de requisitos não-funcionais, tais como gerenciamento de transações e segurança, o desenvolvedor do *bean* pode se concentrar na solução de problemas de negócios. Posteriormente, o desenvolvedor do cliente pode se concentrar na apresentação, não tendo que codificar as rotinas que implementam o suporte para as regras de negócio tais como transação e persistência. Finalmente, como os *enterprise beans* são componentes portáveis, o arquiteto de *software* pode construir novas aplicações a partir de *beans* existentes sendo executadas em qualquer servidor J2EE compatível.

2.4 O Modelo de Componentes CORBA

CORBA Component Model (CCM) é um modelo de componentes do lado servidor para construção e instalação de aplicações CORBA. Esse modelo é bastante semelhante ao *Enterprise JavaBeans (EJB)* descrito anteriormente e utiliza padrões de projeto bem aceitos possibilitando que uma grande quantidade de código seja gerada automaticamente. Isso também permite que serviços de objeto sejam implementados pelo fornecedor do *container* e não pelo desenvolvedor da aplicação.

O Modelo de Componentes CORBA é parte da especificação CORBA 3 (23) e estende o modelo de objetos definindo funcionalidades e serviços em um ambiente padronizado que viabiliza a

implementação, gerência, configuração e instalação de componentes que se integram com Serviços CORBA, tais como transações, segurança, persistência e eventos.

CCM é dividido em dois níveis: básico e estendido (24). Os dois diferem quanto as potencialidades oferecidas. O primeiro fornece essencialmente um mecanismo simples para "componentizar" objetos CORBA. O segundo provê um conjunto de funcionalidades mais rico com extensões ao modelo CORBA. Um componente básico é muito similar, em funcionalidade, ao EJB definido na especificação *Enterprise JavaBeans 1.1* o que permite um mapeamento e integração mais fácil nesse nível.

Além de adicionar à *Interface Definition Language (IDL)* a capacidade de definir os componentes (IDL3), CCM também introduz a Linguagem de Definição de Implementação de Componentes (*Component Implementation Definition Language - CIDL*) com a objetivo de descrever os detalhes de implementação viabilizando a construção de um *framework* completo no lado do servidor, que será gerado, montado e distribuído.

2.4.1 O Componente CCM

Na perspectiva do cliente, um componente CCM é um objeto CORBA estendido que encapsula vários modelos de interação via interfaces e modelos de operações e conexão. Na perspectiva do servidor, componentes são unidades de implementação que podem ser instaladas e instanciadas independentemente em um ambiente de execução padrão estipulado pela especificação CCM.

Em geral, componentes são blocos de construção maiores que objetos possuindo a maioria das suas interações gerenciadas pelos *containers* a fim de simplificar e automatizar os aspectos de construção, composição e configuração. Componentes necessitam constantemente colaborar com diferentes tipos de aplicações/componentes. Para isso, CCM proporciona mecanismos denominados portas, atributos e referência de componentes:

- Portas: proporcionam aos clientes de um componente, aos componentes cooperantes e a outros elementos do ambiente, mecanismos necessários para interação;
- Atributos: são propriedades utilizadas normalmente para fins de configuração;
- Interface Equivalente: possibilita o acesso às portas do componente e identifica unicamente uma instância de um componente.

Existem quatro tipos de portas definidas pelo CCM:

1. **Facetas**: proporcionam interfaces que implementam as operações síncronas invocadas a partir de outros componentes, são também conhecidas como interfaces proporcionadas. Uma faceta

representa determinado aspecto funcional do componente ao qual pertence e não pode ser dissociada desse componente. A navegação através de facetas é centralizada em torno da interface equivalente do componente;

2. **Receptáculos**: indicam uma dependência de uso de uma interface pertencente a outro componente, tipicamente uma faceta. Receptáculos definem uma forma de conectar um componente a uma interface requerida e podem ser de dois tipos: receptáculos simples (somente uma referência pode ser conectada por vez) e receptáculos múltiplos (várias referências podem ser conectadas simultaneamente);

3. **Produtores de Eventos**: são pontos de conexão para produção de eventos de um dado tipo. Existem dois estilos de produtores de eventos: os publicadores de eventos que possuem múltiplos consumidores (um-para-muitos) e os emissores de eventos que possuem apenas um consumidor (um-para-um). No primeiro caso, existe um canal de distribuição de eventos, enquanto no segundo a conexão é realizada diretamente entre o consumidor e o produtor;

4. **Consumidores de Eventos**: são pontos de conexão capazes de consumir eventos de um dado tipo. Um consumidor de eventos permite que uma instância de componente receba eventos de determinado tipo podendo se subscrever a vários produtores de eventos. Quando uma subscrição ocorre, uma chave é gerada para identificar a subscrição do consumidor. Esta chave também é necessária para encerrar uma subscrição.

A figura 2.5 mostra a representação de um componente CCM com suas portas, atributos e interface equivalente.

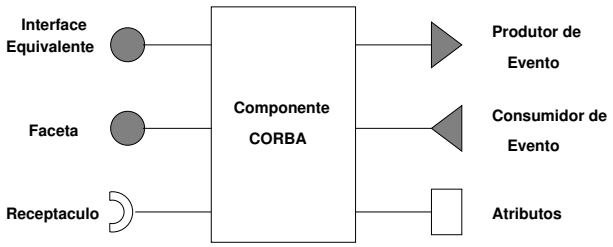

Fig. 2.5: O Componente CCM

Uma determinada instância de um componente é identificada através de sua interface equivalente. A partir dessa interface, um cliente pode obter as referências para as demais interfaces do componente, tais como portas e atributos. CCM proporciona o elemento *home* que padroniza a interface de gerenciamento do ciclo de vida de um componente através de operações de criação, remoção e procura de instâncias do componente. Para obter a referência do elemento *home*, o modelo CCM proporciona a interface *home finder*.

O mecanismo de interação do modelo CCM obedece ao estilo de interação operacional proposto pelo RM-ODP através de chamadas de procedimento remotos (facetas) e ao estilo de interação de sinal através do modelo de comunicação publicador/consumidor das interfaces de sinal dos componentes. Porém, o modelo CCM não define a interface *stream* utilizada no desenvolvimento de aplicações multimídia distribuídas. O OMG possui uma especificação para gerenciamento e controle de fluxo de áudio e vídeo, porém esse suporte não é integrado ao modelo de componentes.

2.4.2 Arquitetura de Containers CCM

O *container* é um elemento chave do CCM. Esse elemento do modelo de componentes constitui um *framework* que provê um ambiente de execução para uma ou mais implementações de componentes. Os *containers* fornecem as seguintes vantagens para os desenvolvedores de aplicações baseadas em componentes:

- Facilitam a tarefa de construção e instalação de componentes CCM assumindo os detalhes de baixo nível do *middleware*, tirando assim essa responsabilidade do desenvolvedor de componentes;
- São responsáveis, em conjunto com outras funcionalidades e ferramentas do modelo CCM, por localizar e criar instâncias de componentes, por interconectar componentes e garantir as políticas do componente com os serviços comuns, tais como ciclo de vida, segurança e persistência.

Os desenvolvedores selecionam e configuram o ambiente de desenvolvimento e suas respectivas políticas utilizando artefatos de programação fornecidos pelo *Framework* de Implementação de Componentes (*Component Implementation Framework - CIF*). Esse *framework* descreve como as partes funcionais e não-funcionais irão interagir (24). CIF automatiza a geração do *"glue code"* da implementação do componente padronizando a interação entre *container* e componentes.

A figura 2.6 mostra o modelo de programação do *container* CCM.

O modelo de programação do *container* CCM inclui:

- As Interfaces Externas, que representam as interfaces do componente hospedado disponíveis para seus clientes e estabelecem o contrato entre o desenvolvedor do componente e o cliente deste componente;
- As Interfaces do *Container*, que por sua vez, estabelecem o contrato entre um componente e o *container* que o hospeda e são compostas das interfaces internas e das interfaces de *callback*. As interfaces internas são utilizadas pelo componente hospedado para acesso aos serviços fornecidos pelo *container*. As interfaces de *callback* são implementadas pelo componente e disponibilizadas para o *container*;

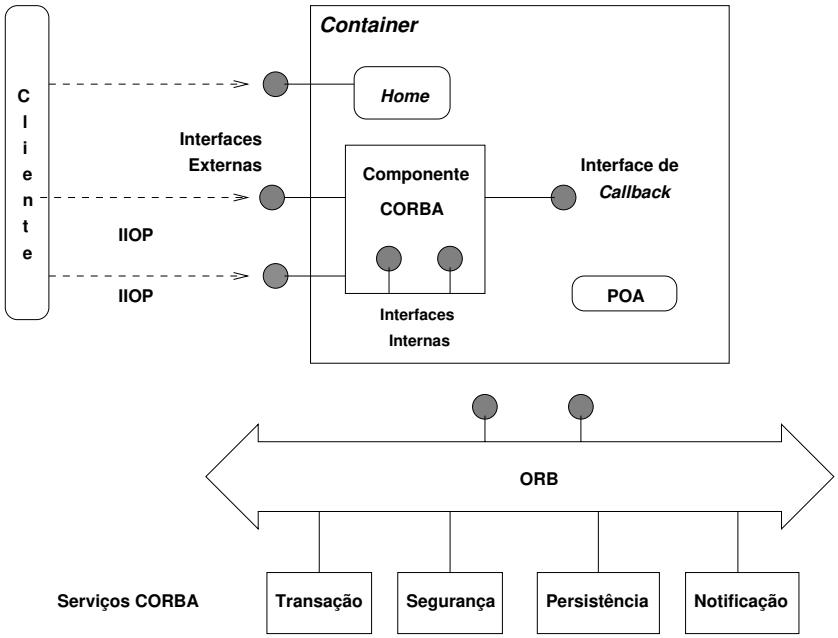

Fig. 2.6: Container CCM

- O Modelo de Uso que especifica o padrão de interação entre o *container*, o POA *(Portable Object Adapter)* e os serviços CORBA;
- A Interface *Home* que provê a navegação entre as interfaces do componente hospedado.

A fim de prover o encapsulamento das instâncias de um dado componente, o *container* utiliza o adaptador portável de objeto POA especializado para o componente e um conjunto de interfaces de acesso aos serviços CORBA. A comunicação entre o cliente e o *container* é realizado utilizando o protocolo IIOP (*Internet Inter-ORB Protocol*).

2.5 O Modelo de Componentes CM-tel

O modelo de componentes CM-tel (5) é especificado em dois níveis de abstração. O primeiro nível define um modelo genérico, estável e neutro em termos de tecnologia (linguagens de programação, plataformas de *middleware*, sistemas operacionais, protocolos de rede, etc.). O segundo nível especifica um mapeamento do modelo de componentes para um tecnologia alvo como CORBA, J2ME ou DCOM.

Com essa abordagem, obtém-se um ganho em flexibilidade uma vez que os componentes da aplicação bem como a especificação do modelo permanecem estáveis, enquanto a implementação desses componentes pode evoluir de forma independente. Outro benefício do modelo CM-tel é a possibilidade de ser executado em um dispositivo móvel com baixo poder de processamento, como

proposto nesse trabalho, graças as suas características de simplicidade de especificação, configuração, composição e instalação.

Modelos de componentes como o EJB e o CCM suportam somente as interfaces síncronas e de notificação segundo o modelo RM-ODP. Como conseqüência, o suporte à comunicação multimídia pelas aplicações constitui uma solução desvinculada do modelo de componentes. O modelo CM-tel possui como principais vantagens em relação a esses modelos o fato de ser neutro em termos de especificação e tecnologia, suportar interfaces de fluxo contínuo e permitir a especificação de atributos de QoS em alto nível para essas interfaces.

Os componentes CM-tel expõem as três interfaces prescritas pelo RM-ODP a fim de permitir a composição. Estas interfaces possuem papéis complementares como cliente/servidor e publicador/consumidor e são referenciadas como portas. A figura 2.7 mostra a estrutura de um componente CM-tel.

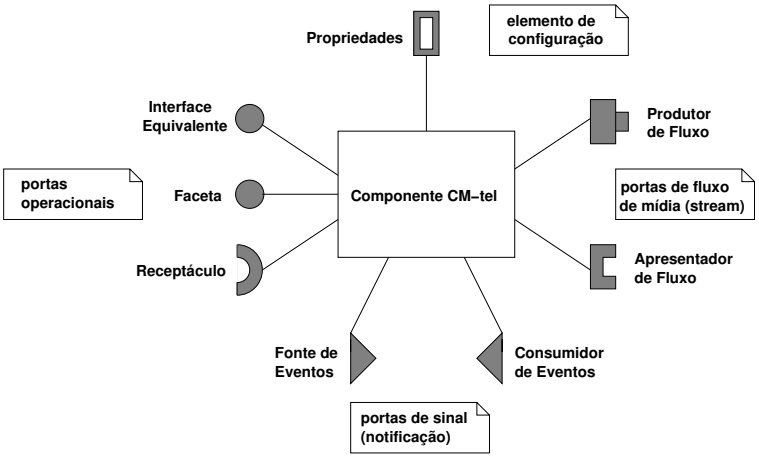

Fig. 2.7: Estrutura de um componente CM-tel

Os seguintes tipos de portas são definidas pelo modelo:

- facetas: interfaces que suportam operações síncronas;
- receptáculos: guardam as referências (facetas, componentes, objetos) requeridas pelo componente;
- fontes de eventos: interfaces de sinal que produzem eventos de notificação assíncrona (eventos);
- consumidores de eventos: interfaces de sinal que consomem eventos quando conectadas às fontes de eventos;
- produtores de fluxos: interfaces que geram e transmitem fluxos de mídia contínua (áudio e vídeo);

- apresentadores de fluxos: interfaces que recebem e apresentam fluxos de mídia quando conectadas aos produtores de vídeo.

Além das facetas, que contêm as operações da lógica da aplicação, são suportadas a interface equivalente, que identifica uma instância de componente unicamente e provê acesso às demais interfaces e a porta de propriedades, que proporciona um conjunto de pares atributo/valor armazenando informações sobre o estado e configuração do componente.

A especificação dos componentes e *containers* CM-tel pode ser feita tanto através de UML quanto por meio de documentos XML. As figuras 2.8 e 2.9 mostram exemplos de especificações de componentes contendo portas emissora e consumidora de eventos respectivamente.

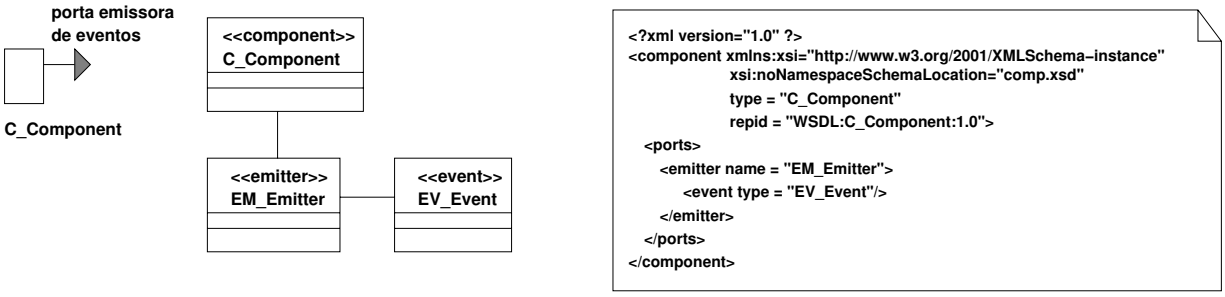

Fig. 2.8: Especificação de componente com porta emissora de eventos

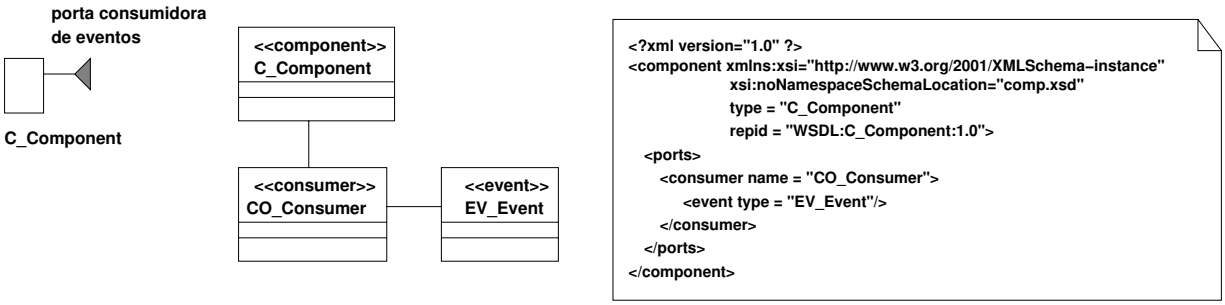

Fig. 2.9: Especificação de componente com porta consumidora de eventos

Os exemplos contêm a representação da estrutura do componente, sua especificação em UML e o respectivo documento XML de especificação. As figuras 2.10 e 2.11 mostram a especificações de uma faceta e um receptáculo.

Exemplos de especificação de portas consumidora e produtora de fluxo são mostrados nas figuras 2.12 e 2.13 respectivamente. Um exemplo de especificação de um componente contendo um elemento de propriedades também é mostrado na figura 2.14.

Um *container*, no modelo de componentes CM-tel, constitui o ambiente de execução para os componentes nele instalados e provê os recursos necessários para tal. Da mesma forma dos componentes, *containers* CM-tel são inteiramente descritos em UML. O modelo define um perfil UML

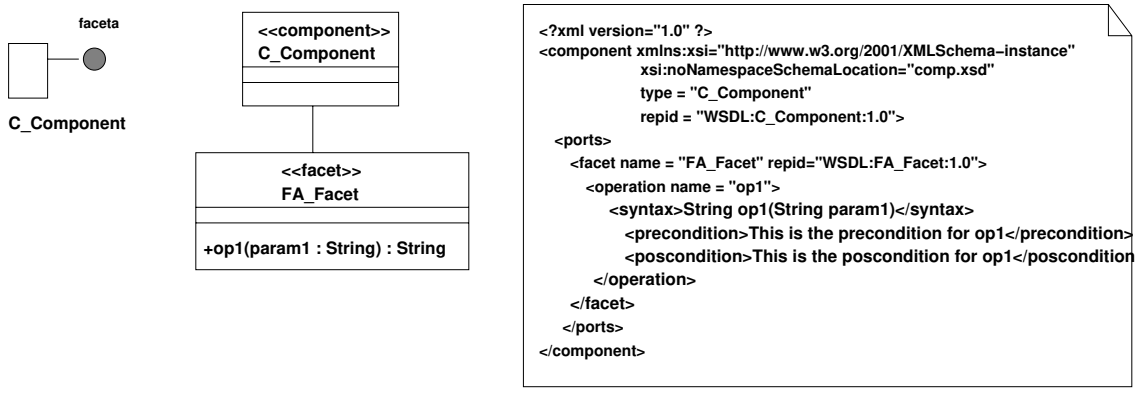

Fig. 2.10: Especificação de componente com uma faceta

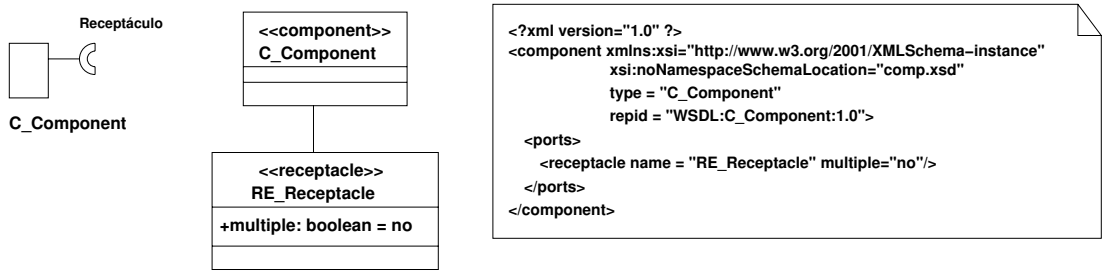

Fig. 2.11: Especificação de componente com um receptáculo

(UML *Profile*) para a descrição de componentes e *containers* (6). A especificação define os tipos de componentes que são instalados no *container* e descreve, para cada componente, suas dependências em relação a outros componentes. A figura 2.15 mostra um exemplo simples de especificação de um *container*.

O modelo CM-tel compartilha algumas características comuns aos modelos de componentes existentes, principalmente com o CCM. Porém, o modelo CM-tel é neutro em termos de especificação e tecnologia, suporta interfaces de fluxo contínuo e permite que atributos de QoS sejam especificados em alto nível para essas interfaces. Além disso, CM-tel é um modelo de componentes bem mais simples do que CCM e EJB, o que favorece o mapeamento e a construção de plataformas de *middleware*

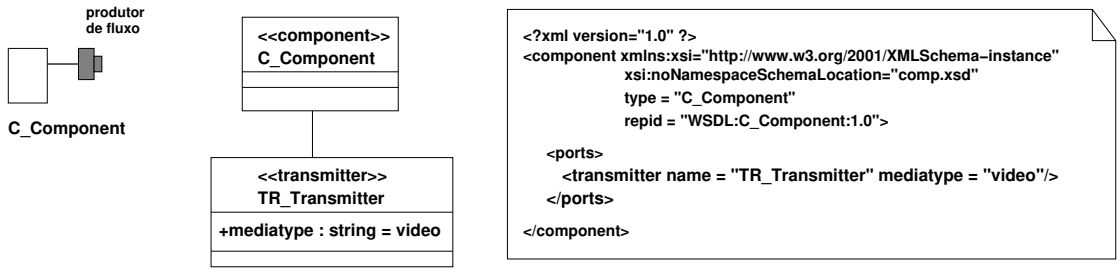

Fig. 2.12: Especificação de componente com uma porta transmissora de fluxo

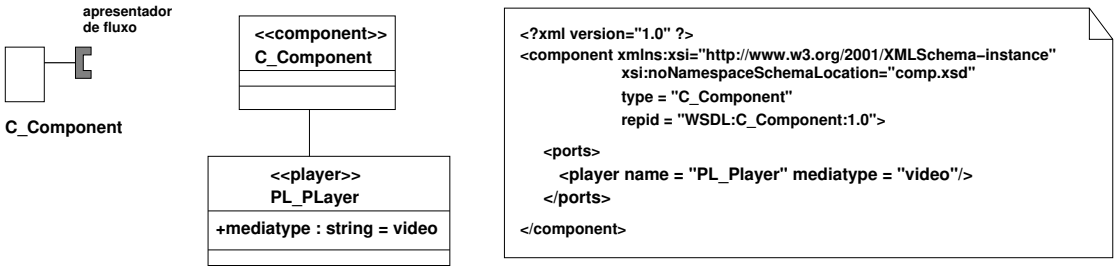

Fig. 2.13: Especificação de componente com uma porta apresentadora de fluxo

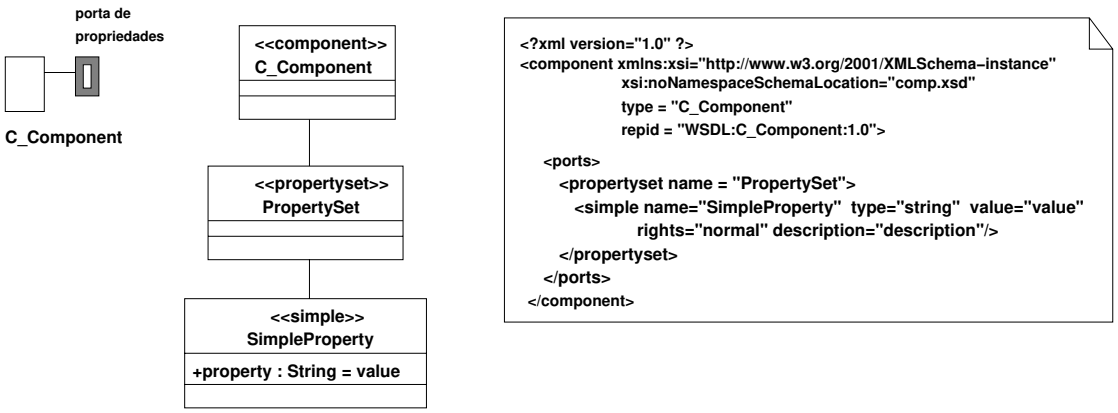

Fig. 2.14: Especificação de componente com propriedades

leves que suportam o modelo.

2.6 A Plataforma CCM-tel

A Plataforma CCM-tel constitui a realização do mapeamento do modelo CM-tel para a tecnologia CORBA/Java (6) (8). O caráter neutro do modelo CM-tel permite que os componentes e *containers* especificados em UML sejam reutilizados em mapeamentos desse modelo para outras tecnologias, tais como J2ME/*Web Services*. A utilização da plataforma CCM-tel através da *Internet* é prejudi-

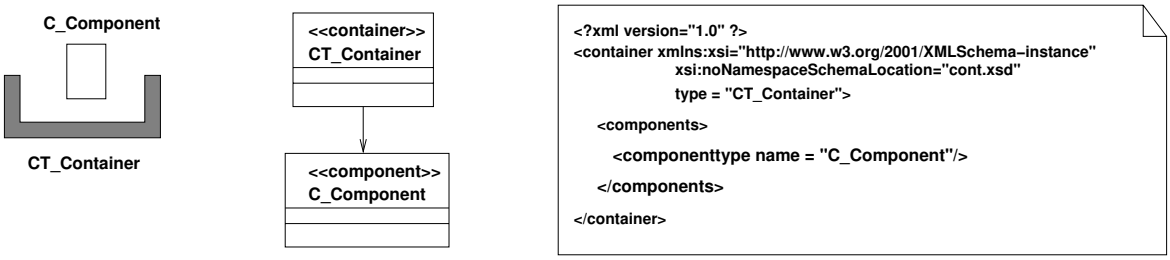

Fig. 2.15: Especificação de um container

cada pela utilização do protocolo IIOP (*Internet Inter-ORB Protocol*) do CORBA que necessita de configurações adicionais nos *firewalls*, ao contrário do protocolo HTTP usado em *Web Services*.

O processo de mapeamento é dividido em duas etapas: a transformação da especificação em UML para uma representação intermediária em XML e a transformação dessa especificação XML em código de componentes e *containers* em uma dada tecnologia.

Para realizar essas tranformações, é utilizada a linguagem XSLT *(Extensible Stylesheet Language Transformation)* (25) que é um vocabulário XML especificando um formato (estilo) de apresentação para documentos XML. Esta especificação descreve as regras de transformação de documentos XML em outros tipos de documentos, como HTML, código Java ou mesmo um documento XML com formato diferente.

Na primeira etapa do processo de mapeamento, as especificações UML são convertidas para uma representação XML no padrão XMI *(XML Metadata Interchange)* (9). Essa conversão se dá com o auxílio de ferramentas de projeto de *software* como ArgoUML (10) e Rational Rose (26). Folhas de estilo XSLT são utilizadas então para transformar essa representação XMI em documentos XML que possuem a sintaxe regida pelos XML *schemas* definidos para componentes e *containers* CM-tel a partir do perfil UML de cada um.

Os documentos que foram gerados na primeira etapa servem de entrada para as transformações realizadas na segunda etapa do processo de mapeamento. Essa etapa, ao contrário da anterior, é dependente de tecnologia. Dessa forma, as especificações de componentes e *containers* sofrem várias transformações, também através de XSLT, cada uma gerando um documento contendo código para uma determinada tecnologia.

Posteriormente, esse código é compilado e empacotado por uma máquina de construção de código. A plataforma CCM-tel consiste, assim, de uma máquina de transformação XSLT que realiza as duas etapas necessárias à geração de código descritas anteriormente, de um conjunto de serviços CORBA, de classes utilitárias para suporte a gerência de componentes e de uma máquina de construção de código.

A máquina de transformação utiliza o processador XSLT Xalan-Java (27) e a máquina de construção de código emprega a ferramenta Ant, ambos projetos de *software* livre disponibilizados pela *Apache Software Foundation*.

A plataforma gera, além dos 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 de comunicação entre componentes, *shell scripts* e arquivos HTML para instalação de *containers* e arquivos XML de construção contendo diretivas de compilação e montagem para a ferramenta Ant (*build.xml*). Ao desenvolvedor, resta acrescentar o código relativo à implementação das operações das facetas e de produção e tratamento de eventos. A figura 2.16 mostra os

elementos que compõem a plataforma CCM-tel.

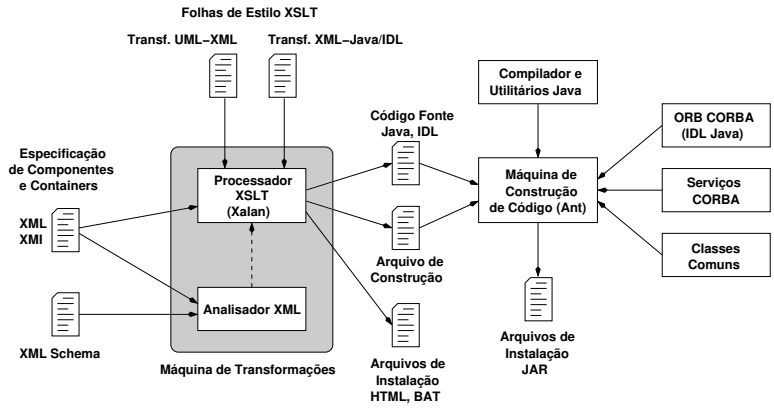

Fig. 2.16: Elementos da plataforma CCM-tel

2.6.1 Containers CCM-tel

O *container* CCM-tel disponibiliza fábricas de componentes e agentes (agências) que serão hospedados no *container*. O *container* ainda conta com um objeto localizador de fábricas através do qual são obtidas as referências das fábricas de componentes e agentes. Artefatos da arquitetura CORBA como o ORB, o adaptador de objetos portáveis são instanciados pelo *container* e disponibilizados para os componentes e agentes abrigados. Através de propriedades de distribuição disponibilizada pelo *container*, um componente ou agente interno ou externo ao *container* pode alterar os parâmetros definidos originalmente pelo descritor de distribuição.

A figura 2.17 mostra a arquitetura de um *container* CCM-tel.

2.6.2 Interação Componentes-Agentes Móveis

O *container* CCM-tel gerencia agentes através de agências que, por sua vez, utilizam o MAF (*Mobile Agents Facilities*) do CORBA. A plataforma CCM-tel implementa na linguagem Java um sistema de agentes leve e de baixo *overhead*, compatível com a especificação MAF.

Agentes CCM-tel devem implementar a interface Java *Runnable*, para que os agentes sejam executados como *threads*, e a interface *Agent*, que permite às agências controlarem seus agentes através de métodos de *callback*. Agentes podem também acessar todos os recursos da tecnologia CORBA, tais como serviços mantidos pelo ORB, acessar o ORB como cliente ou registrar objetos no POA como objetos servidores (serventes) CORBA. Isso permite que agentes e componentes interajam através do ORB. Dessa forma, um agente pode obter a referência da interface equivalente de um componente e,

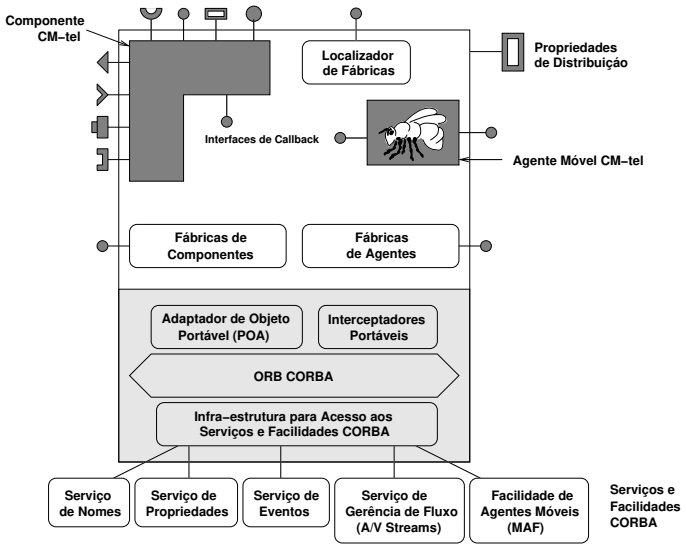

Fig. 2.17: Arquitetura de um *container* CCM-tel

então, obter as demais interfaces do componente. A partir dessas interfaces, o agente pode interconectar componentes, realizar alterações de propriedades ou invocar métodos definidos pelas interfaces. Os componentes podem, da mesma forma, localizar agentes e interagir com suas interfaces através do ORB.

2.7 Considerações Finais

Este capítulo forneceu uma visão geral sobre o desenvolvimento baseado em componentes ressaltando as vantagens da utilização desse paradigma na construção de sistemas de software de grande porte. Algumas definições do termo componente de *software* também foram apresentadas, inclusive a de Heineman e Council (19) usada nesse trabalho. Adicionalmente, o capítulo descreveu resumidamente os modelos de componentes *Enterprise JavaBeans* (EJB), CORBA *Component Model* (CCM) e CM-tel destacando as vantagens desse último principalmente em relação ao desenvolvimento de aplicações telemáticas e à neutralidade em termos de tecnologia. Neste capítulo, também foi apresentada uma visão geral da plataforma CCM-tel bem como algumas desvantagens da tecnologia CORBA utilizada por essa plataforma.

Capítulo 3

Mapeamentos do Modelo CM-tel

Neste capítulo, será abordado o mapeamento do modelo de componentes CM-tel para a tecnologia de *Web Services*. Para isso, serão descritos os requisitos, a análise e o projeto de um mapeamento para *Web Services*. Posteriormente, é abordada a geração automática de código fonte para uma plataforma resultante desse mapeamento.

3.1 Requisitos de um Mapeamento para *Web Services*

Mapeamentos CM-tel produzem modelos específicos para determinada tecnologia e preservam a estrutura e a especificação XML dos componentes e *containers*. Será descrito a seguir, como um mapeamento do modelo utiliza *Web Services* para prover a comunicação entre os componentes distribuídos.

Existem várias definições de *Web Services* (28) em diferentes níveis de abstração. Segundo Clabby (29), *Web Services* é uma arquitetura de computação distribuída de fraco acoplamento que permite a comunicação entre aplicações. Nesse sentido, essa tecnologia é utilizada no mapeamento do modelo CM-tel como forma de expor/consumir os serviços oferecidos pelas portas dos componentes. A tecnologia de *Web Services* tem como objetivo proporcionar a comunicação entre os componentes CM-tel e o acesso a outros serviços através da Internet.

Para dar suporte ao mapeamento, um Serviço de Notificação baseado em documentos XML torna-se necessário. Este serviço é utilizado pelas portas emissoras e consumidoras de eventos. Adicionalmente, um Serviço de Gerência de Fluxo Contínuo é requerido pelas portas produtoras e consumidoras de fluxo.

Podem ser citados os seguintes fatores que motivaram a escolha da tecnologia de *Web Services* para um mapeamento do modelo CM-tel:

- Independência de plataforma de *hardware*, sistema operacional e linguagem de programação;

- Especificação aberta e disponibilizada pelo W3C *(World Wide Web Consortium)*;
- Baseia-se em padrões de Internet bem estabelecidos, como HTTP e XML;
- Maior facilidade para atravessar *firewalls*, se comparado a CORBA ou RMI, devido ao uso do protocolo SOAP sobre HTTP;
- Possibilidade de utilização em dispositivos móveis como telefones celulares e PDAs.

Web Services são acessados através de SOAP *(Simple Object Access Protocol)* (30) que constitui um protocolo "leve" e baseado em texto (XML) cujas mensagens podem ser enviadas utilizando o protocolo HTTP.

Um mapeamento do modelo CM-tel utilizando *Web Services* introduz particularidades da tecnologia aos modelos de *containers* e de distribuição resultantes desse mapeamento. A figura 3.1 mostra a arquitetura do *container* com extensões específicas.

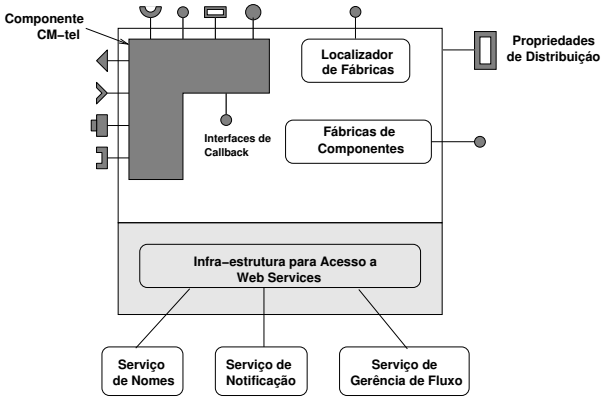

Fig. 3.1: Arquitetura de um *container* resultante do mapeamento *Web Services*

A infra-estrutura de acesso a *Web Services* provê o processamento das mensagens SOAP necessário para comunicação entre portas e acesso aos serviços externos ao *container*. Essa comunicação tanto pode ser no estilo síncrono *(Remote Procedure Call - RPC)* quanto no estilo assíncrono (envio de mensagens).

A partir de descritores de distribuição com extensões específicas da tecnologia escolhida, os *containers* são distribuídos e configurados. O descritor de distribuição especifica, em XML, as políticas que determinam a maneira com que o *container* gerencia certos requisitos não-funcionais como políticas de QoS para fluxo de mídia contínua.

3.1.1 Aderência ao Modelo CM-tel

A seguir, são descritos brevemente alguns elementos essenciais (5) do modelo CM-tel que devem ser implementados por um mapeamento que utilize *Web Services* como mecanismo de comunicação

entre componentes e de acesso a serviços.

Padrão para Especificação do Comportamento de Componentes

Um componente mapeado para *Web Services* deve possuir as seguintes características de comportamento:

- padrão de interação que proporcione uma forma homogênea de interação entre portas complementares onde são definidos os modelos de interação cliente/servidor (entre receptáculos e facetas) e publicador/subscritor (entre as demais portas);
- especificação do componente através de uma notação neutra utilizando a linguagem UML ou a linguagem XML;
- interfaces WSDL *(Web Services Description Language*) (31) geradas por uma ferramenta de geração de código descrevendo a sintaxe dos aspectos funcionais;
- semântica dos aspectos funcionais provida através de restrições em UML;
- suporte a adaptação do ambiente de execução, a aspectos de segurança e a fluxos de áudio e vídeo com garantias de qualidade de serviço.

Esquema de Nomes Padronizado

O padrão para nomes deve definir nomes globais únicos para a identificação de componentes, fábricas de componentes e localizador de fábricas, onde:

- o nome do localizador de fábricas (*Home Finder*) representa o nome de uma instância de um *container*;
- o nome da fábrica de componentes (*Component Home*) é registrado no localizador de fábricas do *container*;
- cada instância de um componente possui uma chave (String) registrada na fábrica do componente.

Padrão para Interoperabilidade

Um mapeamento do modelo para *Web Services* mantém interoperabilidade através de SOAP/HTTP, padrão utilizado para comunicação entre componentes através da Internet. Dessa forma, somente é permitida a passagem de objetos que possam ser definidos através de documentos XML.

Padrão para Customização

A customização de componentes é feita através do elemento de configuração (propriedades). As propriedades de um componente, assim como as demais portas, devem possuir suas interfaces expostas através de *Web Services*. Essas propriedades são configuradas no momento da instanciação do componente e modificadas em tempo de execução.

Padrão para Composição

O modelo resultante do mapeamento deve definir um padrão que suporte as seguintes formas de composição:

- composição orientada a conexão proporcionada através de portas complementares de interconexão (portas operacionais, de sinal e de fluxo contínuo) e através de receptáculos que permitem o acesso do componente a referências de facetas e objetos e a interfaces de componentes;
- composição contextual, através do *container Web Service*. Um *container* está implicitamente conectado a todas as instâncias de componentes hospedadas podendo interceptar e gerenciar todas as chamadas realizadas e recebidas por ele.

Padrão para Suporte a Evolução

O padrão para suporte a evolução deve permitir que mudanças em componentes sejam realizadas sem qualquer efeito sobre os demais componentes. As alterações nos componentes tanto podem ser através de uma nova implementação com adição de novas interfaces preservando as antigas, quanto através de atualizações de uma mesma implementação.

Padrões para Empacotamento e Distribuição

Um mapeamento para *Web Services* deve utilizar descritores de distribuição e de montagem expressos na linguagem XML ou derivados de diagramas UML de distribuição e colaboração respectivamente. Esses descritores possibilitam a geração automática de código para conexões entre portas de componentes.

3.2 Dispositivos Móveis

Os dispositivos móveis, tais como telefones celulares e PDAs, ainda possuem um baixo poder de processamento e armazenamento, se comparados a microcomputadores de mesa. Porém, aplicações instaladas nesses aparelhos são perfeitamente capazes de consumir *Web Services* como um cliente

qualquer. Os dispositivos móveis podem até se tornar pequenos servidores com algumas limitações quanto à performance.

A utilização de *Web Services* envolvendo esses dispositivos deve, naturalmente, respeitar a independência de plataforma de *hardware*, sistema operacional e linguagem de programação. Serviços devem ser oferecidos de forma independente do cliente que o requisita da mesma forma que, para o cliente, não importa em que plataforma o serviço é executado ou em que linguagem de programação o serviço foi escrito. Dessa maneira, a utilização de *Web Services* no mapeamento do modelo CM-tel permite o uso de tais dispositivos como parte integrante de aplicações baseadas em componentes distribuídos.

Para que haja interoperabilidade, a infra-estrutura que provê a disponibilização e o acesso a *Web Services* pelos dispositivos móveis tem que utilizar os padrões de troca de mensagens (SOAP), descrição de serviços (WSDL) e publicação de serviços (*Universal Description, Discovery and Integration* - UDDI). Esses padrões são especificações da W3C integrantes da própria tecnologia de *Web Services*, sendo implementados pela maioria dos *toolkits* disponíveis para dispositivos móveis. Mais detalhes sobre opções de tecnologia serão fornecidos no capítulo 4.

3.3 Análise e Projeto do Mapeamento CM-tel para Web Services

Um mapeamento do modelo de componentes CM-tel que utilize *Web Services* como mecanismo de comunicicação precisa prover elementos capazes de interpretar mensagens SOAP usadas em chamadas de procedimento remoto. Similarmente a outras tecnologias de objetos distribuídos como RMI (32) e CORBA, a infra-estrutura fornece classes locais *(stubs)* que representam o serviço remoto através de geração automática de código.

Do lado do servidor, elementos que fazem a interface entre o processador SOAP e o provedor do serviço *(skeletons)* também são fornecidos pela infra-estrutura. A figura 3.2 mostra como esses elementos interagem.

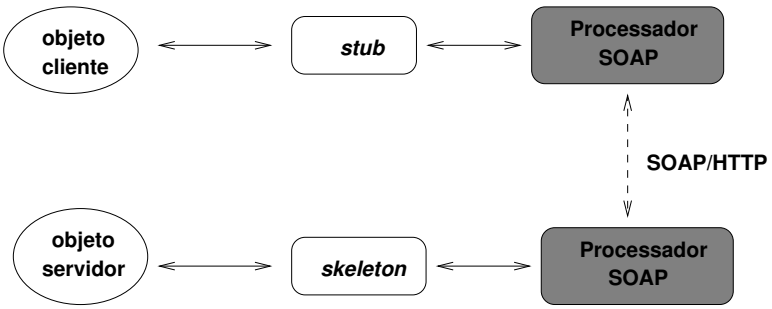

Fig. 3.2: Comunicação entre os objetos distribuídos no mapeamento para *Web Services*

O objeto cliente, de forma transparente, invoca um método do objeto remoto através do *stub* que possui as mesmas assinaturas de métodos do *skeleton* e do objeto remoto. Posteriormente, o *stub* utiliza o Processador SOAP para construir a mensagem SOAP e enviá-la via HTTP ao Processador SOAP do lado servidor que extrai o nome do método e seus parâmetros. Em seguida, o Processador SOAP invoca o método em questão no *skeleton* que, por sua vez, repassa a chamada ao objeto servidor. A mensagem SOAP de retorno é construída pelo Processador SOAP do lado do servidor e contém o valor de retorno do método, se for o caso. Ocorrendo alguma exceção durante a execução do método remoto, a mensagem SOAP de retorno possuirá a descrição dessa exceção.

O protocolo SOAP utilizado para invocação dos serviços é um protocolo baseado em texto. Uma mensagem SOAP constitui-se de um documento XML que é transmitido utilizando outros protocolos de aplicação tais como HTTP e SMTP. No caso do HTTP, a mensagem SOAP é incluída como o corpo *(body)* da requisição. Atualmente, a grande maioria das implementações de *Web Services* faz uso da combinação SOAP/HTTP. Dessa forma, o protocolo segue o modelo *request/response* utilizando a mesma conexão HTTP para envio da requisição e recebimento da resposta. A cada chamada de procedimento remoto ou envio de mensagem, uma nova conexão é criada.

Será descrito a seguir, como a infra-estrutura de *Web Services* é utilizada pelos elementos de um mapeamento do modelo CM-tel.

3.3.1 Componente

Os componentes CM-tel interagem entre si através de suas portas e elementos de configuração. Para tornar possível essa comunicação, o protocolo SOAP é utilizado tanto em chamadas de procedimento remoto quanto no envio de mensagens. O modelo de projeto para portas e elementos de configuração CM-tel mostrado na figura 3.3 indica a forma geral de como cada um dos elementos da estrutura de um componente CM-tel é implementado. Esta forma contém uma fábrica para instanciação da classe que implementa a interface do elemento.

A implementação dos elementos do modelo utiliza a infra-estrutura de *Web Services* fornecida pelo mapeamento para acessar tanto os serviços oferecidos por outros componentes quanto o Serviço de Notificação. A seguir, será mostrado como cada elemento do componente interage com essa infra-estrutura de *Web Services*.

Porta Produtora de Eventos

A implementação da porta produtora de eventos interage com o Serviço de Notificação através de uma referência *(stub)* para esse serviço que, por sua vez, também possui uma interface *Web Service* como mostrado na figura 3.4.

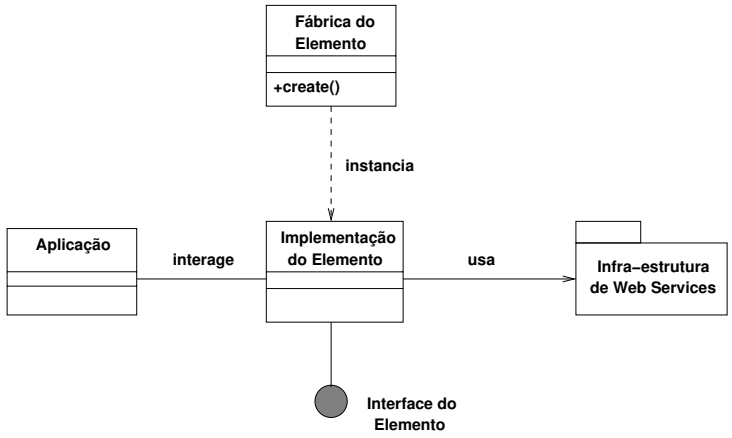

Fig. 3.3: Modelo de projeto para portas e elementos de configuração

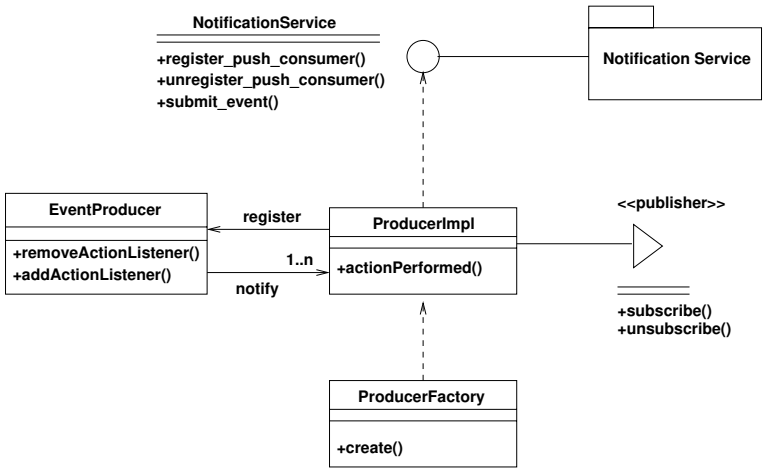

Fig. 3.4: Porta produtora de eventos no mapeamento Web Services

A figura 3.5 ilustra a dinâmica da interação entre os objetos do modelo e as interfaces dos serviços para o cenário de emissão de um evento.

1. A porta produtora de eventos é criada pela fábrica de elementos.
2. A porta produtora de eventos se registra em uma classe da aplicação responsável por gerar eventos locais (classe *EventProducer*).
3. Ao ser gerado um evento, a aplicação notifica a porta sobre a ocorrência do evento.
4. A porta produtora de eventos invoca o Serviço de Notificação enviando o documento XML que descreve o evento.

Uma vez que o evento foi repassado ao Serviço de Notificação, este fica responsável por realizar a entrega do evento aos possíveis destinatários (portas consumidoras de eventos).

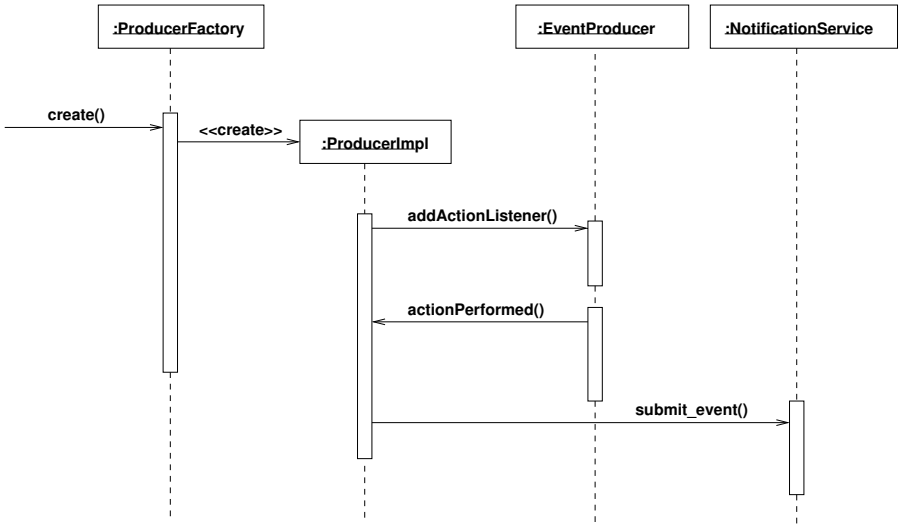

Fig. 3.5: Dinâmica da operação de emissão de eventos

Porta Consumidora de Eventos

A porta consumidora de eventos é exposta pelo componente como um *Web Service* através do qual o Serviço de Notificação realiza a entrega do evento no estilo *push*. O recebimento de um evento está condicionado a um filtro referente a cada porta consumidora que é registrada junto ao Serviço de Notificação. A figura 3.6 mostra a interação entre as classes e interfaces do modelo responsáveis pelo consumo de eventos.

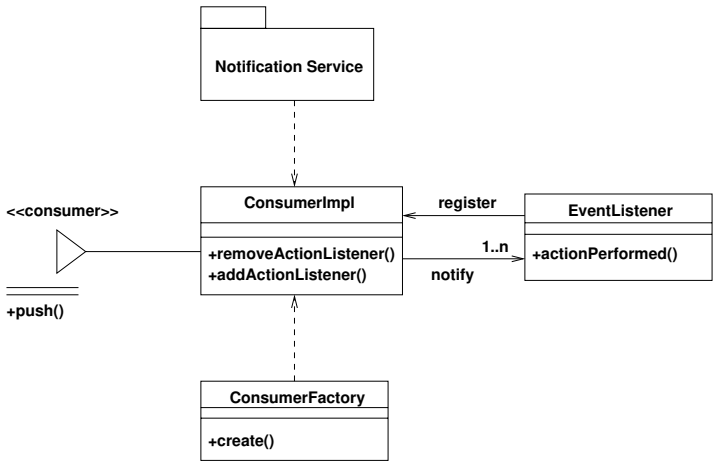

Fig. 3.6: Porta consumidora de eventos no mapeamento Web Services

A figura 3.7 ilustra a dinâmica do recebimento de um evento pela porta consumidora de eventos.

1. A porta consumidora de eventos é criada pela fábrica de elementos.

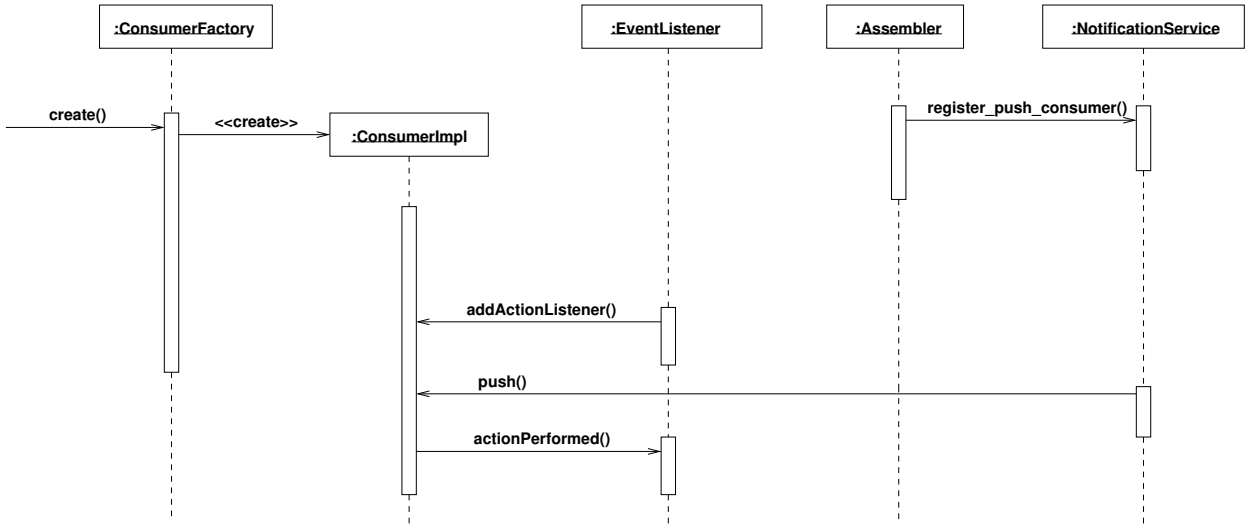

Fig. 3.7: Dinâmica da operação de consumo de eventos

2. O elemento responsável pela montagem da aplicação (*Assembler*) registra a porta consumidora de eventos no Serviço de Notificação.
3. A classe da aplicação responsável por tratar eventos (classe *EventListener*) se registra na porta consumidora de eventos da aplicação.
4. O Serviço de Notificação envia o evento para porta consumidora de eventos.
5. Ao receber o evento, a porta consumidora notifica a aplicação sobre a chegada do evento.
6. A aplicação trata o evento de acordo com a sua lógica de negócios.

Uma implementação para o Serviço de Notificação será descrita com maiores detalhes no capítulo 4.

Faceta

Uma faceta utiliza a infra-estrutura do mapeamento para expor as suas operações como *Web Services*. Para isso, é realizado o processamento da mensagem SOAP contendo a invocação do serviço da faceta. Após a execução do serviço, a mensagem SOAP de retorno é enviada de volta ao cliente. A figura 3.8 mostra o uso da infra-estrutura de *Web Services* para invocação de operações de uma faceta.

Receptáculo

No mapeamento para *Web Services*, o receptáculo armazena a referência de um serviço remoto, normalmente uma faceta. Para isso, é utilizada uma classe local que represente esse serviço *(stub)* que pode ser obtido durante o processo de montagem do componente ou através da geração dinâmica

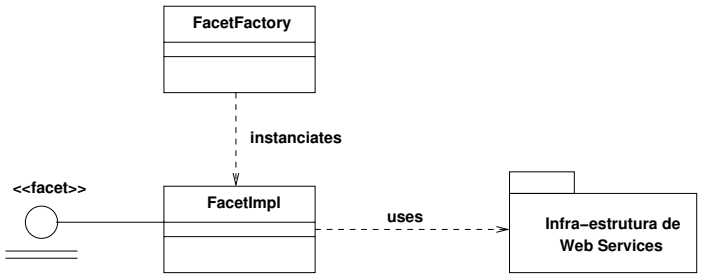

Fig. 3.8: Faceta no mapeamento Web Services

a partir da descrição WSDL do serviço contida, por exemplo, em um registro UDDI. A figura 3.9 mostra a organização das classes e interfaces para esse elemento do mapeamento.

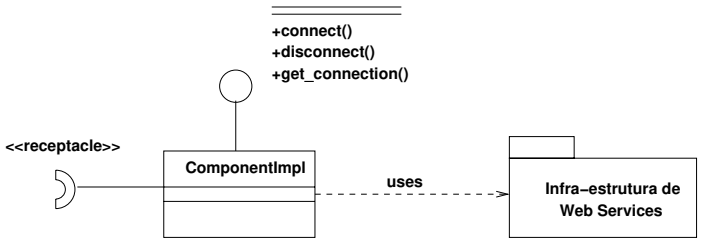

Fig. 3.9: Receptáculo no mapeamento Web Services

O receptáculo expõe operações de conexão e desconexão que disponibilizam as referências dos serviços requeridos. As operações do receptáculo são realizadas na interface equivalente do componente.

Propriedades

De forma similar às facetas, as operações de manipulação de propriedades também são expostas como *Web Services*. Porém, essas operações são limitadas a métodos de acesso para cada propriedade de acordo com a característica pública ou privada contida na especificação XML. A figura 3.10 mostra a interação entre as classes do modelo para manipulação de propriedades e a infra-estrutura de *Web Services*.

Uma configuração inicial das propriedades é utilizada pela fábrica para criação do conjunto de propriedades que será disponibilizada aos clientes do componente. Essa configuração inicial é descrita pela especificação XML do componente. A figura 3.11 ilustra a dinâmica da alteração de uma propriedade do componente após a criação do conjunto de propriedades.

1. O conjunto de propriedades é criado pela fábrica de elementos;
2. O conjunto de propriedades recebe uma requisição de alteração de propriedade;

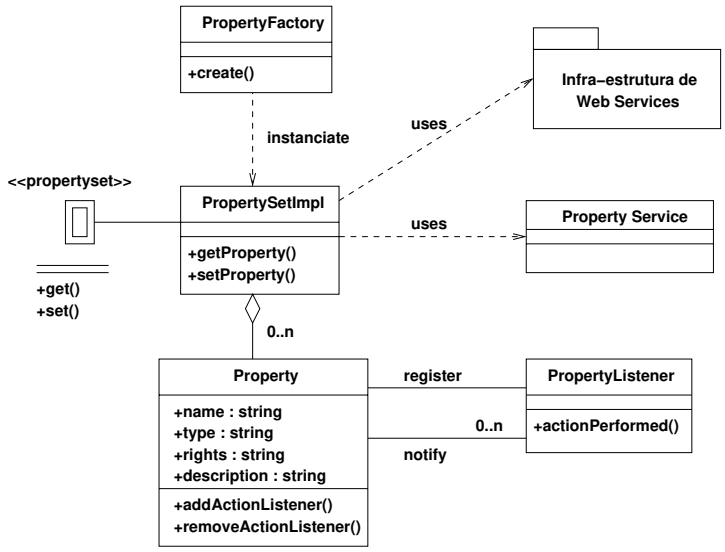

Fig. 3.10: Propriedades no mapeamento Web Services

3. O conjunto de propriedades repassa a requisição para o serviço (*PropertyService*) responsável pela manutenção das propriedades;
4. O serviço de propriedades checa se a propriedade a ser modificada já foi definida;
5. Caso exista a propriedade, a mesma recebe o valor especificado na requisição;
6. Os elementos previamente registrados como *listeners* da propriedade (*PropertyListener*) são notificados da mudança ocorrida localmente.

Porta Produtora de Fluxo

A porta produtora de fluxo no mapeamento *Web Services* necessita de um serviço de *streamming* para geração e transmissão do fluxo de áudio e vídeo. A implementação da porta está associada ao serviço de *streaming* que irá efetivamente realizar a transmissão desse fluxo. Uma possível abordagem para esse serviço seria a utilização de *NetCams* para captura e transmissão de mídia através da Internet. Dessa forma, o elemento consumidor de mídia seria capaz de acessar o endereço URL associado à câmera e reproduzir o conteúdo multimídia utilizando um protocolo específico, como por exemplo, o protocolo JPEG/UDP.

A figura 3.12 mostra a relação entre as classes do modelo para produção de fluxo de mídia.

A operação *subscribe* recebe como parâmetro o endereço (referência) da porta consumidora de fluxo contínuo e retorna um *cookie* contendo informações da conexão, como por exemplo a URL associada à câmera ou microfone e a URL do apresentador de fluxo (*player*). A operação *unsubscribe* disassocia a conexão entre produtor e apresentador de fluxo.

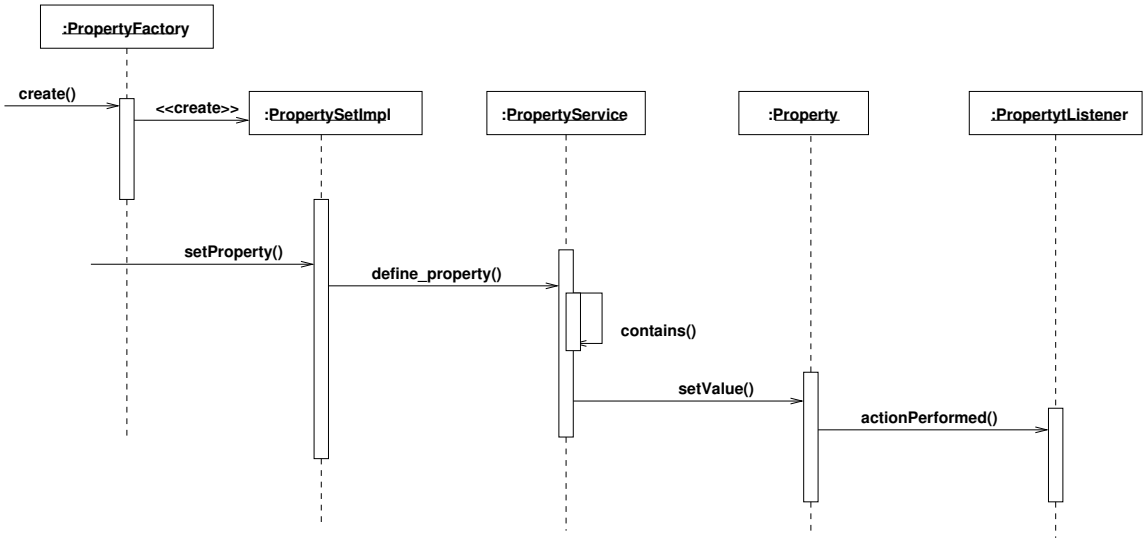

Fig. 3.11: Dinâmica de alteração de propriedades do componente

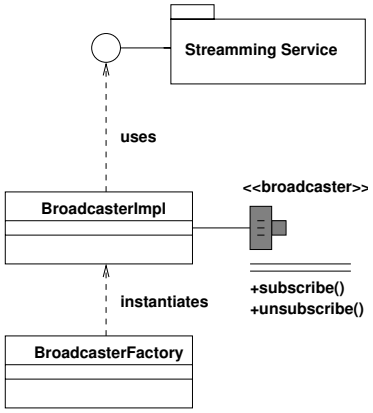

Fig. 3.12: Porta produtora de fluxo no mapeamento Web Services

Porta Apresentadora de Fluxo

A porta apresentadora de fluxo possui operações de controle da exibição do conteúdo multimídia. A implementação da porta acessa o serviço de *streamming* que provê tal conteúdo. A figura 3.13 mostra a interação das classes do modelo para porta consumidora de fluxo.

A operação *start* recebe como parâmetro o endereço do serviço de *streaming* para que a implementação da porta consumidora tenha acesso ao fluxo de mídia.

A figura 3.14 ilustra a interação entre os elementos produtor e consumidor de mídia contínua.

1. O elemento de montagem (*Assembler*) recebe uma requisição de conexão;
2. O elemento produtor de fluxo (*Broadcaster*) recebe uma requisição de subscrição, por parte do *Assembler*, contendo a referência do elemento consumidor (*Player*);

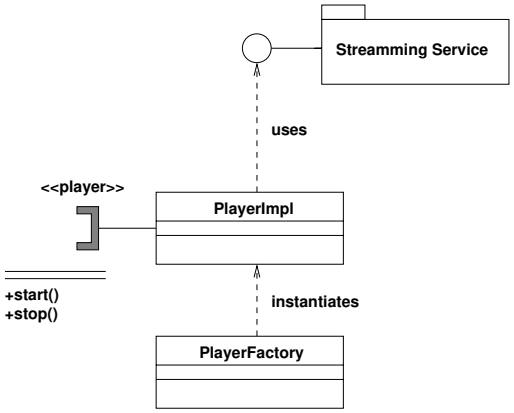

Fig. 3.13: Porta consumidora de fluxo no mapeamento Web Services

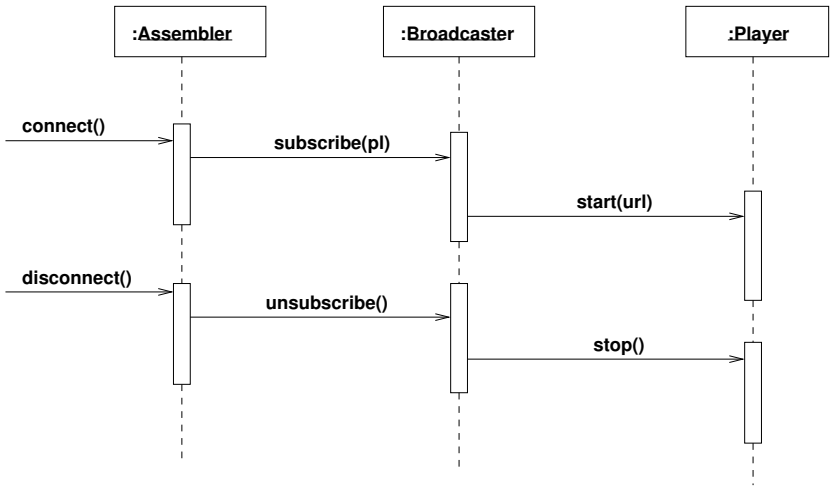

Fig. 3.14: Interação entre as portas consumidora e produtora de fluxo de mídia

3. O elemento *Broadcaster* invoca a operação *start* para iniciar a reprodução do fluxo de mídia sendo a URL do serviço de *streaming* fornecida como parâmetro;
4. O elemento de montagem (*Assembler*) recebe uma requisição de desconexão;
5. O elemento de montagem invoca a operação de desconexão do elemento *Broadcaster*;
6. O elemento *Broadcaster* invoca a operação *stop* para término da reprodução do fluxo de mídia.

As invocações das operações de conexão e desconexão entre as portas podem ser realizadas pelo elemento de montagem (*Assembler*), como mostrado do diagrama da figura 3.14.

3.3.2 Container

No mapeamento do modelo CM-tel para *Web Services*, o *container* disponibiliza classes utilitárias para processamento das mensagens SOAP endereçadas aos componentes hospedados. O *container*

também utiliza um servidor HTTP para o recebimento das requisições e envio de respostas, ambas contendo mensagens SOAP. A figura 3.15 mostra a infraestrutura de *Web Services* disponibilizada pelo *container*.

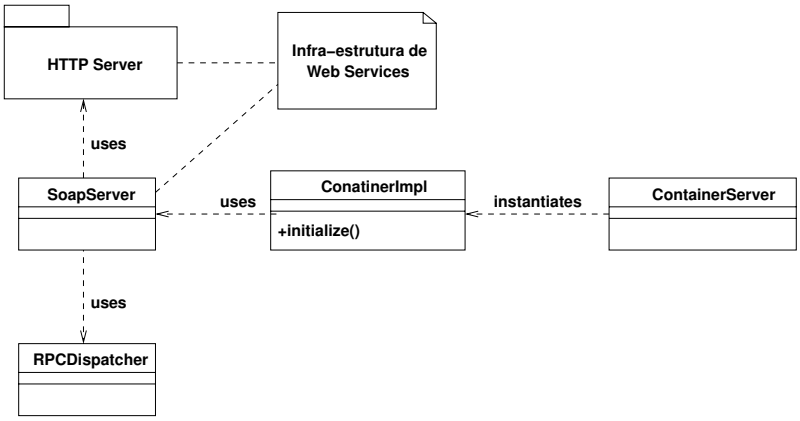

Fig. 3.15: Estrutura do *container* no mapeamento Web Service

Essa infra-estrutura contém um servidor HTTP e classes para processamento de mensagens SOAP. Informações relativas a execução do serviço, tais como nome do serviço, parâmetros e objeto que irá executar o serviço são extraídas da requisição SOAP/HTTP. Nesse ponto, o *container* poderá interceptar a requisição do serviço e realizar procedimentos relativos a requisitos não-funcionais tais como segurança e persistência. Da mesma forma, a mensagem SOAP de retorno é enviada como resposta da requisição HTTP pelo servidor disponibilizado no *container*.

O elemento *ContainerServer* é responsável pela inicialização do *container* e pela instanciação de sua implementação (*ContainerImpl*). Esse último elemento utiliza o *SoapServer* para recebimento e envio das mensagens SOAP. Adicionalmente, o elemento *RPCDispatcher* recebe o endereço do serviço bem como o nome da operação a ser invocada e seus respectivos parâmetros.

3.3.3 Montagem e Distribuição

A montagem da aplicação é feita por um componente especial gerado automaticamente a partir dos descritores de projeto, de montagem e de distribuição. Esses descritores são gerados automaticamente a partir de diagramas UML de distribuição e de colaboração. O componente de montagem (*Assembler*) possui uma faceta com operações de conexão e desconexão responsáveis por criar/destruir as referências dos serviços remotos nos receptáculos. Uma vez que as operações de montagem estão contidas na implementação da faceta, um elemento externo ao componente é responsável por invocar os métodos dessa porta para que a montagem da aplicação tenha início.

A distribuição dos componentes e *containers* nos respectivos nós de processamento depende da

linguagem de programação utilizada. No caso do mapeamento CM-tel para *Web Services* optar pela linguagem Java para implementação de componentes e *containers*, a distribuição poderá ser feita através de arquivos *jar* executáveis.

Um exemplo completo contendo a montagem e distribuição de uma aplicação será apresentado no capítulo 5.

3.4 Geração de Código

O mapeamento do modelo CM-tel para *Web Services* deve escolher uma linguagem de programação para implementação dos componentes e *containers*. Uma vez escolhida a tecnologia alvo do mapeamento como, por exemplo *Web Services*/Java, são geradas interfaces WSDL e classes Java, que por sua vez serão posteriormente compiladas e empacotadas.

A seguir, são listados os artefatos relativos a infra-estrutura de *Web Services* gerados a partir de elementos presentes na especificação XML de componentes e *containers*:

- interfaces WSDL: geradas para descrição do serviço oferecido pelas portas consumidoras e emissoras de eventos, portas de propriedades, facetas e receptáculos;
- *stubs* dos serviços: gerados a partir das interfaces WSDL, são responsáveis pelo acesso ao serviço no lado do cliente;
- *RPCDispatcher*: responsável pela escolha do componente que irá executar o serviço baseado no processamento da requisição HTTP;
- *RPCProvider*: responsável pela escolha da porta do componente que irá executar o serviço baseado no processamento da mensagem SOAP. Essa classe possui a informação sobre o método a ser chamado com seus respectivos parâmetros.

Um exemplo de implementação da máquina de geração de código para tecnologia *Web Services*/J2ME contendo todos os arquivos gerados será apresentado no capítulo 5.

3.5 Considerações Finais

Neste capítulo, foram discutidos mapeamentos do modelo de componentes CM-tel para as tecnologias CORBA e *Web Services*. Foi avaliado como os *Web Services* fornecem os mecanismos necessários para a comunicação entre as portas dos componentes do modelo.

Discutiu-se também os requisitos necessários para realização de um mapeamento para *Web Services* e sua aderência ao modelo CM-tel. Modelos de projeto de alguns dos elementos de componentes

e *containers* CM-tel foram apresentados a fim de fornecer um ponto de partida para o desenvolvimento de um mapeamento *Web Service* em conjunto com outra tecnologia, como por exemplo, Java.

Capítulo 4

A Plataforma MECM-tel

Este capítulo apresenta a plataforma MECM-tel (*Micro Edition Component Model for telematic applications*) que constitui um mapeamento do modelo CM-tel para a tecnologia *Web Services*/J2ME. Tal mapeamento permite que componentes aderentes ao modelo sejam executados em dispositivos móveis. Para dar suporte à plataforma, um Serviço de Notificação foi implementado e também será descrito neste capítulo.

4.1 Projeto da Plataforma

A plataforma MECM-tel realiza o mapeamento do modelo CM-tel para a tecnologia J2ME sendo que todo o mecanismo de comunicação entre os componentes é realizado com *Web Services* através da exposição e invocação de operações descritas nas interfaces WSDL. A plataforma MECM-tel auxilia o desenvolvimento de aplicações para dispositivos móveis baseadas em componentes CM-tel. Primeiramente, a linguagem Java foi escolhida para implementação dos componentes e *containers* do mapeamento devido aos seguintes fatores:

- independência de plataforma de *hardware* e sistema operacional;
- especificação aberta e disponibilizada pelo JCP (*Java Community Process*) (33);
- padrão largamente utilizado pelos fabricantes de telefones celulares e PDAs;
- apoio e investimento por parte de várias empresas como *Sun Microsystems*, IBM, HP, Palm Inc., etc;
- existência de vários *toolkits* para suporte a *Web Services*;
- acesso à comunidade de desenvolvedores através de fóruns de discussão;
- documentação e tutoriais amplamente disponíveis;
- licença de utilização que permite o uso sem custo.

Foram consideradas outras opções para o mapeamento, tais como o SuperWaba (4), a plataforma .NET (34) e o Qualcomm BREW (3). Porém, nenhuma dessas opções agrega todas as vantagens citadas acima ou as qualidades dessas plataformas são insuficientes para justificar sua escolha.

Uma vez feita a opção pela plataforma Java através do J2ME, um comparativo de soluções para suporte a *Web Services* nessa linguagem foi realizado. Entre as alternativas analisadas havia: o kSOAP da *KObjects* (35), o WSTKMD (*Web Services Toolkit for Mobile Devices*) (36) da IBM e o WSP (*Web Services for Palm*) da *Palm* (37).

A opção escolhida foi o kSOAP pelo fato de possuir código aberto, o que possibilitou a implementação de classes adicionais (*glue code*) para tornar o dispositivo móvel um servidor de *Web Services*. O tamanho, em *bytes*, reduzido do arquivo a ser incluído na aplicação e a maior flexibilidade das bibliotecas constituíram vantagens extras decisivas para escolha desse *toolkit*. Adicionalmente foram implementadas funcionalidades básicas de um servidor HTTP que, juntamente com o kSOAP, torna possível o recebimento e processamento de mensagens SOAP contidas em requisições HTTP.

Dessa maneira, o mapeamento do modelo CM-tel para *Web Services* foi construído utilizando a plataforma J2ME em conjunto com o kSOAP para processamento de mensagens SOAP e a implementação de funções básicas de um servidor HTTP para processamento de requisições.

4.1.1 Geração e Construção de Código

A geração de código para esse mapeamento do modelo CM-tel é realizada em duas etapas. Primeiramente, a especificação UML de componentes e *containers* é transformada em uma representação XML, intermediária à geração de código fonte. A segunda etapa consiste na transformação dessa especificação intermediária no código propriamente dito de componentes e *containers* em uma determinada linguagem de programação, no caso, o J2ME.

Na primeira etapa de transformação, as representações dos elementos do modelo em UML são convertidas para o padrão XMI *(XML Metadata Interchange)* (9) através de ferramentas de projeto de *software* tais como ArgoUML (10) ou Rational Rose (26). Após essa conversão, folhas de estilo XSL *(Extensible Stylesheet Language)* (38) regem a transformação desses documentos XMI em documentos XML. Tais arquivos resultantes são instâncias dos XML *schemas* definidos para componentes e *containers* de acordo com o modelo CM-tel.

Na segunda etapa, agora dependente de tecnologia, as especificações XML dos componentes e *containers* produzidas no primeiro passo sofrem transformações também através de folhas de estilo XSL que resultam no código fonte dos elementos do modelo. Em seguida, o código gerado é compilado e empacotado pela máquina de construção de código.

As portas que fornecem algum tipo de serviço a ser utilizado por outro componente, tais como facetas, propriedades e consumidores de eventos são expostas através de interfaces WSDL geradas a

partir da especificação XML dos componentes. Para cada interface WSDL, a máquina de construção de código cria os respectivos *stubs* através de transformações XSLT (*Extensible Stylesheet Language Transformation*) (25).

A figura 4.1 mostra as etapas do processo de transformação desde a especificação dos componentes e *containers* em UML até os arquivos *jar* e *jad* gerados para instalação.

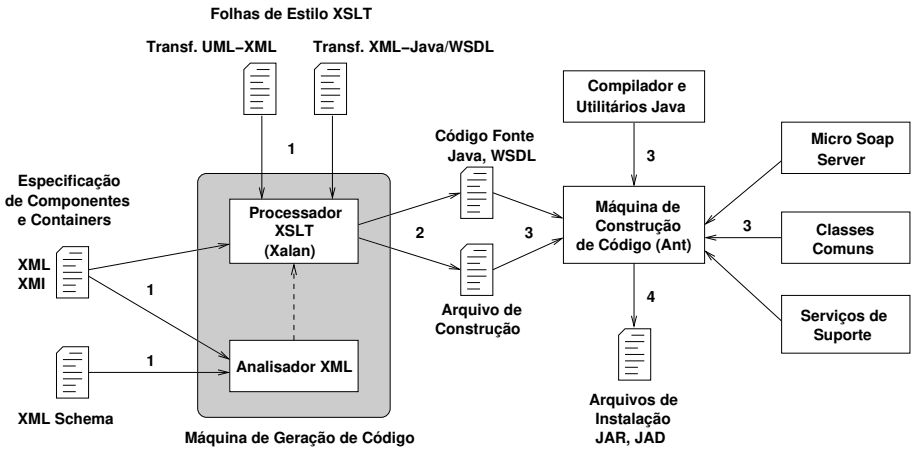

Fig. 4.1: Elementos da plataforma MECM-tel

Inicialmente, a Máquina de Geração de Código recebe como entrada as especificações dos componentes e *containers* contidas em documentos XML ou XMI, os XML *Schemas* que descrevem o formato que esses documentos XML de especificação devem seguir, e as folhas de estilo XSLT com as regras de transformação. Essa etapa é identificada na figura 4.1 com pelo número 1. Caso os arquivos de especificação já estejam em XML (instâncias dos XML *Schemas*), o processo de transformação é iniciado a partir deles.

Como resultado das transformações, a Máquina de Geração de Código gera arquivos contendo código fonte Java (J2ME), arquivos de descrição de serviços (WSDL) e arquivos de construção (*build.xml*) compatíveis com a ferramenta Apache Ant (39), conforme etapa 2 da figura. Esses artefatos servem de entrada para a Máquina de Construção de Código juntamente com o compilador e utilitários Java, classes comuns, o *Micro Soap Server* e Serviços de Suporte, como indicado na etapa 3. As classes comuns são classes da infra-estrutura de componentes que não foram geradas pela Máquina de Geração de Código e que devem ser incluídas durante a compilação e empacotamento dos componentes e *containers*. O *Micro Soap Server* é formado pela biblioteca kSOAP e classes utilitárias responsáveis pelo recebimento, envio e análise das mensagens SOAP. Os Serviços de Suporte são classes auxiliares, tipicamente *stubs*, para prover acesso a serviços como o Serviço de Notificação.

A Máquina de Construção de Código gera os arquivos *jar* de instalação para cada nó especificado

no descritor de distribuição e os respectivos arquivos de descrição *jad*, conforme indicado pela etapa 4 na figura 4.1.

O resultado do mapeamento consiste em um conjunto de classes Java para dispositivos móveis (J2ME) que formam o ambiente de execução dos componentes, os *containers*, bem como os próprios componentes (interfaces equivalentes, propriedades e portas, além de classes utilitárias, classes comuns e de acesso a serviços).

A transformação utiliza o processador XSLT Xalan-Java e a ferramenta Ant é usada para construção.

A plataforma MECM-tel utiliza as especificações MIDP *(Mobile Information Device Profile)* versão 2.0 (40) e CLDC *(Connected Limited Device Configuration)* versão 1.1 (41) do J2ME já disponíveis em inúmeros dispositivos. Para que os serviços disponibilizados nos dispositivos sejam acessados, faz-se necessária a implementação de classes utilitárias que realizam o papel de um servidor HTTP simples capaz de processar requisições e enviar as respostas para os clientes. Com isso, a constrói-se de um pequeno servidor SOAP executando dentro do dispositivo.

O pacote kSOAP permite, de forma transparente, que dispositivos se tornem clientes de *Web Services*. Porém, para que o mesmo dispositivo assuma o papel de servidor, é necessário o desenvolvimento de classes utilitárias responsáveis pela extração do conteúdo da mensagem SOAP e delegação da execução do procedimento remoto para o objeto servidor.

A figura 4.2 mostra os elementos envolvidos no processamento de mensagens SOAP contidas nas requisições HTTP. O código fonte de cada elemento desse processo é gerado automaticamente pela Máquina de Geração de Código a partir da especificação do componente.

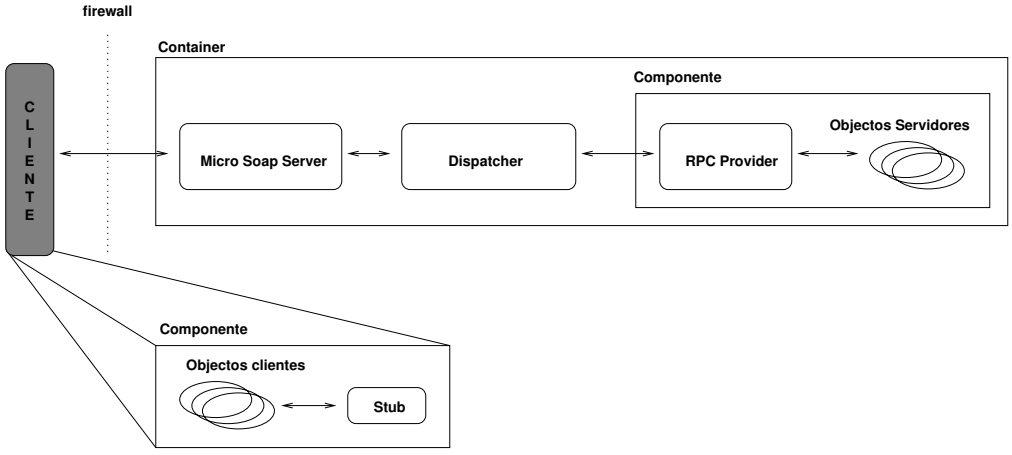

Fig. 4.2: Envio e recebimento de mensagens SOAP

Inicialmente, um cliente envia uma mensagem SOAP que será recebida pelo *Micro Soap Server* que contém o servidor HTTP implementado. Uma vez recebida a mensagem SOAP, são extraídos

o nome do componente que possui o serviço (porta) a ser invocado, a instância desse componente, o nome do método a ser executado e seus parâmetros. Essas informações são repassadas para o elemento *Dispatcher* do *Container* responsável por obter a referência da instância do componente especificado e delegar a chamada do método. Em seguida, o *RPC Provider* obtém a referência do objeto servidor (porta) e invoca o método indicado com os respectivos parâmetros. O valor de retorno do método é repassado de volta até o *Micro Soap Server* que gera uma mensagem SOAP e a envia como resposta da requisição HTTP inicial.

A figura 4.3 mostra um exemplo de requisição SOAP/HTTP responsável pela invocação do serviço de uma faceta.

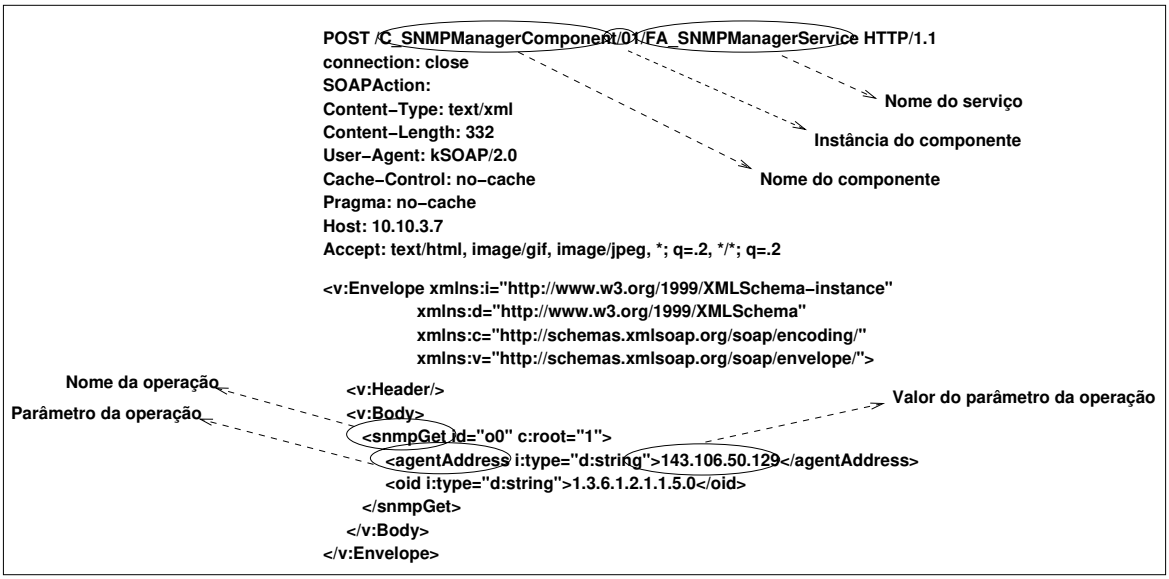

Fig. 4.3: Mensagens SOAP

O *Micro Soap Server* realiza o recebimento dessas mensagens SOAP através de um *socket*. A plataforma possui a limitação de não prover mecanismos de sincronização das requisições podendo acarretar, dependendo da implementação da máquina virtual Java, erros de leitura do *socket*, caso sejam feitas requisições simultâneas.

A utilização de *Web Services* pela plataforma MECM-tel atende aos requisitos propostos no capítulo 3 para um mapeamento que faça uso dessa tecnologia.

4.1.2 Descritores de Montagem e Distribuição

O processo de montagem dos componentes pode ser descrito tanto em UML como em XML, similarmente às especificações de componentes e *containers*. No caso de UML, um diagrama de colaboração deve ser utilizado para descrever o processo de montagem. A figura 4.4 mostra um exemplo

simples de especificação descrevendo a montagem de uma aplicação envolvendo dois componentes.

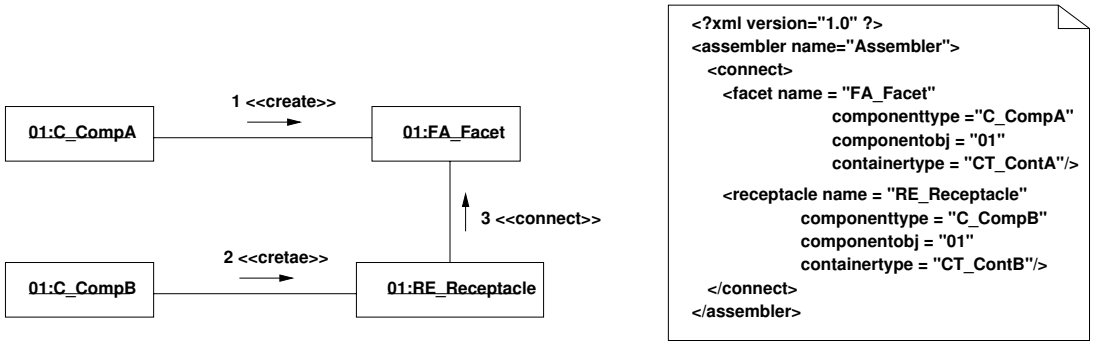

Fig. 4.4: Exemplo de diagrama de montagem

A partir da especificação UML, é gerada a representação XMI do diagrama com o auxílio de uma ferramenta de projeto. Em seguida, é criado o respectivo descritor de montagem também mostrado na figura 4.4. As informações contidas nesse descritor são utilizadas para:

- geração de um componente *Assembler* contendo um faceta responsável pela invocação das operações *connect* e *disconnect* de acordo com o descritor;
- geração de um *container* para hospedar o componente *Assembler*;
- empacotamento dos componentes de forma que as dependências entre eles (portas a serem conectadas) sejam disponibilizadas para os componentes clientes.

O diagrama indica que o componente *C_CompA* possui uma faceta que será armazenada no receptáculo *RE_Receptacle* do componente *C_CompB*. O diagrama também mostra quais as instâncias dos componentes e portas que devem ser conectadas ou desconectadas. Dessa maneira, o componente *Assembler* invoca a operação *connect* do receptáculo para realização da conexão. A ordem em que as conexões são realizadas também é mostrada no diagrama.

A ligação entre as portas presente no diagrama é traduzida no elemento *connect* e indica que o *stub* da faceta *FA_Facet* deve estar contido no pacote do componente que possui o receptáculo *RE_Receptacle*. Ou seja, com a informação de montagem, a Máquina de Construção de Código, através de documentos XML compatíveis com a ferramenta Ant (*build.xml*), disponibiliza as classes de *stubs* dos serviços (portas) para os respectivos pacotes dos componentes clientes.

No caso dos descritores de distribuição, um diagrama UML de distribuição é utilizado para especificar em quais nós da rede cada *container* será instalado e, por conseqüência, cada componente. A figura 4.5 mostra um exemplo simples de especificação para um *container*.

O diagrama indica que o *container CT_Cont* hospedando o componente *C_Comp* deve ser instalado no nó de endereço IP 10.10.1.25. Especifica também que o *container* responde a requisições

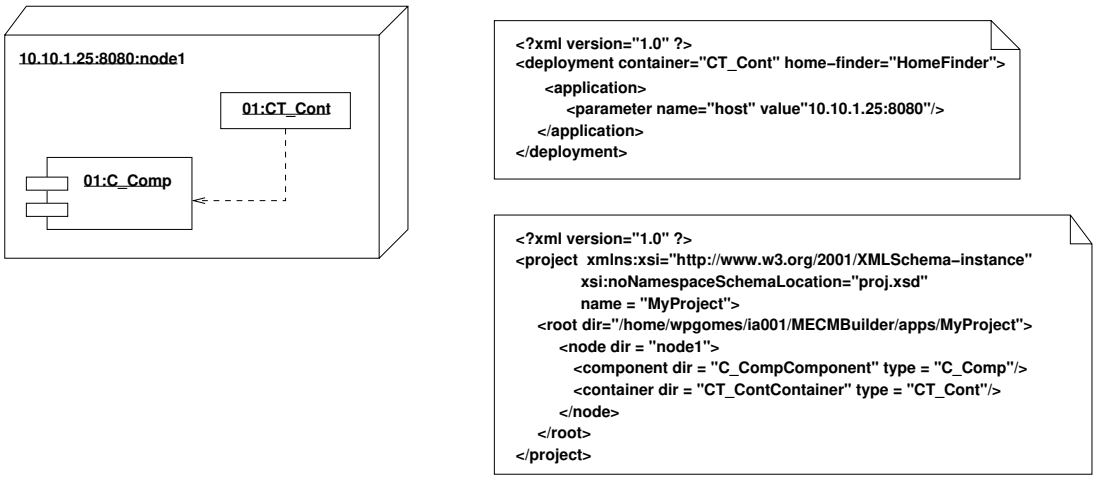

Fig. 4.5: Exemplo de diagrama de distribuição

pela porta 8080. A figura também indica que o nome do arquivo de instalação deve ser *node1*. Todos os componentes e *containers* da aplicação devem estar presentes no diagrama, inclusive o *Assembler*.

Similarmente aos outros diagramas UML de especificação, uma representação XMI intermediária é utilizada para geração do descritor de distribuição e do descritor de projeto, ambos mostrados na figura 4.5. O descritor de projeto estabelece a ordem em que a geração de código acontece e onde os artefatos de *software* serão copiados.

Uma vez gerado e empacotado todo o código da aplicação, o arquivo *jar* executável bem como o seu descritor da aplicação *(jad)* podem ser disponibilizados em um servidor *web* para *download* e instalação automática no dispositivo móvel.

4.2 A Ferramenta MECM-tel Builder

A ferramenta MECM-tel Builder foi desenvolvida utilizando a linguagem Java (J2SE) (1) e implementa a Máquina de Geração de Código e a Máquina de Construção de Código mencionadas anteriormente. A seguir, tem-se uma breve descrição dos três principais casos de uso implementados:

- Criar Projeto: Cria a estrutura de diretórios para determinado projeto onde os arquivos de especificação dos elementos do modelo devem ser copiados;
- Gerar Código: Gera as classes e os arquivos de construção para um projeto (Máquina de Geração de Código);
- Construir Projeto: Compila e empacota os elementos do modelo pertencentes ao projeto (Máquina de Construção de Código).

Entende-se por projeto um conjunto de componentes, *containers*, descritores de montagem e distribuição cuja especificação em XML serve de ponto de partida para geração de código. O código fonte gerado e os arquivos de instalação também fazem parte do projeto. A ferramenta testa o formato dos arquivos de especificação realizando as transformações tanto a partir da representação dos diagramas em XMI quanto a partir da especificação em XML.

A figura 4.6 mostra a interface gráfica da ferramenta MECM-tel Builder onde podem ser vistos os botões através dos quais o usuário inicia os casos de uso. Uma área de texto para exibição de mensagens de informação e erro também está contida na inteface.

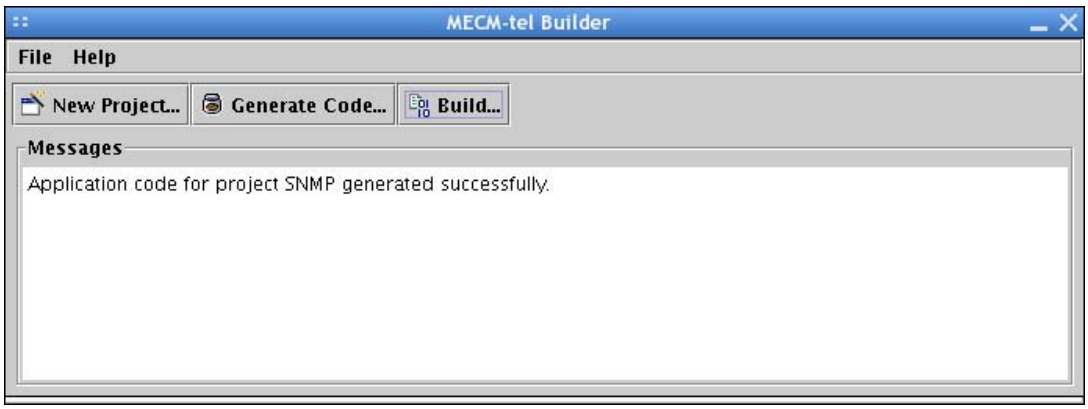

Fig. 4.6: Interface gráfica principal da ferramenta MECM-tel Builder

4.2.1 Geração de Código

A geração dos artefatos pela ferramenta MECM-tel Builder é realizada a partir da leitura das especificações em XML feita por um analisador (*parser*) XML do tipo SAX (27). A medida em que o analisador encontra as *tags* referentes aos elementos da especificação, são invocadas as respectivas transformações. A figura 4.7 ilustra os arquivos gerados para componentes e *containers* a partir das suas especificações em XML.

A presença do elemento *component* no arquivo de especificação dispara as tranformações utilizando as folhas de estilo para esse elemento, como mostrado na figura 4.7. O mesmo ocorre com a faceta através do elemento *facet*. No caso do *container*, as respectivas folhas de estilo são usadas na geração de código da mesma forma que no componente.

No capítulo 5, será apresentado um exemplo de aplicação onde são listados todos os arquivos gerados pela ferramenta MECM-tel Builder.

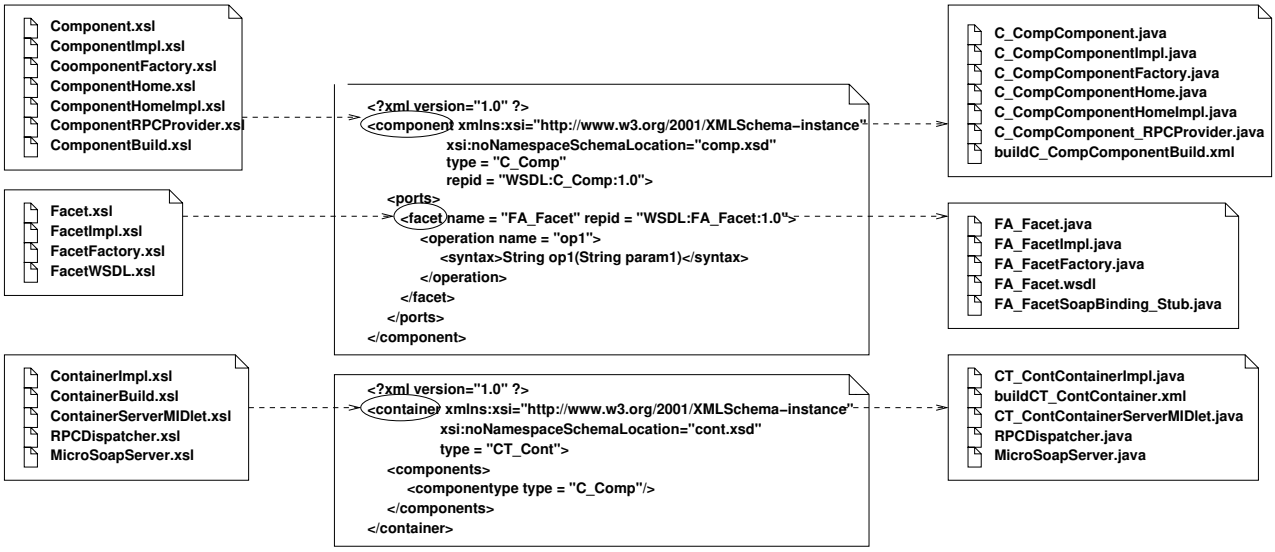

Fig. 4.7: Geração de código a partir da especificação XML de componentes e *containers*

4.2.2 Nomenclatura para o Código Gerado

O mapeamento do modelo CM-tel para *Web Services*/J2ME estipula regras para formação dos nomes dos artefatos de *software* gerados pela ferramenta MECM-tel Builder.

Interfaces WSDL

As interfaces WSDL são geradas para facetas, portas consumidoras de eventos, elementos de configuração e receptáculos. Tais interfaces possuem o mesmo nome do elemento na especificação seguido da extensão *wsdl* como, por exemplo, para a faceta de nome *FA_Facet* é gerado o arquivo *FA_Facet.wsdl*. O nome do serviço disponibilizado por essa faceta recebe o sufixo *Service*. Dessa forma, o nome do serviço para faceta *FA_Facet* seria *FA_FacetService*.

Classes Java

Para as classes Java do componente geradas a partir das especificações XML, tem-se as seguintes regras para formação de nomes:

- Implementações das interfaces têm o sufixo *Impl* adicionado ao nome da interface. Por exemplo, a classe Java contendo a implementação da porta emissora de eventos de nome *EM_EventSource* possui o nome *EM_EventSourceImpl*;
- Implementações da interface equivalente possuem o sufixo *ComponentImpl* acrescentado ao tipo do componente. Por exemplo, o componente de nome *C_Position* possui a implementação de interface equivalente com o nome de *C_PosistionComponentImpl*;

- Implementações das fábricas de componente têm o sufixo *ComponentHomeImpl* acrescentado ao nome do componente;
- Fábricas de componentes, portas e elementos de configuração acrescentam o sufixo *Factory* ao nome da porta ou ao nome do elemento de configuração. Por exemplo, a classe Java da fábrica da porta consumidora de eventos *CO_EventSink* tem o nome de *CO_EventSinkFactory*. No caso da fábrica do componente, é adicionado o sufixo *ComponentFactory*.
- Classes de tratamento de chamadas de procedimento remoto têm o sufixo *_RPCProvider* adicionado ao nome do componente;
- Classes de *stub* das portas, elementos de configuração e receptáculos têm o sufixo *SoapBinding_Stub* acrescentado ao nome da porta ou elemento de configuração. Por exemplo, o stub da faceta *FA_Facet* possui o nome de *FA_FacetSoapBinding_Stub*.

As convenções para formação dos nomes das classes relativas ao *container* são:

- Classe Java que implementa o *container* tem o sufixo *ContainerImpl* acrescentado ao nome do tipo do *container*. Por exemplo, a implementação para o container do tipo *CT_SNMPManager* possui o nome de *CT_SNMPManagerContainerImpl*;
- Classe para execução do *container* através de um *MIDlet* tem o sufixo *ServerMIDlet* acrescentado ao tipo do *container*.

A ferramenta cria uma estrutura de diretórios refletindo os pacotes onde serão geradas as classes Java. Para cada componente e *container*, é criado um pacote cujo nome é formado pelo acréscimo dos sufixos *Component* e *Container* ao nome dos tipos dos componentes e *containers* respectivamente.

Arquivos de Construção e Instalação

Os arquivos de construção do código constituem documentos XML contendo diretivas para a ferramenta Ant. Cada componente e *container* possui o respectivo arquivo de construção. Os nomes desses arquivos são formados pelo acréscimo do prefixo *build_* e do sufixo *Component* e *Container* aos nome dos tipos do componente e *container* respectivamente. Por exemplo, para o componente *C_Position*, é gerado o arquivo de construção *build_C_PositionComponent.xml*.

Nos caso dos arquivos de instalação, são gerados dois arquivos: um *jar* e um *jad*. O primeiro contém todo o código compilado relativo aos componentes e *containers* que serão instalados em determinado nó da rede. Essa informação é obtida através dos descritores de distribuição. O segundo contém informações a respeito desse arquivo *jar*, tais como tamanho em *bytes*, versão, fabricante, nome do MIDlet, permissões, etc.

4.2.3 Empacotamento

A ferramenta MECM-tel Builder obtém as informações necessárias para realizar o empacotamento de *containers* e componentes através dos descritores de projeto e de montagem. O descritor de projeto especifica em quais nós da rede, um conjunto de componentes e *containers* deve ser instalado.

O descritor de montagem especifica quais os pares de portas que devem ser conectadas e seus respectivos componentes e *containers*. Dessa forma, é possível que a máquina de construção de código inclua as dependências (*stubs*) de um componente durante o seu empacotamento. Por exemplo, se o descritor de montagem indica que um receptáculo deve conter a referência de uma faceta, então o *stub* dessa faceta é incluído no pacote do componente que possui o receptáculo.

Durante a geração de código, é criado um arquivo de construção (*build.xml*) para o projeto. Esse *script* é utilizado pela máquina de construção de código para invocar os demais arquivos de construção gerados para cada componente e *container* do projeto.

4.3 Serviço de Notificação

A plataforma MECM-tel utiliza um serviço de notificação baseado em XML para dar suporte às portas consumidoras e emissoras de eventos do modelo de componentes. Esse serviço é responsável pela entrega de eventos no formato de documento XML para as portas consumidoras interessadas em eventos com determinadas características. Para isso, é realizado um filtro através de uma expressão XPath (42) para definir se o evento interessa ou não aos elementos consumidores previamente registrados. A presença de filtros na forma de expressões XPath se torna uma poderosa ferramenta para recuperação de eventos devido a sua sintaxe simples e eficiente.

4.3.1 Arquitetura do Serviço

O Serviço de Notificação foi desenvolvido em Java e possui uma interface *Web Service* com operações de submissão de eventos, registro e desregistro de consumidores. O modelo *push* (43) é utilizado tanto para o emissão quanto para o consumo de eventos. A figura 4.8 mostra os elementos que formam o serviço.

Uma breve descrição dos elementos do serviço é apresentada a seguir:

- Gerente Produtor: Recebe os eventos que chegam através da interface do serviço, adiciona uma marca de tempo (*timestamp*) ao documento XML que descreve o evento e o repassa para o Canal de Eventos;

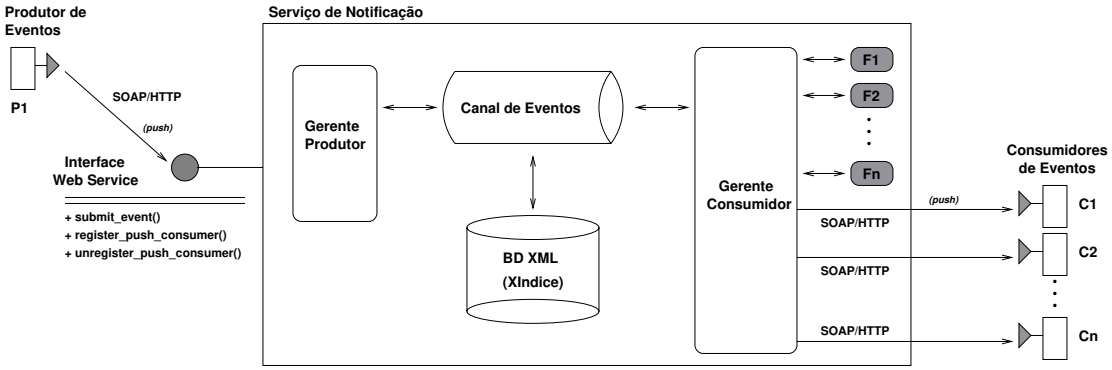

Fig. 4.8: Elementos do Serviço de Notificação

- Canal de Eventos: Verifica a necessidade de persistência do evento através do elemento *timeout* e gerencia a manutenção (consulta, remoção e atualização) dos eventos na Base de Dados. O valor do *timeout* igual a zero indica que o evento é transiente;
- Base de Dados: Armazena os documentos XML que descrevem os eventos e realiza operações através de filtros XPath;
- Gerente Consumidor: Gerencia os filtros relativos a cada elemento consumidor, verifica a validade de cada evento (*timestamp* + *timeout* < *time* do sistema) e realiza a entrega dos eventos através da chamada do serviço da porta consumidora de eventos dos componentes registrados;
- Filtros: Contém a expressão XPath associada à porta consumidora de eventos e o endereço URL do serviço fornecido por essa porta. Os filtros também são armazenados na forma de documentos XML na Base de Dados.

O Serviço de Notificação executa sobre o sistema Apache Axis (44) e a armazenagem dos documentos XML através da Base de Dados XIndice, ambos fornecidos pela *Apache Software Foundation*. Essa implementação do Serviço de Notificação poderá ser utilizada de forma independente da plataforma MECM-tel, bastando que o elemento produtor de eventos faça a invocação das operações do serviço expostas através de Web Services e que os elementos consumidores de eventos sejam capazes de receber os eventos também por meio de Web Services.

4.3.2 Registro de Consumidores de Eventos

O registro de uma porta consumidora de eventos no Serviço de Notificação se dá a partir da invocação da operação *register_push_consumer* exposta na interface *Web Service*. Ao ser instanciada, a implementação da porta invoca a operação do serviço informando o seu identificador (id), o seu endereço URL e a expressão XPath de filtro. O código fonte do componente contendo a chamada

do serviço é gerada automaticamente pela Máquina de Geração de Código. A figura 4.9 mostra a dinâmica do registro de uma porta consumidora de eventos.

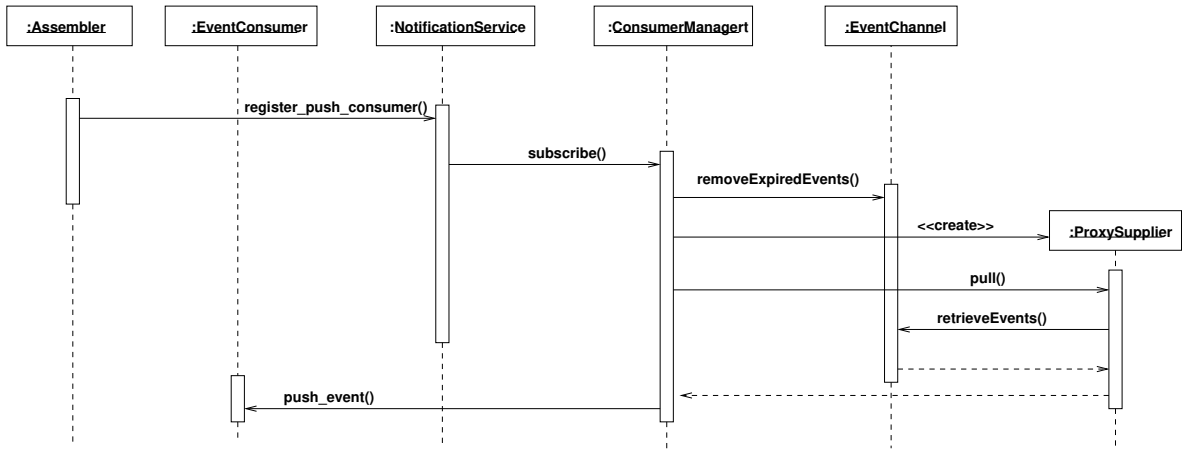

Fig. 4.9: Dinâmica de registro de um consumidor de eventos

Os passos necessários ao registro estão listados a seguir:

1. O elemento de montagem (*Assembler*) invoca a operação de registro na interface do Serviço de Notificação;
2. O Gerente Consumidor é notificado da subscrição de um elemento consumidor;
3. O Gerente Consumidor invoca a operação do Canal de Eventos para remoção dos eventos com prazo de validade vencido;
4. O Gerente Consumidor cria um elemento que representa o consumidor (*ProxySupplier*);
5. O Gerente Consumidor invoca a operação de consulta de eventos;
6. O *ProxySupplier* delega a chamada da operação de consulta ao Canal de Eventos;
7. O Gerente Consumidor invoca a operação *push_event* do elemento consumidor para entrega de eventos.

A expressão de filtro XPath funciona como um comando SQL (*Structured Query Language*) utilizado em Banco de Dados relacionais. Uma expressão XPath retorna subconjuntos dos documentos XML que satisfazem o filtro. Com essa funcionalidade, é possível realizar uma busca por eventos que possuam determinadas características como por exemplo, uma busca pelo nome do emissor, pelo tipo do evento, etc.

4.3.3 Envio de Eventos

O Serviço de Notificação desacopla os elementos produtores e consumidores permitindo que a transmissão de eventos ocorra sem que o primeiro tenha conhecimento do segundo e vice-versa. A

figura 4.10 ilustra o envio de um evento ao Serviço de Notificação e a entrega desse evento aos consumidores de acordo com os filtros.

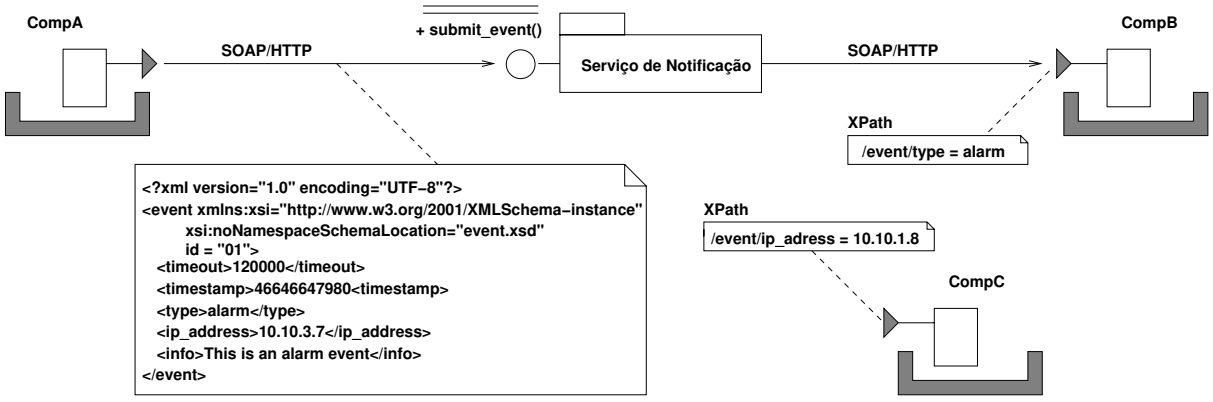

Fig. 4.10: Envio de eventos ao Serviço de Notificação

A seguir, tem-se a seqüência de passos necessários para que um componente consumidor receba um evento do componente produtor de eventos:

1. O componente *CompA* envia o evento ao Serviço de Notificação através de sua interface *Web Service* (operação *submit_event*);
2. O Serviço de Notificação inclui o elemento *timestamp* no documento XML que descreve o evento a fim de registrar o momento de chegada desse evento;
3. O Serviço de Notificação verifica o valor do elemento *timeout* do evento para que o mesmo seja persistido no Banco de Dados ou não (se *timeout* for igual a zero);
4. O Serviço de Notificação consulta todas as portas consumidoras de eventos previamente registradas e realiza uma consulta com cada filtro;
5. O Serviço de Notificação descarta os eventos que estão fora do prazo de validade (*timestamp* + *timeout* < *time* atual);
6. O Serviço de Notificação descarta os eventos que já foram enviados ao(s) consumidor(es);
7. O Serviço de Notificação invoca a operação de recebimento de eventos das portas consumidoras.

O Serviço de Notificação conta com um mecanismo de histórico para evitar que um evento seja enviado mais de uma vez a um determinado consumidor. O serviço registra o *timestamp* do último evento enviado a cada consumidor de forma que somente eventos com o *timestamp* maior do que o registrado no histórico sejam enviados aos consumidores.

Dessa maneira, um elemento consumidor de eventos irá receber somente aqueles eventos que satisfazem seu filtro, que não estejam vencidos e que ainda não foram entregues a esse consumidor.

No exemplo da figura 4.10, o componente *CompB* recebe o evento enviado pelo *CompA*, pois o elemento *type* possui o valor *alarm*. Já o componente *CompC* não recebe o evento, uma vez que o endereço IP contido no elemento *ip_address* é diferente daquele especificado no filtro.

Toda a comunicação entre o Serviço de Notificação e os elementos emissores e consumidores de eventos é feita através de *Web Services*. Desse modo, um elemento consumidor de eventos deve publicar a operação *push_event* através de *Web Sevice* para que seja possível o Serviço de Notificação realizar a entrega do(s) evento(s). O código do componente MECM-tel necessário para isso é gerado automaticamente pela Máquina de Geração de Código.

4.4 Considerações Finais

Este capítulo descreveu a plataforma MECM-tel que constitui um mapeamento do modelo de componentes CM-tel para a tecnologia *Web Services/J2ME*. A plataforma, através da ferramenta MECM-tel Builder, implementa uma Máquina de Geração de Código e uma Máquina de Construção de Código onde os arquivos gerados pela primeira servem de entrada para a segunda.

As regras de formação dos nomes dos arquivos gerados por essa ferramenta também foram descritos bem como os mecanismo de empacotamento e distribuição dos arquivos de instalação criados.

Adicionalmente, o Serviço de Notificação baseado em documentos XML foi descrito juntamente com um exemplo simples de envio de eventos. Esse serviço tem por objetivo dar suporte às portas consumidoras e produtoras de eventos dos componentes da plataforma MECM-tel e se mostrou uma maneira poderosa de propagação de eventos. Graças a utilização de filtros, consumidores determinam quais informações geradas pelos produtores devem ser propagadas até eles pelo canal de eventos.

Dessa forma, a plataforma MECM-tel mostrou-se aderente aos requisitos de um mapeamento para *Web Services* possibilitando a geração de código de componentes e *containers* CM-tel a ser executado em dispositivos móveis.

Capítulo 5

Exemplo de Aplicação

Neste capítulo, será descrito um exemplo de aplicação na área de gerência de redes com o objetivo de ilustrar o uso da plataforma MECM-tel. Essa abordagem inclui a descrição de um cenário para aplicação, sua arquitetura, bem como uma avaliação de desempenho e robustez. Detalhes a respeito da geração automática de código também estão contidos neste capítulo.

5.1 Cenário da Aplicação

Para ilustrar o uso da plataforma MECM-tel, foi desenvolvida uma aplicação de gerência de redes SNMP (*Simple Network Management Protocol*) baseada em XML. Essa abordagem está se tornando uma tendência na área de gerência de redes (45). A idéia principal da aplicação desenvolvida nesse trabalho é permitir que um dispositivo móvel com baixo poder de processamento, como um computador de mão por exemplo, seja capaz de exercer a gerência, através da Internet, de equipamentos de rede que possuam agentes SNMP embarcados.

Com isso, o administrador da rede pode receber em seu terminal móvel notificações sobre problemas ocorridos em determinados equipamentos. É possível também que o administrador consulte informações sobre os dispositivos SNMP e tome ações de configuração a partir de seu computador de mão conectado a Internet.

A Figura 5.1 ilustra os dispositivos da rede que fazem parte da aplicação. Primeiramente, tem-se agentes SNMP executando em diferentes máquinas da rede capazes de gerar *traps* contendo informações sobre algum problema ocorrido. Posteriormente, um gerente SNMP (46) escrito na linguagem Java (J2SE) é executado em outro computador a fim de fazer a gerência direta dos agentes. Um componente MECM-tel, que também pode ser executado na plataforma J2SE, roda na mesma máquina do gerente SNMP. As *traps* geradas são enviadas para o componente MECM-tel que, por sua vez, as converte em eventos XML e os retransmite para outro componente MECM-tel remoto

instalado em um computador de mão.

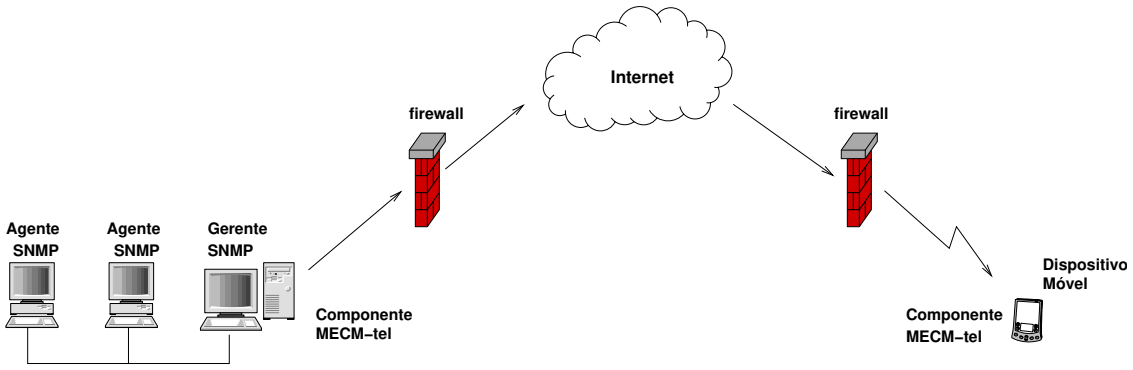

Fig. 5.1: Arquitetura da Aplicação

Dessa forma, o administrador de rede pode requisitar mais informações sobre o equipamento e assim tomar ações corretivas de configuração no dispositivo em questão. O componente MECM-tel instalado no aparelho móvel exerce, dessa forma, a gerência dos equipamentos SNMP.

A seguir, tem-se uma breve descrição dos principais casos de uso da aplicação:

- Receber Notificação: Dispositivo móvel recebe evento(s) correspondente(s) a uma *trap* gerada a partir de um agente SNMP contido no equipamento gerenciado;
- Consultar Parâmetros do Equipamento: Usuário requisita a consulta de parâmetros do equipamento gerenciado, tais como *sysName*, *sysDescription* e *sysLocation*;
- Alterar Parâmetros do Equipamento: Usuário atribui novos valores para parâmetros do equipamento SNMP gerenciado;
- Consultar Comunidade: Usuário requisita o valor da propriedade relativa a comunidade.

Entende-se por comunidade um nome que identifica um grupo de equipamentos gerenciados funcionando como um mecanismo simples de controle de acesso a esses equipamentos.

5.2 Arquitetura da Aplicação

Um componente MECM-tel instalado no mesmo domínio do gerente SNMP é responsável por delegar as operações de gerência e por converter as *traps* geradas pelo agente SNMP em eventos XML. Tais eventos são transmitidos ao Serviço de Notificação pela porta emissora de eventos do componente e recebidos pela porta consumidora do componente instalado no dispositivo móvel remoto. A figura 5.2 mostra a interação mais detalhada entre os elementos da aplicação.

O componente MECM-tel gerente instalado no computador de mão possui dois receptáculos para armazenagem das referências da faceta e do elemento de configuração (propriedades) bem como uma

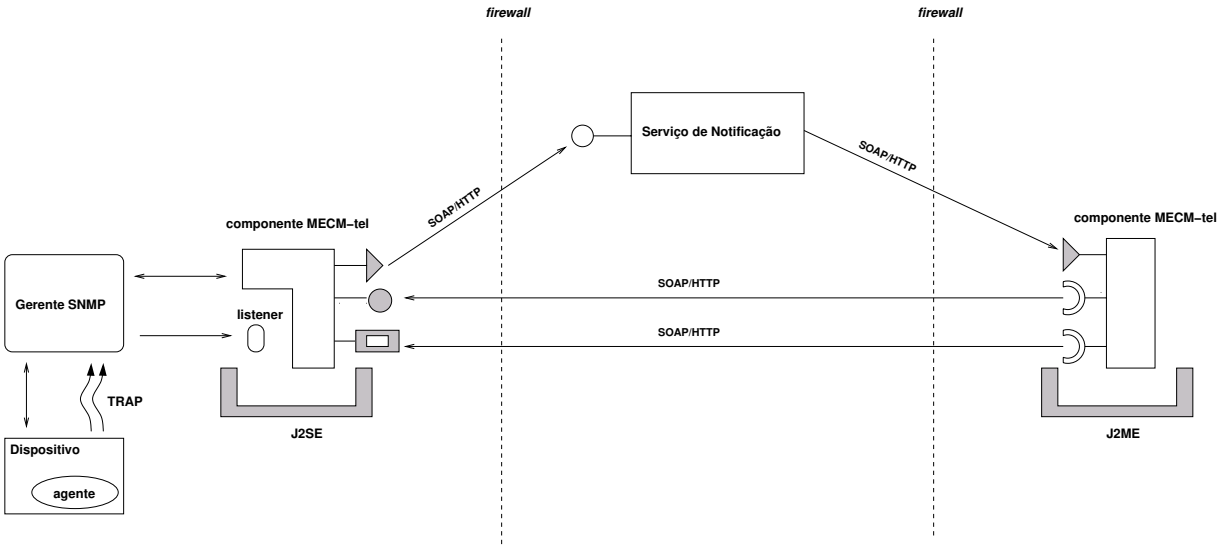

Fig. 5.2: Gerência de equipamentos SNMP através de dispositivos móveis

porta consumidora de eventos. Toda a interação entre os componentes e o Serviço de Notificação e entre os dois componentes é feita através de SOAP/HTTP.

Para execução dessa aplicação ilustrativa de gerência de redes, foram utilizados os seguintes equipamentos:

- Para instalação do *container CT_MicroSNMPManager*, um PDA Palm Tungsten C com 64MB de memória RAM e processador Intel de 400MHz executando o sistema operacional Palm OS 5.2.1;
- Para instalação do *container CT_SNMPManager*, um microcomputador com 256MB de memória RAM e processador Pentium III de 800MHz executando o sistema operacional Linux;
- Para instalação do Serviço de Notificação, um microcomputador com a mesma configuração anterior;
- Para utilização de um agente SNMP, um microcomputador com 256MB de memória RAM e processador Pentium III de 800MHz executando o sistema operacional Windows XP;
- Para conexão sem fio, um ponto de acesso compatível com o padrão IEEE 802.11b.

  Os equipamentos acima fazem parte de uma rede *ethernet* sendo o endereço IP do dispositivo móvel atribuído via DHCP (*Dynamic Host Configuration Protocol*).

5.3 Especificação da Aplicação

A especificação pode ser feita tanto através de diagramas UML, quanto através de documentos XML. No caso da aplicação descrita nesse capítulo, diagramas UML de classes, de colaboração e de distribuição formam as especificações dos componentes e *containers* a partir das quais é gerado código para a aplicação.

5.3.1 Componentes e Containers

A especificação dos componentes e *containers* é feita através de diagramas de classes de acordo com o perfil UML do modelo de componentes CM-tel. A figura 5.3 exibe a especificação UML para o componente *C_SNMPManager*.

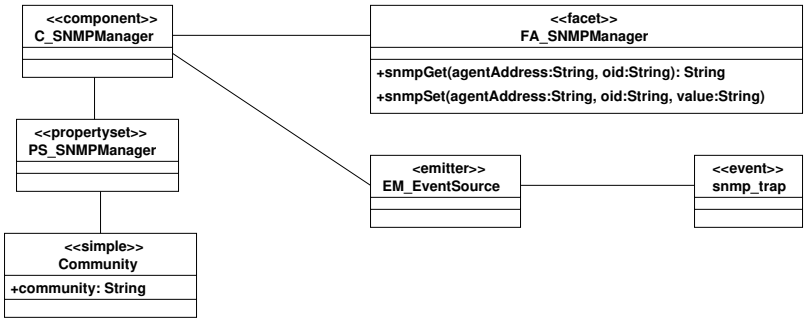

Fig. 5.3: Especificação do componente C_SNMPManager

O *container CT_SNMPManager* mostrado no diagrama da figura 5.4 hospeda o componente *C_SNMPManager*. Esses dois elementos são responsáveis por realizar a interação com o gerente SNMP.

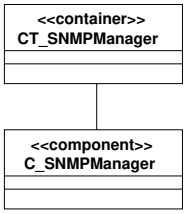

Fig. 5.4: Especificação do container CT_SNMPManager

No lado do dispositivo móvel, é instalado o componente *C_MicroSNMPManager* no *container CT_MicroSNMPManager*. Este componente é responsável pelo recebimento de eventos e invocação das operações remotas de gerência de rede. O filtro (expressão XPath) que será utilizado pela porta consumidora de eventos está especificado na associação entre esta e o evento no diagrama. Em adição

a esses elementos, foram escritas classes de interface gráfica para entrada de dados pelo usuário e exibição das informações a respeito do equipamento gerenciado. As figuras 5.5 e 5.6 exibem as especificações UML do componente *C_MicroSNMPManager* e do *container CT_MicroSNMPManager*.

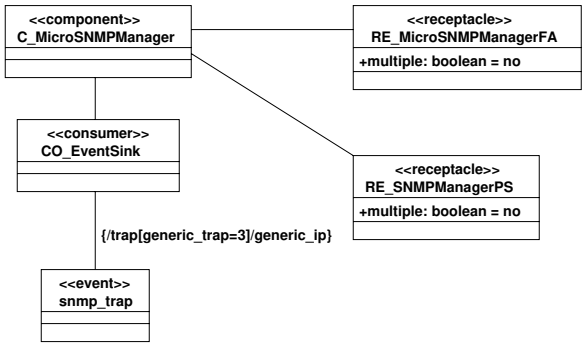

Fig. 5.5: Especificação do componente C_MicroSNMPManager

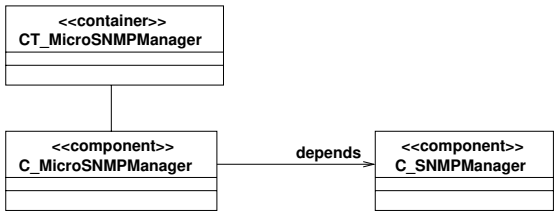

Fig. 5.6: Especificação do container CT_MicroSNMPManager

As especificações em UML descritas acima são exportadas para o formato XMI pela ferramenta de projeto ArgoUML.

5.3.2 Distribuição e Montagem

A especificação da aplicação referente a distribuição dos componentes e *containers* entre os nós da rede é descrita através do diagrama UML de colaboração mostrado na figura 5.7.

O diagrama especifica que o *container CT_MicroSNMPManager* juntamente com o componente *C_MicroSNMPManager* devem ser instalados no elemento da rede cujo endereço IP é 10.10.1.26 (computador de mão). Da mesma forma, o *container CT_SNMPManager* e o componente *C_SNMPManager* devem ser instalados no endereço 10.10.3.7. O componente e *container* responsáveis pela montagem (*Assembler*), por sua vez, serão instalados no nó da rede cujo endereço é 10.10.3.6. O diagrama também indica qual o nó da rede em que o Serviço de Notificação está sendo executado.

As informações para montagem da aplicação são especificadas por um diagrama UML de colaboração, como mostrado na figura 5.8.

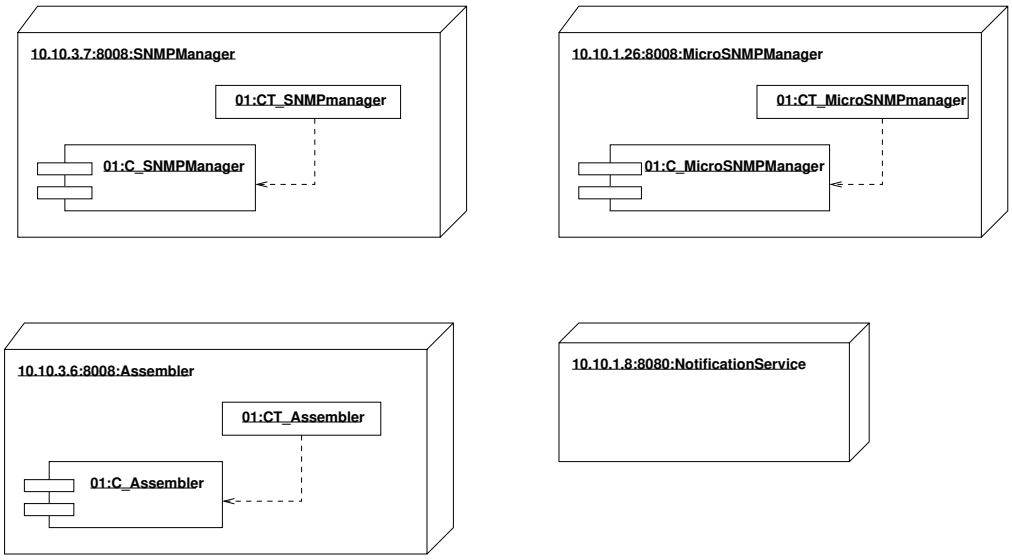

Fig. 5.7: Especificação da distribuição da aplicação

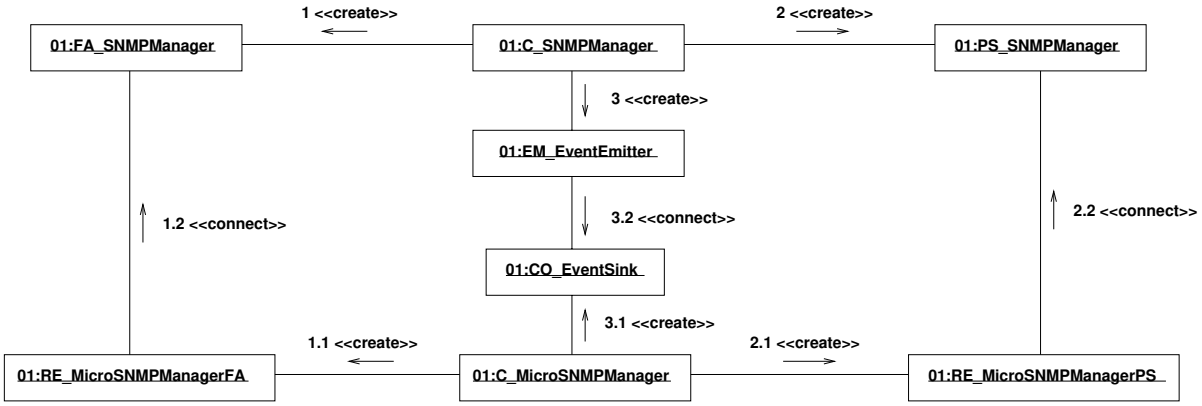

Fig. 5.8: Especificação da montagem da aplicação

O receptáculo *RE_MicroSNMPManagerFA* pertencente ao componente *C_MicroSNMPManager* guarda a referência da faceta *FA_SNMPManager* do componente *C_SNMPManager*. Da mesma forma, o receptáculo *RE_MicroSNMPManagerPS* pertencente ao componente *C_MicroSNMPManager* guarda a referência da porta de propriedades *PS_SNMPManager* do componente *C_SNMPManager*. O diagrama também indica que a porta emissora de eventos *EM_EventEmitter* deve ser conectada à porta consumidora de eventos *CO_EventSink*.

5.4 Geração de Código

Baseado nas especificações mostradas acima, foram gerados os artefatos de *software* da plataforma MECM-tel correspondentes aos elementos da aplicação. Para cada diagrama da especificação, serão

listados os arquivos resultantes das transformações realizadas pela ferramenta MECM-tel Builder.

Componentes

A especificação do componente *C_MicroSNMPManager* mostrada na figura 5.3 é transformada na representação XML intermediária de acordo com a figura 5.9.

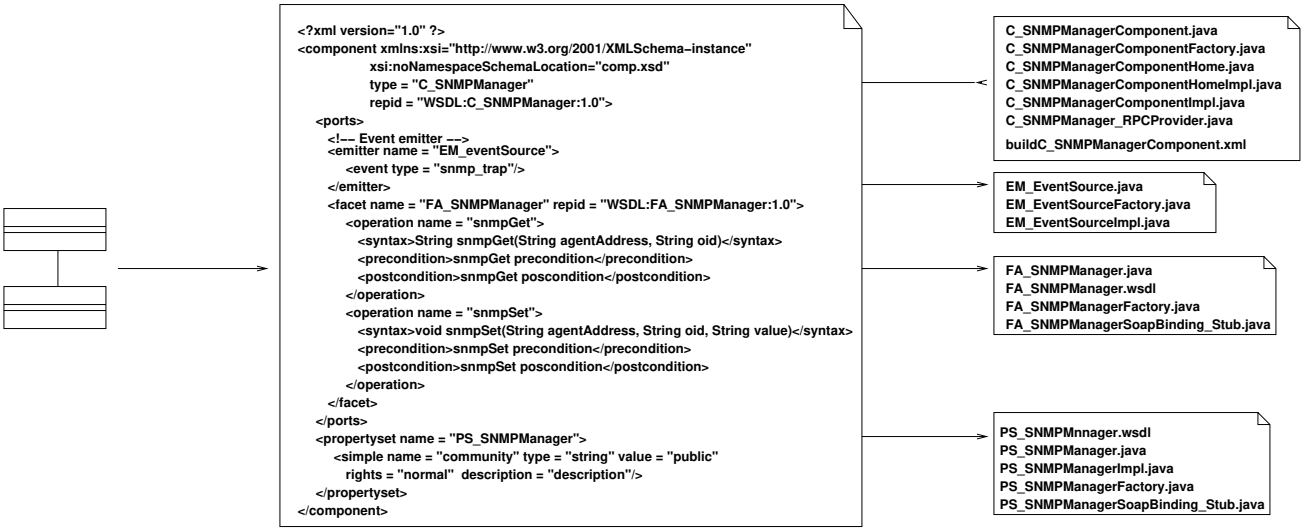

Fig. 5.9: Artefatos gerados para o componente C_SNMPManager

A partir dessa representação, são gerados os artefatos de *software* que constituem o componente como descrito nas seções 4.2.1 e 4.2.2. A figura 5.10 mostra o mesmo processo para o componente *C_MicroSNMPManager*.

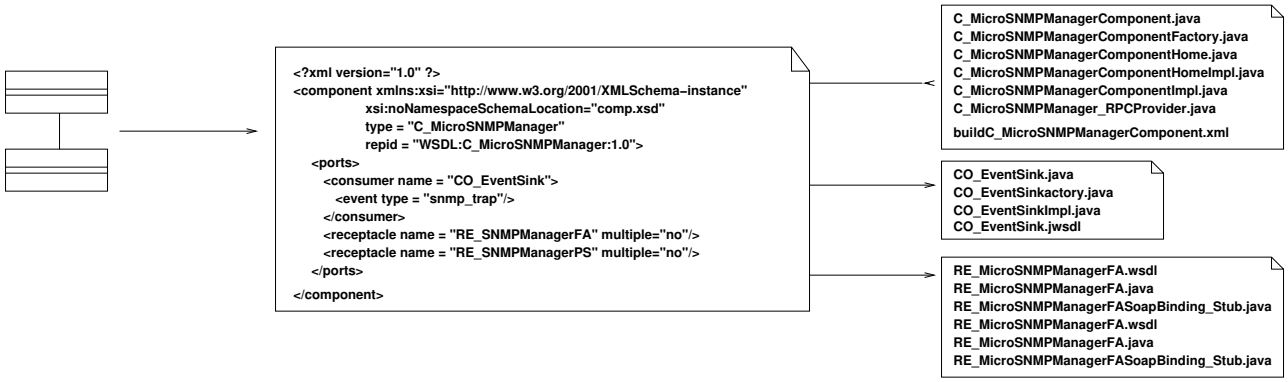

Fig. 5.10: Artefatos gerados para o componente C_MicroSNMPManager

Containers

Os artefatos de *software* gerados a partir das especificações dos *containers* são listados na figura 5.11 e 5.12. Os artefatos gerados são aqueles discutidos nas seções 4.2.1 e 4.2.2.

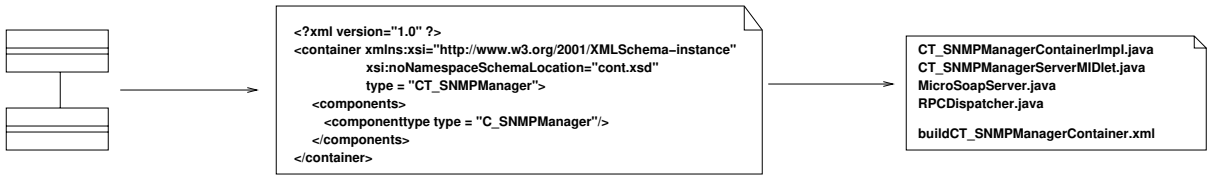

Fig. 5.11: Artefatos gerados para o container CT_SNMPManager

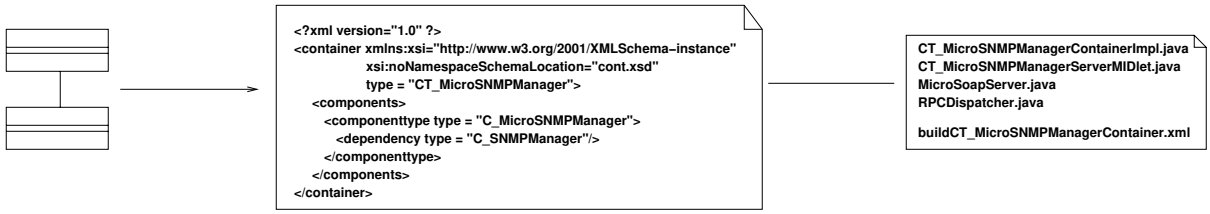

Fig. 5.12: Artefatos gerados para o container CT_MicroSNMPManager

Descritores de Montagem

O diagrama de colaboração que descreve o processo de montagem da aplicação também dá origem a um documento XML intermediário (descritor de montagem), como mostrado na figura 5.13.

As informações contidas no descritor são utilizadas durante geração do componente *C_Assembler* para que as portas especificadas sejam conectadas. Nesse caso, o componente *C_Assembler* conterá as invocações das operações de *connect* dos receptáculos *RE_MicroSNMPManagerPS* e *RE_MicroSNMPManagerFA* bem como das operações de registro da porta consumidora de eventos no Serviço de Notificação.

O descritor de montagem informa à Máquina de Construção de Código quais são as dependências dos componentes para que as interfaces e os *stubs* de portas servidoras sejam disponibilizados nos pacotes dos componentes clientes. Por exemplo, o descritor mostrado na figura 5.13 indica que existe uma conexão a ser feita entre a porta *FA_SNMPManager.java* e o receptáculo *RE_MicroSNMPManagerFA*. Dessa forma, os arquivos *.class* resultado da compilação dos arquivos *FA_SNMPManager.java*, *FA_SNMPManagerSoapBinding_Stub.java*, *PS_SNMPManager.java* e *FA_SNMPManagerSoapBinding_Stub.java* do componente *C_SNMPManager* devem ser incluídos na construção do arquivo *jar* de instalação do *componente C_MicroSNMPManager* durante o processo de empacotamento.

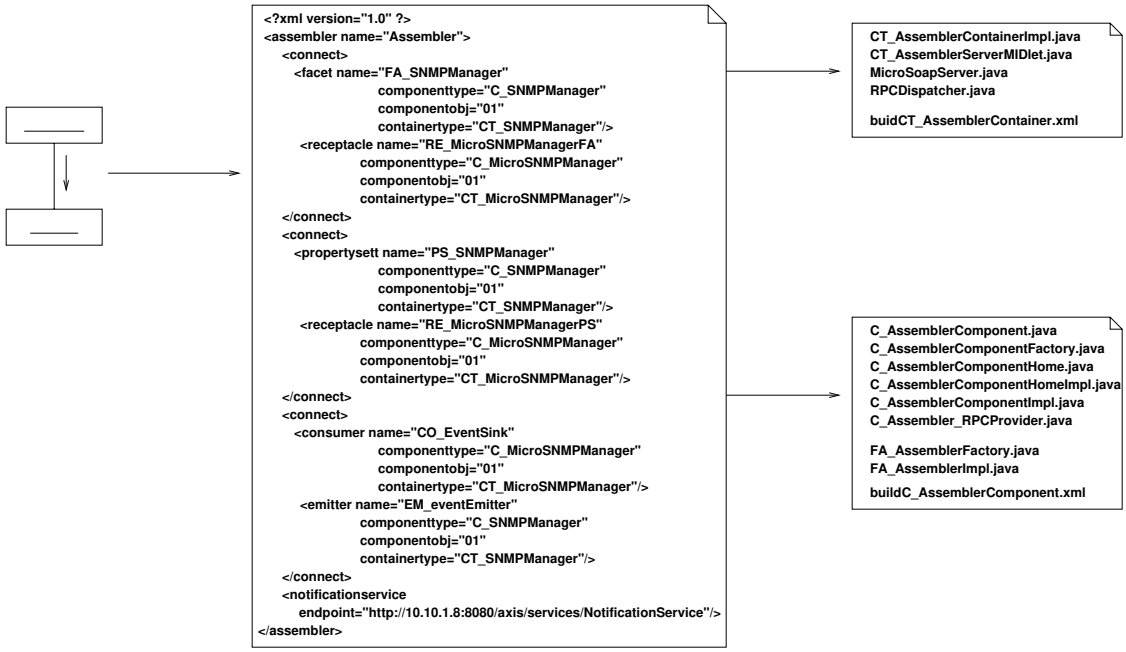

Fig. 5.13: Artefatos gerados a partir do descritor de montagem

Descritores de Distribuição

A partir do diagrama da figura 5.7, são gerados o descritor de projeto SNMP e os descritores de distribuição para cada *container*, conforme descrito na seção 4.1.2.

As informações contidas nesses descritores são utilizadas pela Máquina de Geração de Código a fim de indicar quais são os componentes e *containers* que fazem parte da aplicação. Adicionalmente, os endereços IP dos nós da rede especificados nos descritores de distribuição são utilizados na geração dos *stubs* dos serviços oferecidos pelas portas do componente. Por exemplo, todas as portas pertencentes ao componente *C_SNMPManager* e que oferecem serviços possuem a *string* 10.10.3.7:8080 como parte do endereço URL desse serviço.

O fato dos endereços IP dos componentes serem especificados antecipadamente no diagrama de distribuição pode se tornar uma restrição devido a obrigatoriedade da instalação dos componentes nestes nós. Porém, poderia ser gerado um componente *Assembler* a partir de um outro diagrama de distribuição contendo endereços atualizados dos componentes e *containers* para que as conexões sejam realizadas utilizando esses novos endereços.

Outra forma de tornar dinâmicos os endereços dos componentes seria por meio da consulta a um servidor UDDI contendo a descrição dos serviços disponibilizados pelas portas dos componentes. De posse dessa descrição (interfaces WSDL), os componentes clientes seriam capazes de recuperar o endereço do serviço requerido. Essa abordagem não foi implementada pela ferramenta MECM-tel Builder.

5.5 Execução da Aplicação

A execução da aplicação conforme o cenário descrito na seção 5.1 deve seguir a seguinte seqüência de passos:

1. Instalação do Serviço de Notificação em um servidor de aplicações no endereço especificado no diagrama de distribuição;
2. Instalação da Base de Dados XIndice na mesma máquina em que o Serviço de Notificação é executado;
3. Instalação (cópia) dos arquivos *SNMPManager.jar* e *SNMPManager.jad* para o nó especificado no diagrama de distribuição;
4. Instalação dos arquivos *MicroSNMPmanager.jar* e *MicroSNMPManager.jad* no computador de mão através da sincronização via porta USB do arquivo *prc* correspondente ou através da transferência a partir de um servidor *web* (*Over the Air - OTA*) bastando que se forneça o endereço URL em um aplicativo específico disponibilizado juntamente com a máquina virtual Java.
5. Instalação (cópia) dos arquivos *Assembler.jar* e *Assembler.jad* para o nó especificado no diagrama de distribuição;

Uma vez realizados esses passos, os elementos da aplicação devem ser inicializados na mesma ordem descrita acima. A execução do *container* do arquivo *SNMPManager.jar* é realizada através do pacote ME4SE (47) para execução de MIDlets em ambiente J2SE.

Após a inicialização de todos os elementos da aplicação, uma classe utilitária (*AssemblerTrigger*) invoca a operação exposta na faceta do componente *C_Assembler* que, por sua vez, inicia a montagem da aplicação.

5.5.1 Resultados

Os resultados obtidos durante a execução dos casos de uso da aplicação são descritos a seguir:

1. O caso de uso Receber Notificação foi executado utilizando a geração de *traps* pelo agente SNMP presente no Windows XP. Isso acontece ao se desconectar o cabo rede da máquina ou quando se realiza uma consulta a parâmetros do agente SNMP contendo erros de sintaxe. Dessa forma, a porta consumidora de eventos recebe apenas os eventos correspondentes ao primeiro tipo de *trap*, pois a expressão XPath de filtro (*/trap[generic_trap=3]/generic_ip*) dessa porta especifica tal comportamento;

2. O caso de uso Consultar Parâmetros do Equipamento foi executado para os parâmetros *sysName*, *sysDescription*, *sysLocation* e *sysContact*;
3. No caso de uso Alterar Parâmetros do Equipamento, foram utilizados valores de teste para os mesmos parâmetros listados acima;
4. O caso de uso Consultar Comunidade foi executado obtendo-se o valor *public*.

A Figura 5.14 mostra o dispositivo móvel durante a execução da aplicação realizando a alteração do parâmetro *sysName* e recebendo eventos.

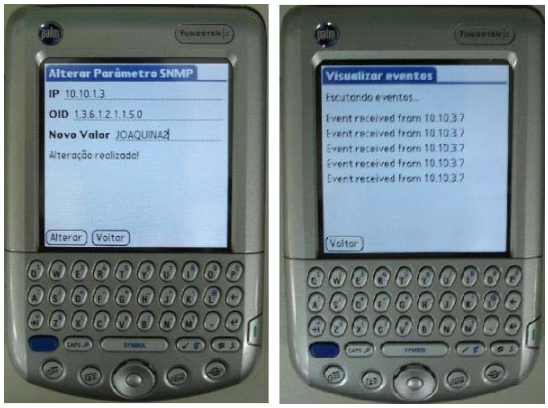

Fig. 5.14: Dispositivo móvel durante a execução da aplicação

5.6 Avaliação

A aplicação foi avaliada quanto a fatores de performance e quantidade de código gerado automaticamente. A tabela 5.1 mostra as medidas do tempo necessário para execução de uma operação na porta do componente remoto, ou seja, o tempo de duração dos casos de uso listados.

Casos de Uso	*Média* (ms)	*Mediana* (ms)	*Desvio-Padrão*	*Número de Medidas*
Consultar Parâmetro	1185,55	1180	26,93	45
Alterar Parâmetro	1192,50	1190	26,78	48
Consultar Comunidade	1197,71	1200	23,90	48

Tab. 5.1: Medidas de performance para invocação de operações remotas

Esses valores foram calculados desconsiderando os pontos extremos (*outliers*), identificados na amostra. Considerando o fato de os valores das médias e das medianas terem ficado próximos, além do valor do desvio-padrão ser relativamente baixo, pode-se concluir que a média é uma medida adequada para representar a amostra.

Um outro teste foi realizado a fim de medir a capacidade da aplicação em responder a requisições. Para isso, a aplicação deveria tratar corretamente a chegada de um determinado número de *traps* por segundo, sendo que a quantidade máxima alcançada foi de três *traps* por segundo. Para razões superiores a esta, o micro servidor SOAP desconsiderava as requisições adicionais e exibia mensagens de erro.

Essa limitação constitui um ponto fraco da plataforma, pois aplicações que necessitam de respostas mais rápidas podem se tornar inconsistentes. Porém, levando-se em consideração as limitações do dispositivo móvel e o *overhead* causado pelo processamento de mensagens, pode-se considerar que o resultado geral foi satisfatório, uma vez que uma vasta gama de aplicações pode ser desenvolvida com esses resultados de performance.

A percentagem de código gerado automaticamente pela ferramenta MECM-tel Builder, para essa aplicação de exemplo, foi de cerca de 91%. Para realização desse cálculo, foram considerados o número de linhas de código gerado e o número de linhas efetivamente escritas pelo programador. Foi utilizado o comando *wc (word count)* do Linux para contagem de linhas dos arquivos com extensão *.xml*, *.java*, *.wsdl*, *.jad* e *.MF*.

Essa percentagem de código gerado certamente varia de acordo com a complexidade e lógica de negócios da aplicação. Quanto maiores e mais complexos são os métodos que implementam a lógica de negócios, mais linhas de código o programador deverá escrever. No caso da aplicação de exemplo, pode-se considerar também satisfatória a quantidade de código gerado.

5.7 Considerações Finais

Este capítulo apresentou um exemplo de aplicação na área de gerência de redes SNMP utilizando componentes MECM-tel e inteiramente especificada através de diagramas UML. A partir dessa especificação descrita no formato XMI, a ferramenta MECM-tel Builder foi responsável pela geração da maior parte do código da aplicação, cerca de 91%.

Um componente específico de montagem (*Assembler*), também gerado automaticamente, realizou a conexão entre as portas dos demais componentes da aplicação de acordo com a especificação dos diagramas UML. Uma vez montada a aplicação, operações de consulta e alteração de parâmetros dos equipamentos bem como o recebimento de eventos foram realizados a partir do computador de mão. Por último, foi feita uma avaliação dos resultados obtidos quanto à performance e à quantidade de código gerado automaticamente.

A aplicação desenvolvida ilustrou o uso de várias funcionalidades da infra-estrutura proposta nesse trabalho para o desenvolvimento de *software* baseado em componentes para dispositivos móveis.

Capítulo 6

Conclusões

Este capítulo apresenta uma revisão do trabalho realizado através de uma retrospectiva do que foi desenvolvido, das contribuições dadas, bem como dos resultados alcançados. O capítulo contém ainda sugestões de trabalhos futuros que poderão ser realizados a fim de melhorar tanto infra-estrutura proposta como o próprio modelo de componentes CM-tel.

6.1 Retrospectiva

O presente trabalho apresentou uma infra-estrutura para auxílio ao desenvolvimento de *software* orientado a componentes para dispositivos móveis, como por exemplo telefones celulares e computadores de mão. O desenvolvimento de aplicações baseadas no modelo de componentes CM-tel tornou-se possível também nesses dispositivos através do mapeamento do modelo para J2ME/*Web Services* neutro em termos de tecnologia (linguagens de programação, sistemas operacionais, protocolos de rede, etc.).

A fim de explorar as vantagens de mobilidade e baixo custo dos dispositivos móveis foi proposta uma infra-estrututa para desenvolvimento de *software* orientado a componentes que transforma aparelhos como telefones celulares e computadores de mão em elementos principais das aplicações que seguem o modelo CM-tel. Isso torna possível que os dispositivos assumam um papel de destaque na aplicação e não o de meros clientes de um determinado servidor que possua maior poder de processamento.

A primeira etapa do trabalho consistiu em transformar as especificações UML dos componentes e *containers* em suas respectivas representações XML intermediárias através da aplicação de transformações XSLT nos diagramas no formato XMI. Isso possibilitou que toda a especificação dos componentes e *containers* fosse feita através de diagramas UML com o auxílio de uma ferramenta de projeto como o ArgoUML.

Posteriormente, foi realizado o mapeamento propriamente dito do modelo CM-tel para J2ME/*Web Services* com a implementação da ferramenta MECM-tel Builder responsável pela geração automática de cerca de 91% do código da aplicação de exemplo. Essa geração automática diminui a probabilidade de erros no código, possibilita a aplicação de padrões de projeto de maneira mais eficiente e aumenta o reuso de código. A plataforma apresentou algumas desvantagens inerentes ao ambiente computacional restrito dos dipositivos móveis tais como limitações de performance, limitação do tamanho em *bytes* das aplicações e bibliotecas Java reduzidas. Foram indicadas também algumas desvantagens da plataforma em si, como por exemplo, a ausência de mecanismos de sincronização de requisições e a baixa perfomance no tratamento de eventos.

A infra-estrutura proposta contou também com um Serviço de Notificação baseado em documentos XML implementado para dar suporte as portas produtoras e consumidoras de eventos do modelo CM-tel.

Para ilustrar as funcionalidades da plataforma, foi desenvolvida uma aplicação de gerência de redes SNMP na qual o dispositivo móvel faz o papel de gerente SNMP e de consumidor de eventos que descrevem *traps*. Aplicações mais complexas em gerência de rede ou laboratórios virtuais (8) também poderiam fazer uso de componentes CM-tel executados em dispositivos móveis. Essa possibilidade torna os sistemas mais flexíveis, além de trazer os benefícios da mobilidade e do baixo custo.

6.2 Trabalhos Futuros

A seguir, são propostos alguns trabalhos futuros que têm como objetivo agregar valor ao modelo de componentes CM-tel e aperfeiçoar a infra-estrutura proposta nesta dissertação:

- Implementação do mapeamento do modelo CM-tel para tecnologia J2SE/*Web Services*;
- Integração das ferramentas CCM-tel Builder e MECM-tel Builder para que por meio de uma única ferramenta, seja possível a geração de código para diferentes tecnologias como Java-/CORBA e J2ME/*Web Services*;
- Implementação de um Serviço de Controle de Fluxo de Mídia para dar suporte as portas produtoras e consumidoras de fluxo do mapeamento J2ME/*Web Services*;
- Geração de código para suporte à qualidade de serviço (QoS) das portas produtoras e consumidoras de fluxo.
- Utilização do Serviço de Registro UDDI (*Universal Description, Discovery and Integration*) para obtenção da descrição dos serviços providos e consumidos pelos componentes.

6.3 Contribuições

O trabalho desenvolvido incrementa a proposta mais ampla do modelo de componente CM-tel reafirmando sua independência de tecnologia e tornando-o mais flexível através da utilização dos dispositivos móveis.

Aplicações baseadas em componentes para dispositivos móveis poderão ser desenvolvidas mais facilmente, em menos tempo, com menos erros e mais reuso de código e de projeto. Reuso este que proporcionado tanto por meio da reutilização direta dos componentes previamente gerados quanto através dos templates usados para geração automática do código. Adicionalmente, a implementação realizada do Serviço de Notificação poderá ser utilizada por diferentes tipos de aplicações e não só por aquelas baseadas no modelo de componentes CM-tel.

Como contribuição em trabalhos científicos, a implementação da transformação das especificações dos componentes e *containers* CM-tel no formato XMI para o formato XML intermediário possibilitou a co-autoria do artigo intitulado "CCM-tel - uma Plataforma para Aplicações Telemáticas e Ubíquas" publicado no 22º Simpósio Brasileiro de Redes de Computadores - SBRC'2004.

De maneira geral, o trabalho realizado atingiu os objetivos propostos como foi constatado no capítulo 5 com a aplicação de exemplo. Naturalmente, melhorias ainda precisam ser feitas em versões futuras da ferramenta MECM-tel Builder e do Serviço de Notificação. Por exemplo, a inclusão de condições de contorno para geração de código a partir de arquivos de especificação inconsistentes. No caso do Serviço de Notificação, melhorias no mecanismo de histórico de envio e recebimento de eventos, suporte ao registro de produtores de eventos bem como mecanismos de controle de acesso ao serviço poderiam ser adicionados. Tanto para o Serviço de Notificação quanto para a ferramenta MECM-tel Builder, é necessária a realização de testes adicionais visando a correção de erros em versões futuras.

Referências Bibliográficas

[1] Gary ComeU Cay Horstmanm. *Core Java* 2, *Volume I-Fundamentais.* Prentice HaU, 2004.

[2] Kim Topley. *J2ME in a Nutshell.* O'Rei11y & Associates, 2002.

[3] QUALCOMM Incorporated. *BREW Distribution System (BDS) Overview,* Setembro 2004. http://brew.qualcomm.com/brew/en/img/about/pdf/bds.pdf.

[4] SuperWaba. *SuperWaba,* August 2004. http://www.superwaba.com.br.

[5] Eliane Guimarães. *Um Modelo de Componentes para Aplicações Telemáticas e Ubíquas.* PhD thesis, Universidade Estadual de Campinas, 2004. Faculdade de Engenharia Elétrica e de Computação.

[6] E.G. Guimarães, E. Cardozo, M.F. Magalhães, W.P. Gomes, R.P. Pinto, L.F. Faina. "CCM-teluma Plataforma para Aplicações Telemáticas e Ubíquas". In *22° Simpósio Brasileiro de Redes de Computadores - SBRC'2004,* Gramado, Rio Grande do Sul, Maio 2004.

[7] Carlos Alexandre Miglinski. Uma ferramenta para suporte ao desenvolvimento de software orientado a componente. Master's thesis, Universidade Estadual de Campinas, 2003.

[8] E. Guimarães, A. Maffeis, R. Pinto, C. Miglinski, E. Cardozo, M. Bergerman, M. Magalhães. "REAL - A Virtual Laboratory Built From Software Components". *Proceedings of the IEEE,* 9l(3):44~8, Mar 2003.

[9] Steven Brodsky Tim Grose. *Mastering XMl: Java Programming with the XMl toolkit, XML, and UML.* John Wiley Professional, 2002.

[10] Tigris.org: Open Source Software Engineering. *Argo UML,* August 2004. `http: / / argouml.tigris.org/`.

[11] Daniel S. Haischt. *The iChilli mobile J2EE platform - A J2EE compliant, distributed surrogate for mobile devices.* PhD thesis, University of Applied Science, Reutlingen, 2003.

[12] Rod Johnson. *Expert one-on-one J2EE Design and Development.* John Wiley Consumer, 2002.

[13] Richard Monson-Haefel. *Enterprise JavaBeans.* O'Reilly & Associates, 2001.

[14] Apache Software Foundation. *Jakarta Avalon,* August 2004. http: / / avalon. apache . org/.

[15] Codehaus. *OpenEJB Container System,* August 2004. http://www . openej b. org /.

[16] Stefan Berger, Scott McFaddin, Chandra Narayanaswami, Mandayam Raghunath. "Web Services on Mobile Devices - Implementation and Experience". In *Proceedings ofthe Fifth IEEE Workshop on Mobile Computing Systems & Applications,* IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY, 10532,2003.

[17] SANKHYA Technologies. *SANKHYA Varadhi - The Digital Bridge,* August 2004. ht tp : //www.sankhya.com/info/varadhi.html.

[18] Doug Schmidt. *Tutorialon the CORBA Component Model (CCM),* August 2004. http: //www.cs.wustl.edu/~schmidt/tutorials-corba.html.

[19] G. Heineman, W. Council. *Component-Based Software Engineering - Putting the Pieces To gether.* Addison-Wesley, 2001.

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

[21] J. Daniels J. Cheesman. *UML Components - A Simple Process for Specifying Component-Based Software.* Addison-Wesley,2001.

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

[23] Object Management Group. *CORBA/IIOP Specification,* August 2004. http://www . omg. org/technology/documents/formal/corba_iiop.htm.

[24] Object Management Group. *CORBA Component Model, v3.0,* August 2004. http://www . omg.org/technology/documents/formal/components.htm.

[25] Michael Fitzgerald. *Learning XSLT.* O'Reilly & Associates, 2003.

[26] Terry Quatrani. *Visual Modeling with Rational Rose 2002 and UML.* Addison Wesley, 2002.

[27] The Apache Software Foundation. *Xalan-Java v2.* 6. *O,* Setembro 2004. http://xml. apache.org/xalan-j/.

[28] Eric Newcomer. *Understanding Web Services - XML, SOAp, UDDI and WSDL.* Addison Wesley, 2002.

[29] Joe Clabby. *Web Services Explained - Solutions and Applications for the Real World.* Prentice Hall,2003.

[30] Robert Englander. *Java and SOAP.* O'Reilly & Associates, 2002.

[31] Aaron E. Walsh. *UDDI, SOAP and WSDL - The Web Services Specification Reference Book.* Prentice Hall, 2002.

[32] William Grosso. *Java RM/.* O'Reilly & Associates, 2001.

[33] Community Development of Java Technology Specifications, Setembro 2004. http: / / j cp . org/en/home/index.

[34] Niel M. Bomstein. *.NET and XML.* O'Reilly & Associates, 2003.

[35] KObjects. *kSOAP* 2, Setembro 2004. http://ksoap . org /.

[36] IBM. *Web Services Tool Kit for Mobile Devices,* Setembro 2004. http://www . alphaworks.ibm.com/tech/wstkmd/.

[37] PalmOne. *Web Services for Palm Handhelds,* Setembro 2004. http://pluggedin. palmone.com/regac/pluggedin/WebServices.jsp.

[38] Kurt Cagle. *Professional XSL.* Wrox,2001.

[39] Jessy Tilly. *Ant - The De.finitive Guide.* O'Reilly & Associates, 2002.

[40] Sun Microsystems Inc. *Mobile Information Device Profile (M/DP); JSR* 37, *JSR* 118, August 2004.http://java.sun.com/products/midp/.

[41] Sun Microsystems Inc. *Connected Limited Device Configuration (CLDC); JSR 30, JSR 139,* August 2004. http://java . sun. com/products/cldc/.

[42] Martin Gudgin Aaron Skonnard. *Essential XML Quick Reference: A Programmer s Reference to XML, XPath, XSLT, XML Schema, SOAp, and More.* Addison Wesley, 2002.

[43] Object Management Group (OMG). *Notificatíon Servíce Specificatíon,* August 2004. http://www.omg.org/technology/documents/formal/notification_ service. htm%.

[44] Paul Brown Chris Haddad, Kevin BedeU. *Programmíng Apache Axís.* O'Rei11y & Associates, 2002.

[45] Mi-Jung Choi, James W. Hong, Hong-Taek Ju. "XML-Based Network Management for IP Network. *ETRl Journal,* 25(6):445-463, December 2003.

[46] Jon Sevy. Java SNMP Package. Technical report, Drexel University, 2004. `http: / / edge . mcs.drexel.edu/GICL/people/sevy/snmp/snmp.html.`

[47] kObjects.org. *ME4SE,* December 2004. `http://kobj ects. org/.`