UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Fernando Luis Caprio da Costa Junior

MobiMan: Uma Arquitetura Orientada a Serviços Web para o Gerenciamento de Ambientes de Computação Móvel

São Leopoldo
2006

Fernando Luis Caprio da Costa Junior

MobiMan: Uma Arquitetura Orientada a Serviços Web para o Gerenciamento de Ambientes de Computação Móvel

Dissertação submetida à avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada

Orientador: Prof. Dr. Luciano Paschoal Gaspary

São Leopoldo
2006

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Caprio da Costa Junior, Fernando Luis

MobiMan: Uma Arquitetura Orientada a Serviços Web para o Gerenciamento de Ambientes de Computação Móvel / por Fernando Luis Caprio da Costa Junior. — São Leopoldo: Ciências Exatas e Tecnologicas da UNISINOS, 2006.

70 f.: il.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos. Ciências Exatas e Tecnológicas Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, BR–RS, 2006. Orientador: Gaspary, Luciano Paschoal.

1. Gerenciamento. 2. Computação Móvel. 3. Web Services. 4. WSDM. I. Gaspary, Luciano Paschoal. II. Título.

"If human beings don't keep exercising their lips, he thought, their mouths probably seize up. After a few months consideration and observation he abandonded this theory in favor of a new one. If they don't keep on exercising their lips, he thought, their brains start working."

Douglas Adams

Agradecimentos

Ao meu orientador Prof. Luciano Paschoal Gaspary, pelo incentivo, apoio e orientação, os quais foram imprescindíveis para a conclusão desta dissertação.

Aos professores do Programa de Pós-Graduação em Computação Aplicada (PIPCA), que de alguma forma ou outra tornaram meu trabalho possível.

A meus pais, Fernando Caprio e Maria Luiza, pelo apoio incondicional durante os dois anos de mestrado, meus sinceros agradecimentos.

Aos meus irmãos, Cristiano Costa e Simone Costa, por todas as vezes que me ajudaram quando precisei, meu muito obrigado.

A minha namorada Bruna Peter, por compreender minha ausência em vários momentos, meu amor e meu carinho.

Aos amigos Daniel Bonatto, Dario Franz e Marcelo Cardozo que trilharam esse caminho ao meu lado e auxiliaram na superação dos diversos percalços encontrados.

Aos colegas de curso que tornaram-se verdadeiros amigos e ajudaram na superação dos obstáculos presentes nessa caminhada.

A todos integrantes do projeto MHolo, em especial o Prof. Jorge Barbosa, pelo apoio fornecido no alcance dessa conquista.

A Hewlett-Packard Computadores (HP), pela concessão da bolsa de mestrado e apoio financeiro.

Resumo

Os serviços e as aplicações de computação móvel possuem exigências para o seu bom funcionamento, cujo atendimento é dificultado em virtude do ambiente altamente dinâmico em que estão inseridos. Nesse ambiente, o gerenciamento passa a ser peça chave para propiciar o funcionamento harmonioso tanto da infra-estrutura física, quanto das aplicações. As soluções existentes para essa finalidade se restringem à monitoração dos recursos de hardware, não oferecendo mecanismos para o gerenciamento das aplicações executadas nos dispositivos portáteis. Esta dissertação propõe uma arquitetura para gerenciamento de ambientes de computação móvel que permite obter e disseminar, de forma flexível, informações sobre o funcionamento dos dispositivos e, sobretudo, das aplicações. A arquitetura é baseada no padrão Web Services Distributed Management, que propõe o emprego de serviços Web na modelagem de interfaces de gerenciamento. A dissertação apresenta, ainda, resultados obtidos com a instanciação da arquitetura em um ambiente real.

Palavras-chave: Gerenciamento, Computação Móvel, Web Services, WSDM.

TITLE: "MobiMan: A Web Service-Oriented Architecture for Management of Mobile Computing Environments"

Abstract

Mobile computing services and applications pose special requirements to run properly, whose satisfaction is hampered due to the highly dynamic environment where they are executed. In this environment, management becomes an important element to enable harmonious functioning of both the mobile physical infrastructure and the applications. Current solutions are restricted to resource monitoring, and do not provide mechanisms to manage the applications running on top of the mobile devices. This work proposes an architecture to manage mobile computing environments, which allows one to obtain and disseminate, in a flexible manner, information about the devices and the applications. The architecture is based on the Web Services Distributed Management standard. This work also presents results obtained with the instantiation of the architecture in a real setup.

Keywords: Management, Mobile Computing, Web Services, WSDM.

Sumário

Lista de Abreviaturas

3G Third Generation

AP Access Point

BER Basic Encoding Rules

CCML Centaurus Communication Markup Language

CF Compact Framework

CLR Common Language Runtime

COM Component Object Model

DLL Dynamic Link Library

EMS Enterprise Management Service

EPR End Point Reference

HML Holo Modeling Language

HNS Holo Name Server

HTTP Hiper-Text Tranport Protocol

IP Internet Protocol

MIB Management Information Base

MOWS Management Of Web Services

MUWS Management Using Web Services

NMS Network Management System

OTASP Over the Air Service Provisioning

OTAPA Over the Air Parameter Administration

OTASD Over the Air Software Download

OTAMI Over the Air Mobile Diagnostics

PDA Personal Digital Assistant

SMI Structure of Management Information

SMS Short Message Service

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol

TCP Transport Control Protocol

UDDI Universal Description Discovery and Integration

URL Uniform Resource Locator

VM Virtual Machine

WS Web Service

WSDL Web Services Description Language

WSDM Web Services Distributed Management

XML eXtensible Markup Language

Lista de Figuras

Capítulo 1

Introdução

Nos últimos anos tem-se observado uma proliferação do número de dispositivos portáveis graças a fatores como a redução de preço dos mesmos. Entre esses dispositivos se encontram desde *notebooks* até *hardware* de menor porte como PDAs *(Personal Digital Assistants)* e celulares de terceira geração (3G). Tais dispositivos se conectam à rede de comunicação através de tecnologias de rede sem fio como *bluetooth* e *WiFi* (padrão IEEE 802.11), atingindo velocidades de até 54 Mbps (802.11a e 802.11g). A franca utilização desses equipamentos está condicionada à disponibilização de aplicações avançadas, envolvendo, por exemplo, áudio, vídeo e sensibilidade à localização.

No cenário recém mencionado, em que aplicações cada vez mais sofisticadas passam a ser implantadas, o gerenciamento dessas aplicações e dos dispositivos torna-se de fundamental importância. A obtenção de indicadores de funcionamento e a execução de procedimentos junto aos dispositivos e aplicações podem ser decisivas para regular e assegurar o bom funcionamento do ambiente de computação móvel. Por exemplo, no caso de uma aplicação de videoconferência executando em um dispositivo portátil, é bastante desejável coletar informações que permitam acompanhar a qualidade de vídeo e voz percebidas pelo usuário. Somando a essas informações outras vinculadas ao dispositivo (ex: bateria restante e taxa de datagramas descartados pelo dispositivo), é possível alimentar uma aplicação de gerenciamento, que, por sua vez, poderá reagir de diversas formas a uma possível degradação do serviço.

As diversas informações providas pelo ambiente de gerenciamento, de acordo com o exemplo recém mencionado, podem ser divididas em dois grupos: infra-estrutura e aplicação. Gerenciar a infra-estrutura física, incluindo estações e dispositivos de rede, tem sido uma prática bastante comum há muitos anos. O gerenciamento de aplicações, por outro lado, é freqüentemente ignorado, mesmo em redes cabeadas.

1.1 Definição do problema

Algumas soluções foram propostas para o gerenciamento de ambientes de computação móvel [1, 2, 3]. Essas, porém, têm seu foco em infra-estrutura, se preocupando exclusivamente com gerenciamento de pontos de acesso, PDAs, celulares e com aspectos como mobilidade e desconexão. Outra questão importante é que muitos dos mecanismos utilizados por essas soluções para comunicação com os dispositivos são proprietários, o que dificulta sua interoperação com sistemas de gerenciamento consagrados, baseados em padrões abertos (usados em muitas instituições).

O gerenciamento das aplicações, conforme pode ser percebido, acaba sendo negligenciado. Propostas que oferecem um arcabouço para o desenvolvimento de aplicações de computação móvel, tais como MHolo [4], One.world [5] e .NET Compact Framework [6], não se preocupam em prover gerenciamento para essas aplicações visando, por exemplo, detectar possíveis falhas, analisar seu desempenho e permitir sua configuração.

1.2 Objetivos

Para suprir a lacuna recém mencionada, este trabalho propõe uma arquitetura orientada a serviços Web para o gerenciamento de ambientes de computação móvel denominada MobiMan (*Mobile Computing Management Architecture*). A arquitetura permite obter e disseminar, de forma flexível, informações sobre o funcionamento dos dispositivos e, sobretudo, das aplicações que executam nos mesmos. Além disso, os objetivos específicos a serem atingidos pela arquitetura proposta nesta dissertação são:

- permitir gerenciamento tanto das aplicações quanto da infra-estrutura de ambientes de computação móvel;
- minimizar o custo computacional, não gerando uma sobrecarga muita alta na utilização dos recursos (CPU, memória e rede de comunicação);
- oferecer, quando possível, transparência para o desenvolvedor, permitindo que a arquitetura possa ser empregada sem alteração do código das aplicações.

Em consonância com tendências atuais da área de gerenciamento - e para potencializar sua integração com outros sistemas - a arquitetura foi desenvolvida com base no padrão WSDM (*Web Services Distributed Management*) [7], padronizado pelo OASIS em maio de 2005. É importante destacar que, até onde sabemos, este é o primeiro trabalho a propor um componente para gerenciamento de dispositivos e aplicações móveis através do *framework* WSDM.

Esta dissertação está inserida em um projeto mais amplo denominado MHolo, que visa a oferecer um arcabouço para a especificação e a execução de aplicações de computação móvel. O projeto está dividido em três áreas: Aplicação, Infra-estrutura e Gerenciamento. O presente trabalho faz parte da área de Gerenciamento.

1.3 Organização da dissertação

O restante desta dissertação está organizado da seguinte forma: o Capítulo 2 apresenta ambientes para o desenvolvimento de aplicações móveis, os requisitos de gerenciamento impostos por aplicações dessa natureza e as tecnologias de gerenciamento atualmente existentes. O Capítulo 3 introduz e discute trabalhos relacionados que se preocupam de alguma forma em oferecer gerenciamento para ambientes de computação móvel. O Capítulo 4 aborda a arquitetura MobiMan. O Capítulo 5 apresenta uma instanciação da arquitetura e sua avaliação em um ambiente real. Por fim, o Capítulo 6 encerra o trabalho com considerações finais e perspectivas de trabalhos futuros.

Capítulo 2

Gerenciamento de Ambientes de Computação Móvel: Conceitos e Tecnologias

Este capítulo revisa alguns conceitos e tecnologias importantes no gerenciamento de ambientes de computação móvel. A Seção 2.1 introduz alguns ambientes que se propõem a apoiar o desenvolvimento de aplicações para dispositivos portáteis. A Seção 2.2 apresenta desafios encontrados no gerenciamento de aplicações e dispositivos em ambientes de computação móvel. A Seção 2.3 descreve o protocolo SNMP e suas limitações, em particular quando aplicado a ambientes de computação móvel. Por fim, a Seção 2.4 discorre sobre a utilização de Web Services como alternativa para o gerenciamento de equipamentos e aplicações.

2.1 Ambientes para o desenvolvimento de aplicações de computação móvel

Atualmente, existem diversos ambientes voltados ao desenvolvimento de aplicações para a computação móvel. Esta seção apresenta três deles. O primeiro, .NET Compact Framework (CF) [6], foi escolhido por se tratar de uma plataforma amplamente adotada em sistemas Windows, sendo desenvolvido e mantido pela Microsoft. A segunda proposta, one.world [8], constitui um trabalho acadêmico com características inovadoras, tais como o suporte à representação de contextos. Por fim, o MHolo [4] é um ambiente que também aborda questões relacionadas com mobilidade e contextos. Soma-se a isso, o fato do MHolo oferecer uma forma de representação intuitiva e natural para o desenvolvimento de aplicações para a computação móvel.

2.1.1 .NET Compact Framework (CF)

O Microsoft .NET Compact Framework [6] é uma plataforma para o desenvolvimento de aplicações que executem em dispositivos inteligentes (*smart devices*), como Pocket PCs e Smart Phones. Esses reconhecidamente possuem recursos limitados, apesar de suportarem tecnologias como *bluetooth* e *WiFi* para comunicação de dados sem fio. A plataforma consiste em uma versão reduzida do .NET para *desktops*, contendo um subconjunto de suas classes e algumas novas, próprias para uso em aplicações de computação móvel. Mesmo tendo sido desenvolvido para a versão Mobile do Windows, o .NET CF também pode executar nos sistemas operacionais Windows convencionais (XP, 2003 e 2000).

As aplicações .NET CF podem ser desenvolvidas em diferentes linguagens de programação, tais como C++, C#, J# e Basic. Essa facilidade se deve ao fato do *framework* fazer uso de uma máquina virtual (*Common Language Runtime - CLR*), que executa um código intermediário gerado com base no código fonte desenvolvido pelo usuário.

Os aplicativos para .NET são desenvolvidos com o Microsoft Visual Studio. Essa plataforma de desenvolvimento gera o código intermediário da aplicação. Uma tendência é a utilização desse ambiente para o desenvolvimento de Web Services. Nesse caso, o Visual Studio também fornece os arquivos com as descrições necessárias para a disponibilização dos serviços. O ambiente permite que o código seja executado tanto em PDAs como também em um emulador, a exemplo do Pocket PC 2003 (emulador do Windows Mobile 2003).

O .NET CF procura oferecer maior facilidade de desenvolvimento para os programadores através de um conjunto de classes que, apesar de ser extenso, apresenta uma interface de programação simples. Além disso, uma nova implementação da CLR oferece maior eficiência para execução do *framework* em dispositivos portáveis, que apresentam limitações tanto de memória quanto de CPU.

A Figura 2.1 ilustra uma aplicação desenvolvida com o .NET Compact Framework, disponível para usuários que tenham acesso à rede sem fio. Nessa aplicação, um Web Service é capaz de oferecer um conjunto de informações sobre as condições meteorológicas, tais como: temperatura, umidade e pressão atmosférica. Na ilustração, o primeiro fluxo representa o processo de descoberta, onde a aplicação, executando no PDA do usuário, descobre os serviços existentes no ambiente[1]. Os fluxos seguintes representam a requisição e a obtenção de informações, no caso, a temperatura atual na cidade de São Paulo. Neste exemplo, o *framework* é responsável pelo processo de descoberta dos serviços, bem como pelo suporte às comunicações

[1] A Figura apresenta uma simplificação do mecanismo de descoberta. A descoberta dinâmica de serviços, comum a Web Services, é realizada interagindo com um elemento intermediário (*service broker*) que possui a descrição dos serviços.

realizadas entre a aplicação executando no PDA e o Web Service remoto. Para oferecer outras funcionalidades na aplicação em questão pode-se ampliar o conjunto de operações previstas no Web Service ou, até mesmo, criar novos Web Services.

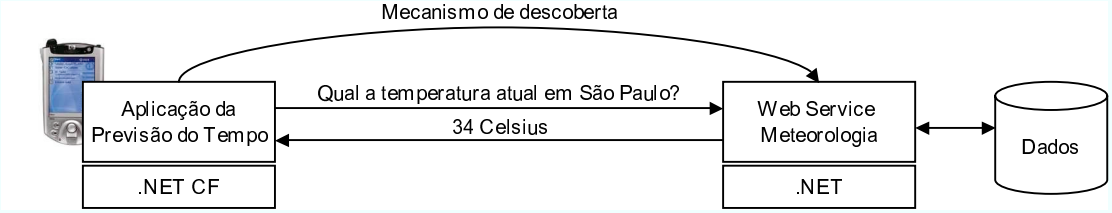


Figura 2.1 – Aplicação modelada em .NET CF utilizando Web Services.

Apesar de oferecer um conjunto de funcionalidades bastante extenso, o *framework* não se preocupa em tratar algumas características próprias de ambientes de computação móvel. A definição de contextos, por exemplo, não é possível. Esse tipo de representação pode facilitar a modelagem e o desenvolvimento de aplicações que necessitem adaptações de comportamento decorrentes da mobilidade dos dispositivos. É importante mencionar que essas adaptações dependem de um suporte a ser oferecido pelo ambiente de execução.

2.1.2 One.world

One.world [8] é um *framework* que visa auxiliar a construção de aplicações para ambientes de computação móvel. O mesmo possui um conjunto de diferentes serviços, tais como descoberta, *checkpointing*, migração e replicação. As aplicações que executam nos dispositivos móveis fazem uso de tais serviços. Além disso, a proposta define um modelo de programação que deve ser adotado pelos desenvolvedores de aplicações one.world.

O *framework* foi implementado na linguagem Java, o que faz com que as aplicações tenham de ser desenvolvidas também em Java. No caso, o usuário tem de estender as classes one.world de forma a utilizar as funcionalidades oferecidas pelo *framework* em sua aplicação. Atualmente, a plataforma tem suporte para Windows e Linux. O suporte para PDAs, porém, está em fase de desenvolvimento. No caso dos PDAs, a API Java oferecida é enxuta e tem apenas um subconjunto das funcionalidades da API para *desktops*. A implementação, portanto, deve ser refeita levando em consideração tais limitações.

Com o one.world, os desenvolvedores podem criar, de forma facilitada, aplicações sensíveis ao contexto. No caso, o programador só precisa se preocupar com a aplicação e as adaptações de contexto necessárias. Os serviços necessários para realização dessas adaptações são fornecidos pelo *framework*.

One.world é formado basicamente por quatro componentes (Figura 2.2). O

primeiro deles é uma *máquina virtual* que suporta composição *ad-hoc* entre aplicações. Diferentes dispositivos, executando aplicações one.world, podem entrar em uma infra-estrutura de rede sem fio e utilizar os serviços do *framework,* sem a necessidade de alterações nas aplicações one.world já em execução, tanto no ambiente em questão como no dispositivo do usuário. O segundo componente são as *tuplas,* que definem um meio comum para o compartilhamento de dados entre diferentes aplicações. O terceiro são os *eventos,* que permitem trocas explícitas de informações entre aplicações, de forma que as mesmas possam se adaptar a mudanças. Por fim, os *ambientes* são responsáveis por hospedar os componentes das aplicações, guardando os dados persistentes e oferecendo uma forma hierárquica de composição dinâmica de aplicações e serviços.

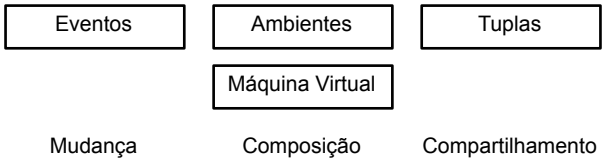


Figura 2.2 – Componentes do one.world.

O conceito de ambientes é o mecanismo central para composição de aplicações em one.world. Os ambientes servem como *containers* para tuplas de armazenamento e permitem proteção e isolamento para as aplicações, tanto entre si, como com o *kernel* do *framework* (localizado na raiz ou /). Cada aplicação consiste de no mínimo um ambiente, que é utilizado para armazenamento persistente dos dados. Uma aplicação, porém, não está limitada a um único ambiente. Uma questão importante dos ambientes é que eles permitem a composição dinâmica. Nesse caso, um ambiente gerencia todos os que estão aninhados a ele. Esse ambiente, então, tem controle sobre todas as ações realizadas pelos seus subambientes, tanto aquelas com o *kernel* do *framework* como as com os ambientes externos.

O exemplo apresentado na seção anterior em .NET é ilustrado em one.world na Figura 2.3. O mesmo apresenta uma estrutura hierárquica de ambientes. O primeiro deles é a /, que simboliza a raiz de execução da plataforma. Dentro da / existe o ambiente Meteorologia, que apresenta tuplas com um conjunto de informações sobre as condições meteorológicas. No exemplo ilustrado, o ambiente que representa a aplicação de um usuário (Aplicação de Previsão do Tempo), obtém as informações desejadas sobre a temperatura. Os dados são trocados através de mensagens assíncronas fornecidas pela máquina virtual. A aplicação one.world, portanto, utiliza os serviços do *framework* para o recebimento da temperatura requisitada. Os dados meteorológicos estão armazenados nas tuplas do ambiente Meteorologia. Quando o usuário requisita um conjunto específico de dados, esse será também armazenado nas tuplas do ambiente Aplicação de Previsão do Tempo. Outras funcionalidades

poderiam ser oferecidas à aplicação de previsão do tempo com o acréscimo de novos ambientes ou através do ambiente Meteorologia.

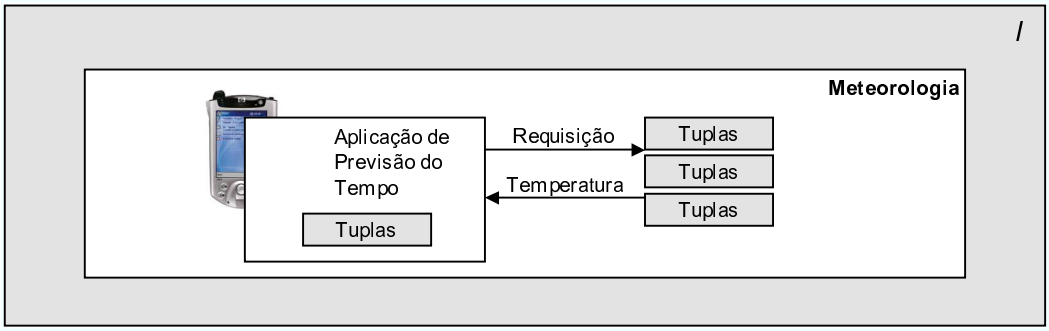


Figura 2.3 – Aplicação modelada em one.world.

2.1.3 MHolo

MHolo [4] é um ambiente para especificação e execução de aplicações de computação móvel à luz do Holoparadigma [9]. O ambiente é utilizado para especificação de aplicações conscientes de contexto, cobrindo todos os passos necessários para seu desenvolvimento. MHolo é composto pelos seguintes elementos: (a) linguagem Holo, linguagem de programação concorrente que permite a especificação de aplicações através dos conceitos do Holoparadigma, (b) HML (*Holo Modeling Language*), linguagem de modelagem que fornece programação visual em Holo, (c) HoloCase, ferramenta *case* que suporta a especificação de aplicações (através da HML) e a geração automática de código em Holo, (d) HoloEnv, ambiente de desenvolvimento integrado (IDE) que permite edição, compilação e execução de programas Holo e (e) HoloVM, máquina virtual que suporta a execução de programas nativos feitos em Holo.

O Holoparadigma possui como unidade de modelagem o ente. Um ente elementar (Figura 2.4a) é organizado em três partes: comportamento, interface e história. O comportamento define o conjunto de ações que o ente é capaz de executar. Dentre essas, as que podem ser acessadas externamente, são descritas na interface. A história, por fim, consiste em um espaço de tuplas que fica encapsulado no ente. Um ente composto (Figura 2.4b) possui a mesma organização; no entanto, suporta a existência de outros entes na sua composição (entes componentes). Nesse caso, a história é compartilhada pelos entes componentes. A Figura 2.4c apresenta um ente composto de três níveis e exemplifica a história encapsulada. O nível 0 representa o primeiro ente da hierarquia. Os níveis seguintes exemplificam o compartilhamento no acesso à história. Os entes que estão no nível 1, por exemplo, possuem acesso a história do ente que está no nível 0. A mesma lógica é utilizada nos níveis seguintes.

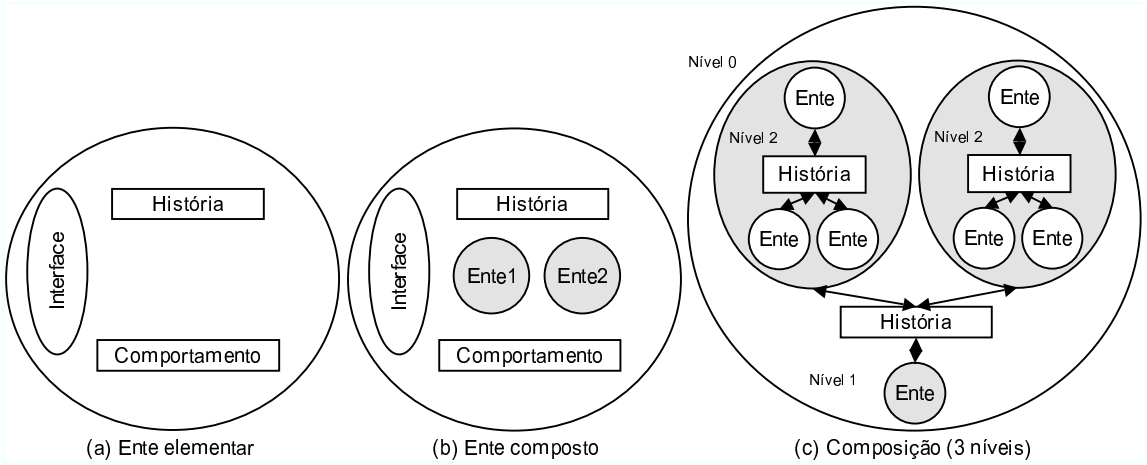


Figura 2.4 – Entes no Holoparadigma.

Os entes Holoparadigma podem ser movidos de um contexto para outro através da primitiva *move* da linguagem Holo. Essa mobilidade permite que eles mudem de nível hierárquico e, assim, tenham acesso a história e ao comportamento do novo ente hospedeiro. Além disso, novas instâncias desses entes podem ser criadas através da primitiva *clone*. Ao ser invocada, essa instancia um novo ente com a mesma funcionalidade do ente que foi clonado. O novo ente criado, então, pode ser movido para outro contexto e é totalmente independente do ente original.

A aplicação de previsão do tempo, apresentada nas seções anteriores em .NET e one.world, é ilustrada na Figura 2.5 em Holo. A mesma mostra uma aplicação composta por três entes. O primeiro é o ente *dHolo*, comum a todas execuções de aplicativos Holo. O segundo ente é a aplicação de previsão do tempo, que pode estar presente no dispositivo móvel do usuário. O terceiro ente, por fim, é o ente *Meteorologia*. No exemplo ilustrado o ente *Aplicação de Previsão do Tempo* entra dentro do ente *Meteorologia* (fluxo 1). A partir desse momento o mesmo é capaz de obter informações sobre as condições climáticas lendo a história do ente no qual está presente (fluxo 2). Embora todo ente possua além da história - interface e comportamento - por caráter de simplificação os mesmos foram apenas ilustrados no ente *Metereologia*. A interface desse ente pode, alternadamente, ser utilizada para que a aplicação obtenha as condições climáticas sem ter de se mover para o contexto do ente *Meteorologia*. Nesse caso, a aplicação faria uma chamada a um método externo disponibilizado pela interface do ente *Meteorologia*. De forma semelhante, o comportamento pode ser utilizado para fornecer métodos que retornem as informações climáticas, sem que os dados tenham de ser lidos diretamente da história do ente *Metereologia* pela aplicação. Dessa forma o ente que representa a aplicação de previsão do tempo faria a mesma mobilidade descrita no fluxo 1, porém, acessaria um método do comportamento ao invés de ler a informação da história do ente *Meteorologia*.

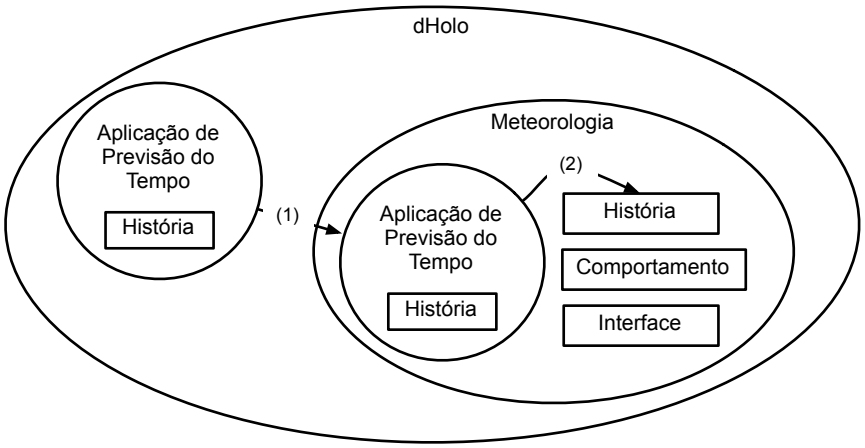


Figura 2.5 – Aplicação modelada em Holo.

As aplicações MHolo, conforme mencionado anteriormente, executam em uma máquina virtual denominada HoloVM. O suporte à execução distribuída das aplicações, no entanto, é oferecido em conjunto com um servidor chamado HNS (*Holo Name Server*). Esse servidor possui uma visão global da execução de todos os entes Holo em diferentes dispositivos, guardando a hierarquia desses entes em uma estrutura de dados local. Sempre que um ente realiza uma movimentação, por exemplo, essa mobilidade é informada pela HoloVM ao HNS para que ele atualize a hierarquia dos entes.

2.2 Gerenciamento de ambientes de computação móvel

As diferentes aplicações para a computação móvel, tais como as desenvolvidas com os ambientes recém mencionados, possuem necessidades de gerenciamento que são importantes para seu correto funcionamento. Embora a área de gerenciamento já ofereça inúmeras abordagens para monitorar e controlar ambientes tradicionais, essas abordagens não atendem de forma integral aos requisitos impostos por ambientes tão dinâmicos como os de computação móvel. Assim, deve ser feita uma revisão das áreas funcionais clássicas de gerenciamento como falhas, configuração, contabilização, desempenho e segurança. Além disso, todas as lições aprendidas ao gerenciar aplicações executando em *desktops* e servidores precisam ser repensadas levando em consideração, por exemplo, restrições de CPU e memória impostas pelos dispositivos onde as aplicações móveis são executadas. Esta seção apresenta alguns dos desafios relacionados com cada uma das cinco áreas funcionais para o gerenciamento de ambientes de computação móvel. Os desafios aqui enumerados são resultados de um estudo exploratório desenvolvido durante a dissertação visando caracterizar as exigências de gerenciamento que um ambiente de computação móvel possui.

2.2.1 Falhas

Quando se trata de ambientes de computação móvel é importante que exista um sistema capaz de monitorar possíveis falhas, no intuito de garantir o funcionamento correto da infra-estrutura de rede e das aplicações. A mobilidade constante dos usuários, por exemplo, acarreta um novo conjunto de possíveis falhas que não são comuns às redes com fio. A movimentação dos usuários entre diferentes células, que oferecem conexão de rede sem fio, pode levar a falhas nas aplicações devido a fatores como desconexão e perda de sinal.

Além da mobilidade, um aspecto introduzido pelos dispositivos que fazem parte de um ambiente de computação pervasiva (PDAs e *notebooks*) é a pequena autonomia em relação à bateria. Nesse contexto, diferentes aspectos relacionados com falhas devem ser gerenciados. Como exemplo, poderia-se identificar que a bateria de um determinado dispositivo está terminando e adaptar as aplicações para otimizarem o consumo, garantindo assim, tempo suficiente para salvar as informações e evitar que os dados do usuário sejam perdidos.

Outra questão importante é que muitas aplicações são sensíveis ao contexto e dependem de informações que, muitas vezes, são obtidas através de sensores [10]. Gerenciar tais sensores é fundamental, pois muitas vezes são importantes para o funcionamento de aplicações que dependem de informações críticas, a exemplo de aplicações médicas [11]. As limitações de CPU e memória dos dispositivos portáteis dificultam o emprego de soluções tradicionais de gerenciamento. Nesse contexto, novas soluções precisam ser empregadas para gerenciar falhas em aplicações e sensores e, assim, atender as características próprias dos ambientes de computação móvel.

2.2.2 Configuração

Quando um usuário se desloca pelas diferentes células que fazem parte da infra-estrutura de rede sem fio, muitas vezes existem algumas configurações que devem ser feitas em suas aplicações ou, até mesmo, em seu dispositivo. Nesse sentido, existem necessidades como a de autoconfiguração nesse tipo de ambiente. A mesma surge da dificuldade de configurar todos os nodos manualmente toda vez que ocorre mobilidade o que, além de tomar muito tempo, levaria a uma grande quantidade de erros. No contexto de configuração voltada a ambientes de computação móvel, os protocolos existentes são muitas vezes ineficientes [12]. Essa característica levou ao surgimento de propostas como DCDP (*Dynamic Configuration and Distribution Protocol*) [12], Mobile IP [13] e DMA (*Dynamic Mobility Agent*) [14]. A autoconfiguração de aplicações, por outro lado, permite que as mesmas adaptem seu comportamento de acordo com informações obtidas do ambiente de execução. O sistema de gerenciamento pode, por exemplo, informar uma aplicação de videoconferência das limitações de banda de comunicação, oferecidas pela infra-estrutura

de rede sem fio, em um determinado momento. A aplicação, então, é capaz de fazer ajustes na qualidade da imagem transmitida em tempo de execução. Isso pode ser feito usando um esquema de codificação que resulte em uma *stream* que exija menos banda para ser transmitida. Dessa forma, o usuário vai continuar recebendo o vídeo sem atrasos, mesmo que com uma qualidade inferior.

2.2.3 Contabilização

A área de contabilização se preocupa em monitorar diferentes dispositivos e aplicações que fazem parte de um ambiente de computação móvel, de forma que o administrador possa caracterizar o uso da infra-estrutura pela qual é responsável. Em se tratando de redes sem fio, um novo conjunto de informações pode ser obtido dos dispositivos e das aplicações. Os pontos de acesso (APs), por exemplo, podem inferir a localização dos usuários através de triangulação e outras tecnologias disponíveis. Já as aplicações, quando gerenciadas, fornecem um novo conjunto de informações. Tais informações, somadas as obtidas da infra-estrutura, permitem que o gerente visualize uma caracterização mais completa e precisa da utilização das aplicações e dos dispositivos. Diferentes trabalhos [15, 16, 17] fazem análises de ambientes de computação móvel com auxílio dos dados obtidos dos APs e dos servidores de autenticação. Entre os dados que podem ser obtidos, pode-se citar: número total de usuários na rede sem fio, vazão de dados, tempo das sessões, total de dados enviados/recebidos, entre outros. Com essas informações é possível caracterizar, por exemplo, necessidades de banda de comunicação e, até mesmo, diferenças entre perfis de usuários em determinadas organizações.

2.2.4 Desempenho

Assim como nas redes com fio, aspectos relacionados com gerenciamento de desempenho são importantes em contextos de rede sem fio. Com esse tipo de informação se pode analisar o comportamento dos dispositivos e das aplicações que fazem parte do ambiente de computação móvel. As informações de desempenho da rede podem, por exemplo, ser utilizadas para detectar insuficiências de sinal e possíveis necessidades de redimensionamentos da infra-estrutura presente. Além disso, aplicações como voz sobre IP (VoIP) executando em *Smartphones* dependem que requisitos mínimos de qualidade de serviço (QoS) sejam atendidos. Para essas aplicações é importante garantir que não exista um atraso muito grande na transferência de datagramas entre um cliente VoIP e outro. De forma análoga, aplicações que dependem de informações de contexto podem ter necessidades de qualidade de contexto (QoC) [18]. Esse é o caso, por exemplo, de uma aplicação médica ou de uma aplicação de um corpo de bombeiros, que devem receber as informações em tempo hábil sob pena das pessoas não receberem o atendimento necessário.

2.2.5 Segurança

Para um dispositivo acessar uma rede de comunicação cabeada, este deve se conectar fisicamente em um ponto de rede presente na infra-estrutura. Quando se trata de redes sem fio, por outro lado, a conexão física é inexistente. Essa característica, apesar de facilitar o usuário, dificulta o gerenciamento de segurança sob o ponto de vista do administrador da rede. O problema fundamental é que qualquer pessoa pode fazer tentativas de acesso à rede sem estar fisicamente presente em um local monitorado pelo administrador da rede. Nesse contexto, antenas comumente vendidas ou até mesmo feitas em casa podem ser empregadas. Um usuário malicioso estacionado na frente da universidade, por exemplo, pode utilizar uma antena para capturar pacotes da rede sem ser autorizado. Diferentes informações obtidas dos dispositivos de rede sem fio e das aplicações devem ser analisadas pelo sistema de gerenciamento para garantir a segurança da infra-estrutura. Como exemplo, quando uma pessoa autentica em um servidor para utilizar a rede sem fio, esse pode anotar sua localização. Caso o mesmo usuário autentique novamente em uma localização distante em um intervalo curto de tempo, um alerta de segurança pode ser gerado, pois o mesmo usuário não poderia estar em dois lugares ao mesmo tempo. Um outro exemplo é uma aplicação feita para operar em um aeroporto. Nesse caso, poderia-se pensar em gerenciamento de segurança no nível de aplicação. Um passageiro que tenta acessar informações sigilosas sobre o funcionamento do aeroporto, por exemplo, pode fazer com que o sistema de gerenciamento notifique o administrador da rede de uma tentativa de acesso não autorizada.

2.3 Arquitetura SNMP

Visto algumas necessidades de gerenciamento nas diferentes áreas funcionais, esta seção apresenta o protocolo SNMP, que é comumente empregado no gerenciamento de redes cabeadas. São enumeradas, também, algumas insuficiências do protocolo para o gerenciamento de ambientes de computação móvel.

A crescente expansão do número de dispositivos e, conseqüentemente, da complexidade das redes TCP/IP incentivou o surgimento de mecanismos que permitissem seu efetivo gerenciamento [19]. Uma das primeiras propostas reais nesse sentido foi lançada em 1988, com a definição do protocolo SNMP ou SNMPv1. Esse protocolo tornou-se rapidamente o esquema de gerenciamento independente de fabricante mais utilizado [20].

O modelo de gerenciamento de rede adotado pelo protocolo SNMP possui quatro elementos chave: estação de gerenciamento, agentes de gerenciamento, base de informação de gerenciamento (MIB) e protocolo para troca de informações de gerenciamento (vide Figura 2.6) [21, 20, 19].

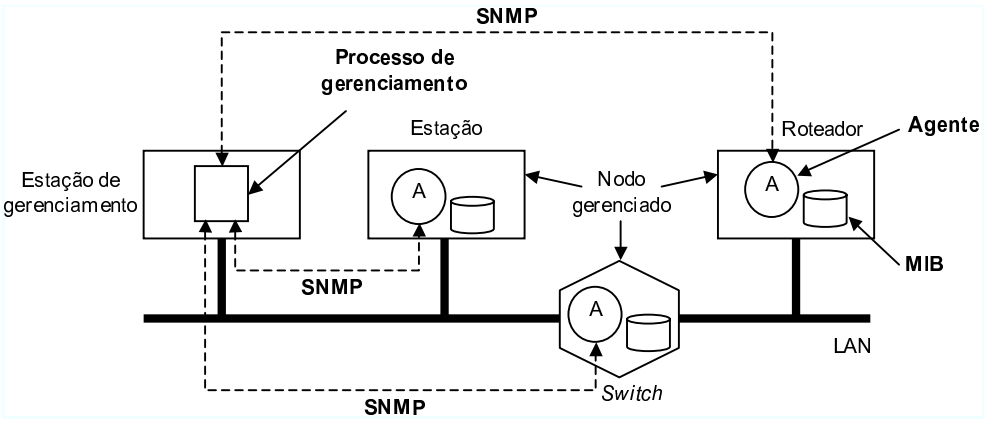


Figura 2.6 – Arquitetura de gerenciamento TCP/IP.

2.3.1 Estação de gerenciamento

A estação de gerenciamento é a interface entre o administrador de rede humano e o sistema de gerenciamento. Segundo Stallings [20], ela terá, no mínimo:

- um conjunto de aplicações de gerenciamento para análise de dados, recuperação de falhas, entre outras;
- uma interface de usuário através da qual o administrador possa monitorar e controlar a rede;
- um protocolo através do qual a estação de gerenciamento e as entidades gerenciadas possam trocar informações de gerenciamento e controle;
- uma base de dados com informações extraídas a partir das bases de informação de gerenciamento de todas as entidades gerenciadas.

2.3.2 Agente de gerenciamento

Outro elemento chave da arquitetura é o agente de gerenciamento. Equipamentos de rede como *switches*, roteadores e *hubs* podem conter agentes SNMP e, portanto, podem ser monitorados pela estação de gerenciamento [20]. O agente:

- responde a pedidos de informações oriundos da estação de gerenciamento;
- realiza ações requisitadas pela estação de gerenciamento;
- pode de maneira assíncrona enviar à estação de gerenciamento informações importantes, embora não solicitadas.

2.3.3 Base de informações de gerenciamento

Cada dispositivo gerenciável mantém uma ou mais variáveis que descrevem seu estado. Essas variáveis são denominadas objetos [19]. A coleção desses objetos

é denominada MIB. A estação de gerenciamento monitora um equipamento recuperando valores de objetos da MIB do mesmo. Além disso, a estação de gerenciamento pode solicitar a realização de uma ação no agente ou modificar a configuração de seus parâmetros alterando valores de variáveis específicas.

2.3.4 Protocolo de gerenciamento (SNMP v1)

A estação de gerenciamento e os agentes trocam informações através de um protocolo de gerenciamento de rede, que provê as seguintes funcionalidades [20]:

- Get: permite que a estação de gerenciamento recupere o valor de objetos do agente;
- Set: permite que a estação de gerenciamento altere o valor de objetos do agente;
- Trap: permite que o agente notifique a estação de gerenciamento sobre a ocorrência de eventos significativos.

2.3.5 Limitações do protocolo SNMP

O protocolo SNMP apresenta algumas limitações que dificultam sua adoção em ambientes de computação móvel [22]. Entre as desvantagens do protocolo, duas podem ser destacadas. A primeira está relacionada com o fato dos agentes SNMP serem implementados de forma simplificada, com operações, em geral, do tipo *polling-based*. Tal característica faz com que a aplicação de gerenciamento se torne um gargalo, uma vez que a mesma deve solicitar por informações de gerenciamento sempre que tiver interesse. Todas as informações requisitadas devem, então, ser processadas pela aplicação. Isso também afeta a rede de comunicação, pois um enorme conjunto de dados trafega entre a aplicação de gerenciamento e o agente SNMP. A segunda desvantagem é que o protocolo não permite a definição de filtros, o que possibilitaria uma coleta e disseminação seletiva das informações de gerenciamento, desonerando o dispositivo móvel e a rede de comunicação.

Adicionalmente às questões recém comentadas é importante mencionar que o protocolo SNMP apresenta limitações relacionadas com desempenho. Alguns trabalhos analisaram o desempenho do emprego do protocolo SNMP em contraste com a utilização de Web Services e XML [23, 22]. Apesar de SNMP ter um desempenho superior para a transferência de um conjunto muito pequeno de dados, o uso de XML pode ser mais eficiente para transferência de conjuntos maiores caso a compactação seja empregada [23]. Em função dessa constatação a próxima seção introduz as possibilidades de incorporar Web Services como alternativa para o gerenciamento de ambientes de computação móvel.

2.4 Web Services como uma nova alternativa

Os Web Services [24] representam uma tecnologia emergente, desenvolvida e mantida pelo WWW Consortium (W3C) [25] para disponibilizar serviços na Web com interfaces padrões. Dessa forma, as interfaces podem ser utilizadas por aplicações que não conhecem a implementação dos serviços. O principal objetivo dos Web Services, portanto, é estruturar os conteúdos Web e os serviços associados de forma que possam ser acessados por aplicações distribuídas. Essa seção, em um primeiro momento, descreve os Web Services. Em um segundo momento, são apresentadas duas especificações para criação de sistemas de gerenciamento baseados nesse tipo de tecnologia.

A arquitetura de Web Services [26] é baseada na interação entre três componentes principais (Figura 2.7): provedor de serviços (*service provider*), registro de serviços (*service registry*) e requerente de serviços (*service requestor*). Esses componentes interagem usando operações de publicação (*publish*), busca (*find*) e ligação (*bind*). O provedor é a entidade que provê acesso aos Web Services e publica a descrição dos serviços (1) em um registro. A publicação de um serviço é realizada através de sua descrição seguindo o padrão WSDL (*Web Services Description Language*). O requerente localiza, através da descrição, o serviço que tem interesse no registro (2) e usa essa informação para realizar a ligação com os provedores escolhidos. Nesta etapa, o requerente recebe, além da informação de como acessar o provedor, informações de como a requisição ao serviço deve ser realizada. Por exemplo, o nome da operação a ser acessada e os parâmetros que devem ser informados (bem como os seus tipos). De posse destas informações, o serviço pode ser diretamente acessado (3).

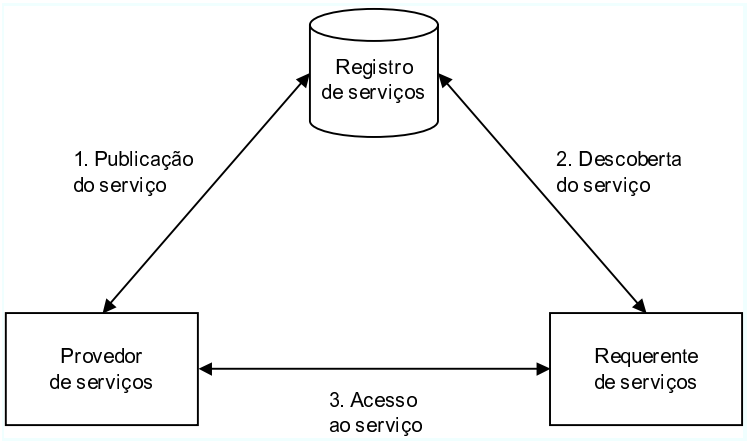


Figura 2.7 – Modelo conceitual de Web Services.

As interfaces de um serviço, conforme recém mencionado, são descritas através da WSDL. A mesma constitui um *framework* baseado em XML para descrição dos

serviços como pontos de comunicação fim-a-fim capazes de trocar mensagens. Um serviço é descrito por: (a) um identificador uniforme de serviço (*Union Resource Identifier - URI*), (b) as operações suportadas e (c) as mensagens que podem ser trocadas. A herança de serviços também é suportada. O WSDL não define qual protocolo deve ser utilizado para comunicação entre os Web Services e as aplicações de gerenciamento, porém existe uma tendência na utilização do *Simple Object Access Protocol* (SOAP). O SOAP é um protocolo sem estados (*stateless*) com codificação baseada em XML. O mesmo opera geralmente sobre HTTP/TCP/IP.

A descoberta de interfaces, por sua vez, é suportada pelo *Universal Description Discovery and Integration* (UDDI). Tal mecanismo permite que os provedores de serviço os ofereçam em uma forma padrão para que possam ser descobertos e utilizados pelos diferentes consumidores. O UDDI é implementado como um Web Service conhecido em termos de localização e interface.

Diversos padrões são empregados para criação de Web Services. As propostas de gerenciamento utilizam esses padrões para oferecer soluções que possam ser facilmente integradas. Um conjunto desses padrões é apresentado na próxima subseção.

2.4.1 Padrões utilizados na concepção de Web Services

Esta seção, em um primeiro momento, apresenta o padrão WS-Addressing [27], utilizado para o endereçamento de Web Services. Em um segundo momento é introduzido o padrão WS-Notification [28], que oferece suporte a eventos.

WS-Addressing

A especificação do WS-Addressing [27], desenvolvida pela IBM, Microsoft e Bea, provê uma forma padrão de incorporar suporte ao endereçamento das mensagens enviadas por Web Services. Nesse caso, as mensagens SOAP síncronas e assíncronas, que trafegam entre os consumidores e os Web Services, são endereçadas de forma uniforme.

Em SOAP não existe uma maneira de especificar o destino de uma mensagem. Além disso, o mesmo não trata o retorno dessa mensagem e possíveis erros que venham a acontecer na transmissão da mesma. Esses detalhes foram deixados para a camada de transporte. Mesmo que a utilização de endereçamento na camada de transporte seja suficiente para alguns casos, em outros, porém, é um fator limitante. Uma determinada aplicação, por exemplo, pode querer fazer uma requisição SOAP e solicitar a resposta por e-mail ou SMS. Para permitir esse tipo de aplicação, o WS-Addressing incorpora elementos como: *delivery*, *reply-to* e *fault handler* no envelope SOAP. A Figura 2.8 apresenta um exemplo de cabeçalho SOAP acrescido do WS-Addressing. Os elementos do WS-Addressing são:

- MessageID: identificador único da mensagem;

- ReplyTo: endereço de resposta padrão;
- FaultTo: endereço de resposta no caso de falhas;
- To: endereço destino;
- Action: ação a ser executada no endereço destino.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
               xmlns:wsa="http://www.w3.org/2004/12/addressing">
    <S:Header>
        <wsa:MessageID>
            http://example.com/SomeUniqueMessageIdString
        </wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://myClient.example/someClientUser</wsa:Address>
        </wsa:ReplyTo>
        <wsa:FaultTo>
            <wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
        </wsa:FaultTo>
        <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
        <wsa:Action>http://myserver.example/DoSomething</wsa:Action>
    </S:Header>
    <S:Body>
         <!-- Corpo da mensagem SOAP -->
    </S:Body>
 </S:Envelope>
```

Figura 2.8 – Envelope SOAP com WS-Addressing.

O WS-Addressing introduz o conceito de *endpoint reference* (EPR). *Endpoint* é um termo comumente utilizado para identificar um endereço no qual o Web Service pode ser encontrado. *Endpoint reference*, por outro lado, é um modelo para descrever esses destinos. Além do EPR, existem as propriedades de endereçamento das mensagens, que podem incluir um ou mais EPRs, e provêm um contexto para a informação de destino. Para facilitar o entendimento, pode ser feita uma analogia com o sistema de correios: o endereço de destino é escrito no centro do envelope e o de retorno atrás. As informações contidas em cada um desses endereços (nome, rua, cidade e CEP) são o EPR. A forma como as mesmas estão dispostas no envelope, por outro lado, são as propriedades de endereçamento de mensagem do WS-Addressing.

WS-Notification

O WS-Notification (WSN) [28] é um conjunto de especificações com objetivo de padronizar o processo de criação de Web Services com suporte à geração de eventos. Os eventos ocorrem entre produtores e consumidores. No caso, os consumidores assinam por diferentes informações que são disponibilizadas de forma assíncrona pelos produtores através de notificações. O WSN consiste de três especificações desenvolvidas em conjunto com o WSRF (*Web Service Resource Framework*) [29]:

- WS-BaseNotification: esse documento apresenta a funcionalidade básica, definindo *NotificationProducers* (produtores de notificações), *NotificationConsumers* (consumidores de notificações), notificações e assinaturas;

- WS-Topics: quando um usuário assina por uma informação em um produtor de notificações, essa assinatura é associada com um ou mais tópicos. O documento que descreve o WS-Topics ilustra como a estrutura desses tópicos deve ser definida e criada;
- WS-BrokeredNotification: em alguns casos, a entidade que cria a notificação não consegue tratar um grande número de assinaturas. Esse documento define uma forma de criar um *publisher* que simplesmente cria mensagens e as distribui através de um *NotificationBroker*.

A Figura 2.9 exemplifica uma utilização do WS-Notification. Nessa, um Web Service com suporte ao WS-BaseNotification permite que as aplicações de gerenciamento assinem por informações que serão entregues de forma assíncrona. No caso, a aplicação de gerenciamento assina por um tópico chamado *TonerBaixo*. Sempre que o *toner* da impressora estiver baixo, então, o usuário é notificado.

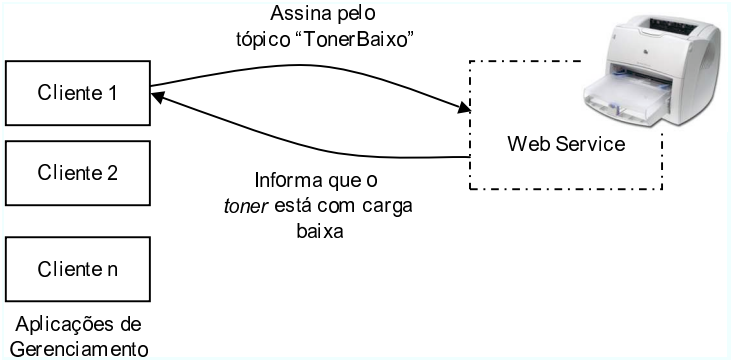


Figura 2.9 – Funcionamento do WS-Notification.

2.4.2 Web Services Distributed Management

Visto alguns padrões comuns à implementação de Web Services, esta seção apresenta o *Web Services Distributed Management* (WSDM) [7]. O mesmo foi desenvolvido por um grupo de companhias como Hewlett Packard e IBM. Definido como um padrão aprovado pelo comitê OASIS (*Organizations and Society in Information Systems*), o WSDM é distribuído livremente e consiste em dois conjuntos de especificações que definem: gerenciamento utilizando Web Services (*Management Using Web Services - MUWS*) [30, 31] e gerenciamento de Web Services (*Management of Web Services - MOWS*) [32].

O MUWS especifica uma forma de representar e acessar interfaces de gerenciamento usando Web Services. O principal objetivo é que os sistemas que se tenha interesse em gerenciar adotem esse padrão e, assim, as aplicações de gerenciamento que implementam o MUWS podem gerenciar esses sistemas de forma distribuída.

O MUWS é dividido em duas partes. A primeira delas define os conceitos fundamentais para o gerenciamento com utilização de Web Services. A segunda

parte define formatos de mensagem específicos para promover interoperabilidade entre diferentes implementações que façam uso do MUWS.

A segunda especificação (MOWS), por sua vez, apresenta um modelo para o gerenciamento de Web Services. Além disso, são propostos métodos para descrever e acessar os serviços utilizando o MUWS, conforme será visto nas próximas subseções.

MUWS

O principal objetivo do MUWS é prover gerenciamento distribuído com a utilização de Web Services. Através de um *framework* flexível e dos protocolos comuns a Web Services, o mesmo consegue facilitar o processo de gerenciamento. As funções propostas pelo MUWS são as esperadas em sistemas de gerenciamento em geral. Entre elas, pode-se citar: monitoração da qualidade de serviço, controle de tarefas e gerenciamento do tempo de vida de um recurso. A proposta consiste de um conjunto de especificações de Web Services para prover troca de mensagens, descrição e descoberta de recursos, propriedades de acesso e notificações. O WSDM propõe um contrato entre o consumidor e os recursos gerenciáveis. Dessa forma, o consumidor sabe quais mensagens podem ser trocadas entre ele e o recurso que está administrando. O foco central do MUWS, no entanto, é o recurso gerenciável.

A arquitetura do MUWS é ilustrada na Figura 2.10. A mesma apresenta os conceitos básicos do gerenciamento com base em Web Services. Nessa arquitetura, um Web Service é utilizado para prover acesso a um recurso (impressora). Um sistema de gerenciamento é capaz de fazer a descoberta desse dispositivo e, então, trocar mensagens de forma a obter informações, assinar por eventos ou controlar o mesmo. Pode-se perceber que um Web Service *endpoint* (*End Point Reference - EPR*) é utilizado para acessar um recurso gerenciável. Na ilustração, o mesmo está em uma impressora capaz de notificar quando o *toner* está com nível baixo. As mensagens trocadas entre o sistema de gerenciamento e a impressora são, portanto, feitas através do EPR da mesma, que é descoberto previamente. A descoberta de EPRs é definida no documento WS-Addressing.

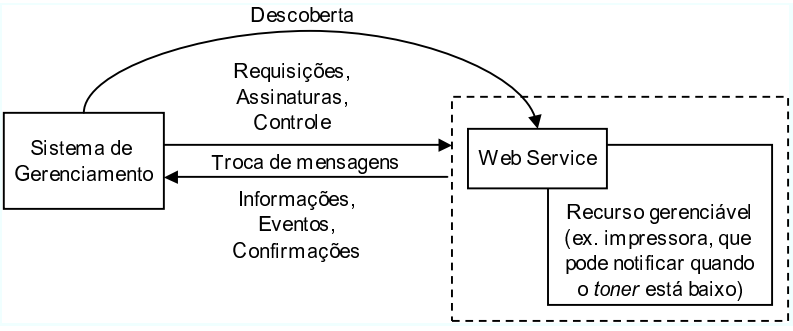


Figura 2.10 – Arquitetura proposta pelo WSDM.

A gerenciabilidade é um aspecto presente em cada recurso. Uma impressora, por exemplo, é obviamente capaz de imprimir. Imprimir é o aspecto operacional/funcional dessa impressora. A mesma impressora, por outro lado, pode indicar se está ligada, ou até mesmo, se o seu *toner* acabou. Tais indicações compõem as capacidades de gerenciamento oferecidas pelo recurso. Um recurso gerenciável, portanto, deve suportar um conjunto de capacidades de gerenciamento. Cada uma dessas capacidades, tais como a habilidade de indicar se o dispositivo está ligado, apresenta uma semântica distinta. Uma implementação de um recurso gerenciável provê uma série de capacidades de gerenciamento via Web Services *endpoints*. No WSDM, cada capacidade de gerenciamento possui, portanto:

- uma única identificação;
- uma semântica definida;
- um conjunto de propriedades, operações, eventos (notificações) e metadados (incluindo políticas).

Cada funcionalidade de gerenciamento definida no WSDM é extensível. Novas capacidades podem ser definidas de forma similar. O MUWS apresenta mecanismos, padrões e refinamentos para a definição de novas funcionalidades. Além disso, o mesmo suporta a descoberta, o uso e a identificação dessas novas funcionalidades.

O fato da arquitetura do WSDM focar no recurso gerenciável faz com que não existam restrições nas estratégias de implementação do sistema de gerenciamento. De fato, o WSDM isola o consumidor ou aplicação de gerenciamento dos aspectos de implementação do recurso gerenciável ou Web Service EPR, conforme ilustra a Figura 2.11. Como exemplo, um recurso gerenciável pode ter ou não um agente de gerenciamento. O acesso ao recurso, em ambos os casos, é feito da mesma forma. As informações de como esse recurso foi implementado são, portanto, transparentes para o consumidor.

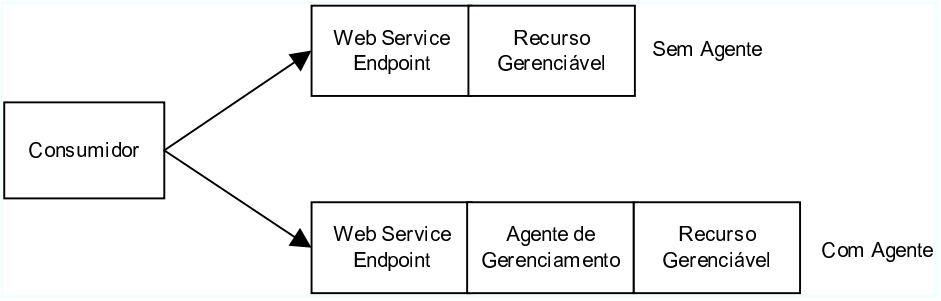


Figura 2.11 – Isolamento de implementação proposto pelo WSDM.

Outra questão importante do WSDM é o suporte à composição. Essa permite que diferentes funcionalidades de gerenciamento possam ser agregadas, de forma gradual, a um recurso. Uma mensagem SOAP enviada a um Web Service, por exemplo,

pode representar uma operação de gerenciamento como a obtenção de informações sobre a carga do *toner* da impressora. De forma semelhante, a mesma mensagem SOAP com cabeçalhos do *WS-Security*, assinados e encriptados, resulta na mesma requisição sendo feita de forma segura. Esse tipo de composição é chamado de composição em nível de mensagem.

Existem dois grupos de composições que podem ser realizadas no WSDM: em nível de implementação do Web Service e referente às funcionalidades específicas do recurso gerenciado, conforme exemplifica a Figura 2.12. Nessa ilustração, as capacidades de gerenciamento específicas do recurso somadas às capacidades de gerenciamento comuns e às disponibilizadas pela plataforma de Web Services dão suporte ao gerenciamento da impressora.

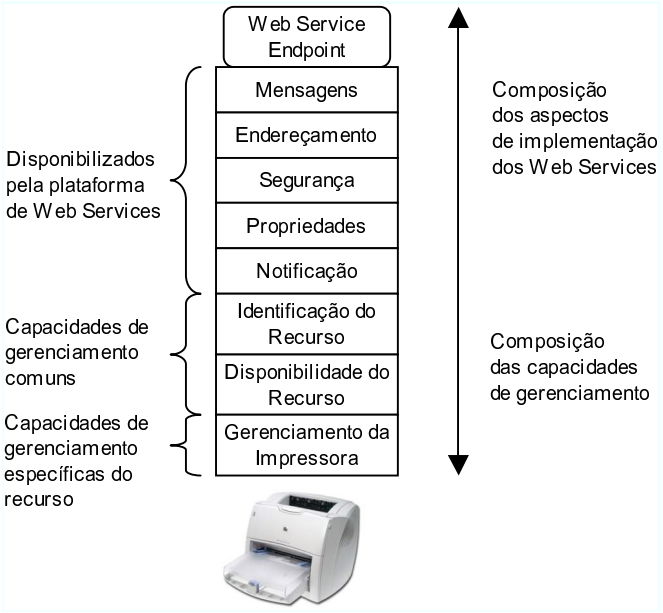


Figura 2.12 – Suporte à composição oferecido pelo WSDM.

A arquitetura do WSDM permite que a proposta possa ser adotada em diferentes *hardwares*. É possível gerenciar desde dispositivos de pequeno porte, como celulares e PDAs, até equipamentos com poder computacional superior, como servidores de aplicação. Tal característica é permitida pela pequena barreira oferecida para o suporte do MUWS. Apenas alguns requisitos são obrigatórios na implementação da proposta. O processo de composição, então, permite que novas funcionalidades possam ser gradualmente introduzidas, de acordo com o poder computacional do dispositivo e com a disponibilidade de recursos.

Conforme descrito anteriormente, a base do MUWS são os Web Services. Um conjunto de especificações desses serviços deve ser agrupado para suportar o gerenciamento de um recurso. Em última análise, as propriedades, as operações, os metadados e os eventos suportam a representação dos aspectos funcionais do recurso.

As *propriedades* caracterizam o funcionamento do recurso. Uma propriedade é descrita através de um *schema* XML e representada por um *Global Element Declaration* (GED). Além do *schema*, a propriedade possui uma descrição da sua semântica, cardinalidade e qualquer metadado relevante. Um recurso gerenciável deve, obrigatoriamente, oferecer um XML que apresente todas as propriedades que suporta. As *operações* definem as ações que o recurso pode executar. As mesmas são descritas através do WSDL (*Web Service Definition Language*) pelo componente *operations*. Já os *metadados* são empregados nas propriedades e nas operações para representação das informações de gerenciamento. Tais metadados podem ser dinâmicos ou estáticos. Os *eventos*, por fim, são uma forma de comunicação assíncrona que pode ser empregada para prover um mecanismo de *publish/subscribe*. Os eventos são representados por uma combinação de um tópico e um conteúdo de mensagem GED. Esses devem ser exclusivos e sua combinação identifica um evento de forma única.

O MUWS faz uso do padrão WSRF (*Web Services Resources Framework*) para representar as informações de um recurso. Para tal, um WS-Resource é criado pela composição de um *endpoint* de gerenciamento com um ou mais recursos gerenciáveis. Já os eventos são descritos com base no WS-BaseNotification [28]. Caso um recurso gerenciável inclua a habilidade de suportar eventos com os consumidores, então a definição das capacidades do recurso deve utilizar o conceito de tópicos, conforme descrito no WS-Topics [33].

Por fim, para acrescentar novas informações de gerenciamento além das existentes no WS-ResourceProperties deve-se utilizar o conceito de metadados. Os dados são extraídos do metadado através de tecnologias de processamento de documentos, tais como XPath. Atualmente, o WSDM não define formas de associar ou acessar conteúdos de metadados.

MOWS

O MOWS define como empregar o MUWS para gerenciar Web Services. O padrão pode ser utilizado de duas formas: (a) criando um novo Web Service capaz de comunicar com o EPR do recurso que está sendo gerenciado e prover, assim, um EPR adicional específico para gerenciar o recurso e (b) utilizar o mesmo EPR do recurso que está sendo gerenciado para oferecer tanto as funcionalidades regulares como as capacidades de gerenciamento propostas pelo MUWS.

2.4.3 WS-Management

De forma semelhante ao WSDM, o WS-Management define uma arquitetura baseada em Web Services para o gerenciamento de aplicações e dispositivos. O mesmo é uma proposta da Sun, Microsoft e outras empresas para criar interoperabilidade entre sistemas de gerenciamento. A Microsoft planeja ter o WS-Management

executando na nova versão Server do sistema operacional Windows. A interoperabilidade é obtida através de especificações de Web Services e formas padrões de expor um conjunto de operações que são primordiais a todos os sistemas de gerenciamento. As funções suportadas são:

- *Discover*: permitir a descoberta de recursos gerenciáveis e a navegação entre eles;
- *Get, Put, Create, Rename* e *Delete*: gerenciar configurações e valores dinâmicos dos recursos;
- *Enumerate*: enumerar o conteúdo de coleções como *logs* e grandes tabelas;
- *Subscribe*: permitir assinaturas por determinados eventos gerados pelos recursos;
- *Execute*: fornecer métodos específicos de gerenciamento com parâmetros de entrada e saída bem definidos.

O WS-Management define o mínimo que deve ser implementado em um sistema de gerenciamento. Uma implementação pode estender o número de funcionalidades e, além disso, escolher em não suportar alguma das funcionalidades previstas na proposta.

A arquitetura do WS-Management é apresentada na Figura 2.13. A mesma exemplifica o gerenciamento de dois PDAs com o emprego de Web Services. Cada um desses PDAs possui um Web Service de gerenciamento, que é responsável por gerenciar outro Web Service implementado com o *framework* .NET. O Web Service de gerenciamento possui um *endpoint* descrito pelo WS-Addressing. A aplicação de gerenciamento, então, utiliza esse EPR para acessar o serviço de gerenciamento e fazer qualquer uma das operações ilustradas.

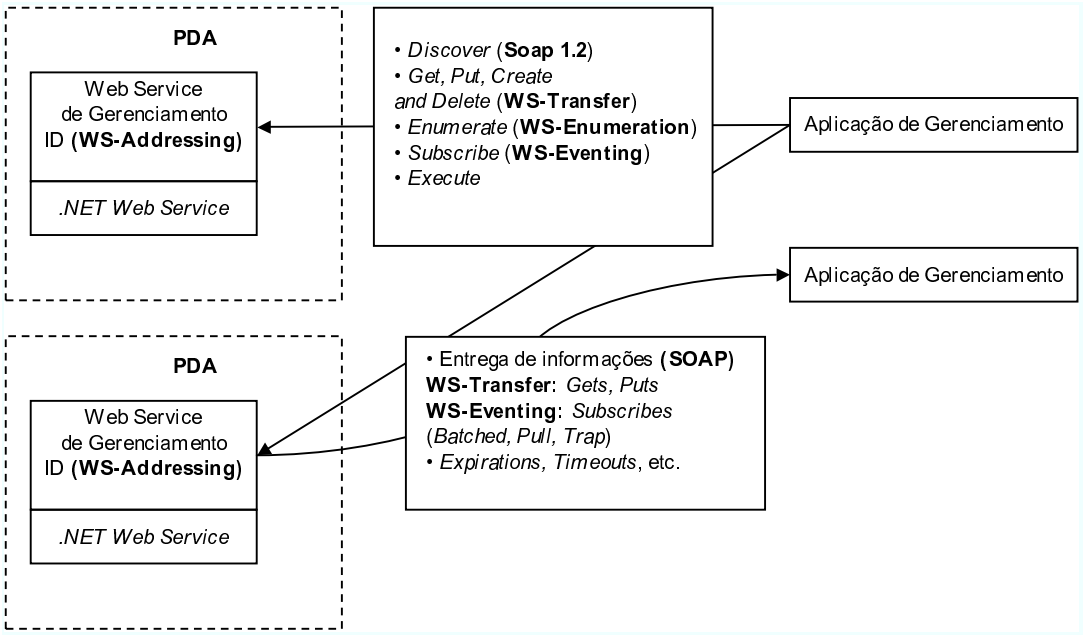


Figura 2.13 – Arquitetura do WS-Management.

Existem, basicamente, três diferenças importantes entre o WSDM e o WS-Management. A primeira delas é que o WS-Management somente se preocupa com o gerenciamento de recursos utilizando Web Services. O WSDM, por outro lado, além de suportar o gerenciamento de recursos com Web Services (MUWS), também oferece uma descrição de como deve ser feito o gerenciamento de Web Services (MOWS).

A segunda diferença é que o WSDM adota um modelo de informações de gerenciamento com a utilização do WS-Resource. O WS-Management, porém, deixa o modelo de informações como uma escolha de critério do desenvolvedor do serviço de gerenciamento. Essa característica pode ser interessante para empresas que queiram desenvolver todo um novo sistema de gerenciamento baseado em um modelo de informações já consolidado, como o CIM (*Common Information Model*) [34], proposto pelo DMTF.

Por fim, a terceira diferença é que o WSDM constitui-se de um padrão aprovado pelo OASIS no ano de 2005. Já o WS-Management ainda não foi submetido para padronização. Por esse motivo o WSDM foi escolhido para ser utilizado na arquitetura de gerenciamento descrita nesta dissertação.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta uma revisão bibliográfica relativa aos principais trabalhos relacionados a esta dissertação. A Seção 3.1 descreve um sistema unificado de gerenciamento que leva em consideração as características próprias de um ambiente de computação móvel. A Seção 3.2 ilustra um sistema de gerenciamento para dispositivos de uma rede sem fio. A Seção 3.3 aborda um *framework* para o gerenciamento de celulares da tecnologia CDMA. Por fim, a Seção 3.4 descreve uma comparação entre as diferentes propostas apresentadas.

3.1 Universal Manager

Adwankar propõe uma solução integrada, denominada *Universal Manager*, para gerenciamento de ambientes de computação móvel [1], que interopera com tecnologias atuais de gerenciamento. A idéia dos autores é prover uma única aplicação capaz de gerenciar um grupo de dispositivos completamente heterogêneo, que vai desde celulares até *notebooks* e *desktops*. A solução adotada foi a criação de um *gateway* capaz de prover uma interface para acesso aos dispositivos, cujo protocolo de comunicação empregado não é compatível com o SNMP.

O *Universal Manager*, portanto, funciona com um *gateway* multi-protocolo que apresenta uma interface de acesso às informações de gerenciamento de dispositivos incompatíveis com o protocolo SNMP. A arquitetura proposta é apresentada na Figura 3.1. Conforme ilustra a figura, o *Universal Manager* atua em diferentes redes. O mesmo pode gerenciar, de forma integrada, desde equipamentos como *switches*, *hosts* e roteadores de uma organização até celulares e diferentes dispositivos móveis que se conectem por outra rede. As corporações mantêm suas MIBs e o sistema de gerenciamento corporativo (*Enterprise Management System - EMS*) provê uma visão completa de toda rede.

No sistema proposto, os dispositivos da organização são encontrados por mecanismos de descoberta e o agente corporativo executa diretamente as ações de

gerenciamento. Para dispositivos de computação móvel, porém, os mecanismos padrões de descoberta não funcionam diretamente. A solução adotada, então, foi fazer com que os mesmos notifiquem sua presença e suas capacidades para um *gateway* multi-protocolo (fluxos 1, 2, 3 e 4).

O gerenciamento de celulares foi realizado com uma implementação em *SyncML* capaz de fornecer informações ao *gateway*. Isso foi feito levando em consideração que os celulares em geral não possuem soluções baseadas em SNMP devido às limitações do seu *hardware*. A solução adotada, portanto, foi continuar utilizando SNMP para comunicar com um *gateway*, e não diretamente com os equipamentos incompatíveis. Nesse caso, o *gateway multi-protocolo* atua como um *proxy* para fornecer acesso aos dispositivos. O *gateway* multi-protocolo, portanto, mapeia o destino da ação para um dispositivo em particular. No caso de celulares, o mesmo atribui um IP para o IMEI (*International Mobile Equipment Identity*) do dispositivo. Sempre que mensagens são enviadas para esse IP, as mesmas são repassadas para o celular através de um protocolo que seja suportado pelo último (fluxos 4, 3, 2 e 1). Caso o aparelho só suporte o serviço de *Short Message Service* (SMS), as operações de gerenciamento são trocadas com SMS. O agente de gerenciamento de terminal (*Terminal Management - TM*) presente no dispositivo é configurado de acordo com suas capacidades. O TM tem acesso a detalhes de *firmware* e pode realizar operações como obter ou configurar dados, instalar imagens, bem como gerenciar falhas e segurança.

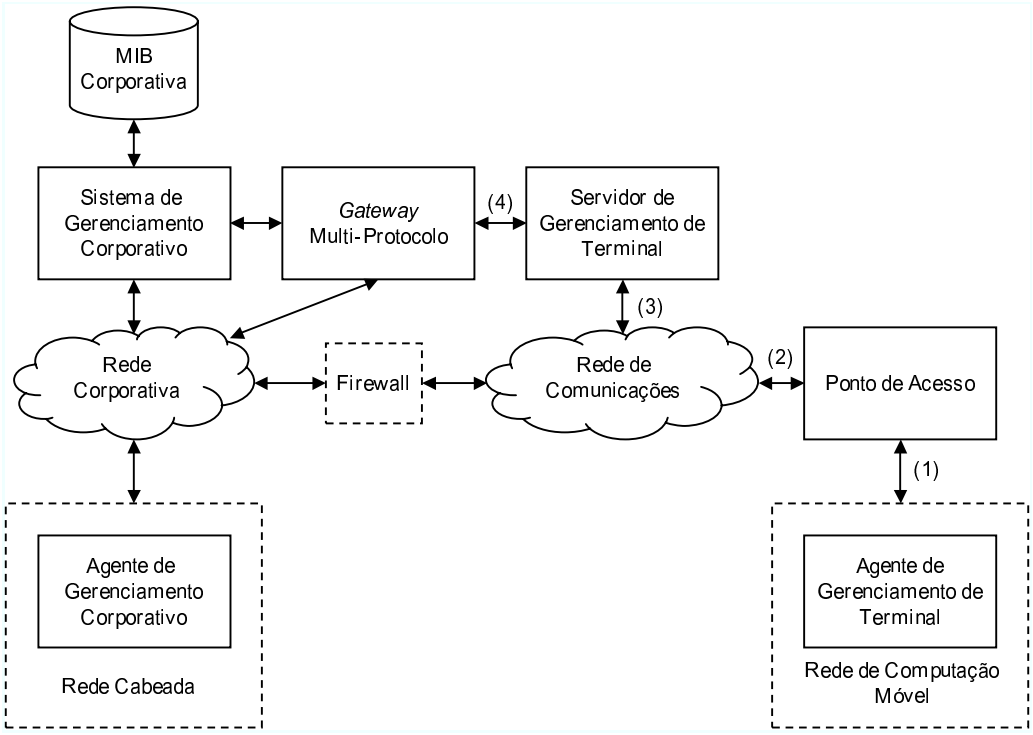


Figura 3.1 – Arquitetura do Universal Manager.

3.2 WLAN-NMS

O WLAN-NMS (*Network Management System for Wireless LAN Service*) [2] é um sistema para o gerenciamento de dispositivos como pontos de acesso, *switches*, *hubs* e roteadores que é capaz de monitorar o estado de cada um desses elementos em tempo de execução. O sistema, implementado na linguagem Java, tem seu foco no gerenciamento de falhas e de configuração. O mesmo foi desenvolvido para ser utilizado em redes sem fio, tentando satisfazer requisitos como alta estabilidade de rede, minimização do tempo de inatividade decorrente de falhas e alta disponibilidade de conexão.

Os diferentes dispositivos passíveis de gerenciamento são apresentados em um mapa da rede construído pela aplicação. Além disso, operações como adição, modificação, remoção, busca e detalhamento dos equipamentos são disponibilizadas por uma interface Web. O estado dos dispositivos é verificado pelo NMS a cada minuto de forma a detectar possíveis falhas. Quando essas ocorrem, as mesmas são ilustradas no mapa de rede e, ao mesmo tempo, reportadas por e-mail para o administrador. Além disso, as informações sobre as falhas podem ser repassadas para o sistema de cobranças.

A arquitetura do WLAN-NMS é ilustrada na Figura 3.2. A aplicação é composta basicamente por dois elementos: um gerente de configuração e um gerente de falhas. O primeiro cria uma *thread* sempre que requisições para adição, remoção ou modificação de elementos gerenciáveis são feitas. Adicionalmente, coleta informações dos dispositivos com o protocolo SNMP e as salva em uma base de dados. Caso ocorra algum problema, o mesmo é reportado por RMI para o gerente de falhas. O gerente de falhas monitora o estado dos dispositivos através de SNMP e PING. As *traps* do protocolo SNMP são empregadas para notificar as possíveis falhas. Por fim, a comunicação com o sistema de cobranças é feita através de *sockets*. Um estudo feito pelo autor aponta que o WLAN-NMS consegue gerenciar, de forma eficiente, até 50.000 elementos de rede.

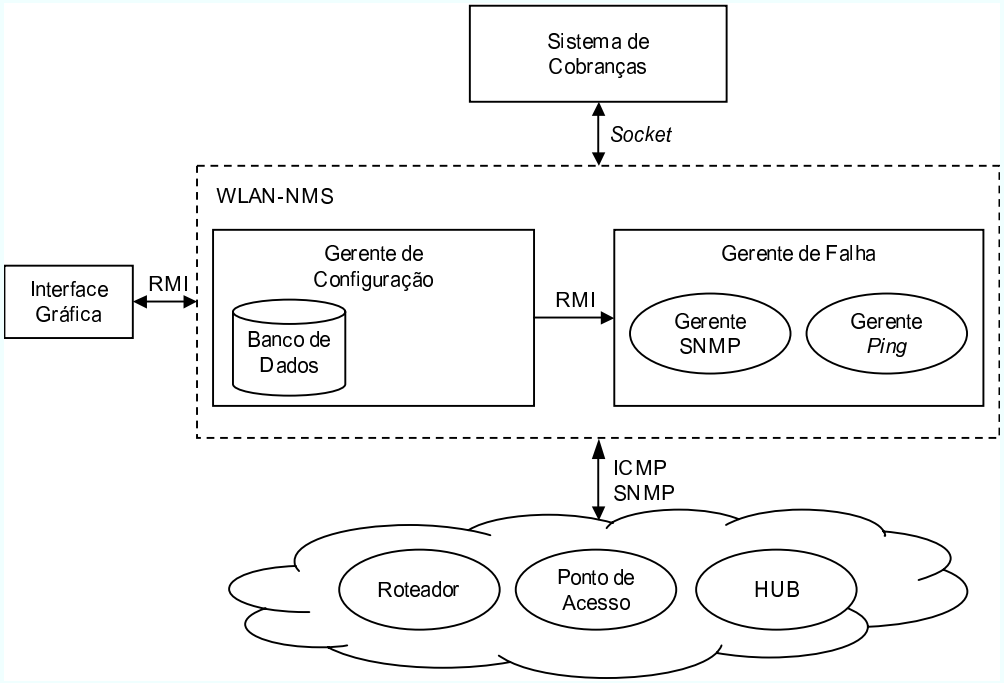


Figura 3.2 – Arquitetura do WLAN-NMS.

3.3 OTAHM

O OTAHM *(Over The Air Handset Management)* [3] é um *framework* para gerenciamento de sistemas celular CDMA através do protocolo WAP. O *framework* oferece funcionalidades como suporte à administração de parâmetros e instalação de softwares nos dispositivos. Tal característica permite que os usuários possam escolher um provedor de serviços, assinar por novos serviços e obter aplicativos. As operadoras, por sua vez, podem empregar o OTAHM para detecção e correção de falhas. As funcionalidades do *framework* podem ser classificadas, basicamente, em:

- OTASP (*Over the Air Service Provisioning*): o OTASP fica nas estações móveis e permite que um usuário assine por um ou mais serviços;
- OTAPA (*Over the Air Parameter Administration*): é responsável pela administração de parâmetros, tais como alteração de configurações dos equipamentos e, até mesmo, configuração de topologias de redes celular devido ao deslocamento do dispositivo (*roaming*);
- OTASD (*Over the Air Software Download*): permite a obtenção de módulos de software sob demanda; alguns serviços, assinados pelo usuário, podem necessitar de um conjunto de aplicações, que são obtidas com o OTASD;
- OTAMD (*Over the Air Mobile Diagnostics*): oferece estatísticas e diagnósticos sobre os diferentes dispositivos, permitindo a descoberta de anormalidades na rede e melhora na qualidade de serviço (QoS).

A arquitetura do OTAHM é ilustrada na Figura 3.3. O sistema funciona através da troca de mensagens entre o servidor OTAHM e uma entidade (OTAHM *Entity*) na estação móvel. O servidor envia e recebe mensagens de gerenciamento e dados através do *gateway* WAP. As mensagens podem ser indicações de serviços com URLs ou dados em WML. A entidade, no equipamento móvel, processa essas mensagens e inicia as ações de gerenciamento necessárias. Além disso, as entidades OTAHM apresentam comunicação com a camada de sinalização, transporte (omitida da figura por motivo de simplicidade) e os objetos gerenciáveis. A utilização de uma interface, para a camada de transporte e sinalização, permite compatibilidade com os diferentes protocolos existentes. Com tal característica, pode-se oferecer gerenciamento similar de um equipamento móvel que troca de uma rede para outra. O servidor fica em uma rede IP e possui uma interface com a rede sem fio. Isso permite o gerenciamento de celulares com protocolos de sinalização em sistemas que suportem o padrão CDMA.

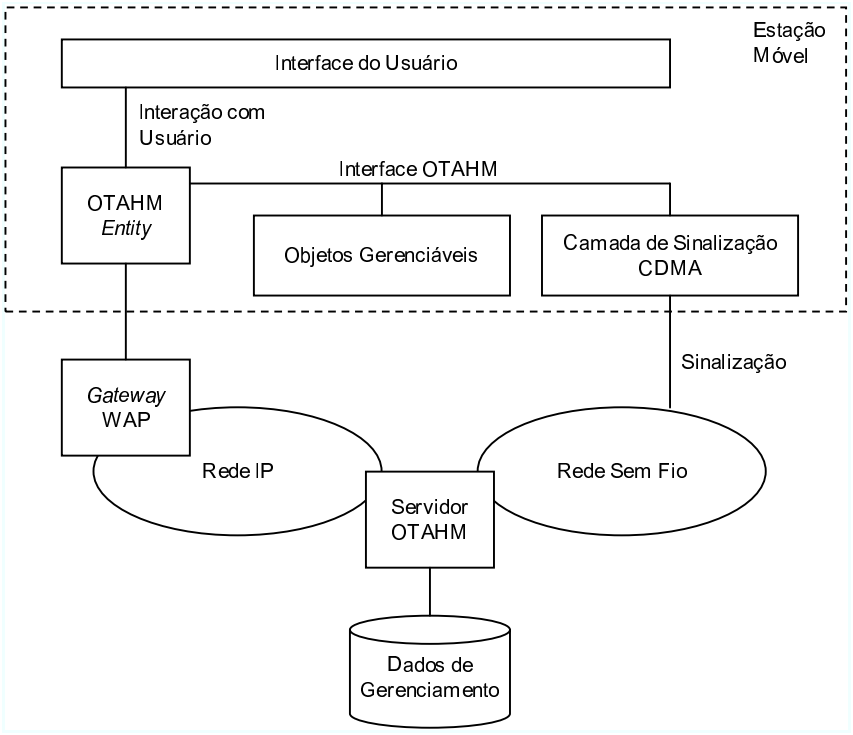


Figura 3.3 – Arquitetura do OTAHM.

3.4 Considerações parciais

A Tabela 3.1 faz algumas comparações entre as três plataformas apresentadas. Com essa, pode-se perceber que as propostas ilustradas utilizam diferentes protocolos para comunicação entre as aplicações de gerenciamento e os recursos que estão sendo gerenciados. Tais protocolos variam desde padrões abertos como o SNMP até

padrões proprietários a exemplo do CDMA.

Uma questão importante que deve ser mencionada é o foco de gerenciamento abordado por cada um dos trabalhos apresentados. Embora se preocupem com a monitoração de uma grande variedade de dispositivos como PDAs, celulares e *notebooks*, nenhum deles trata o gerenciamento das aplicações. Além dessa limitação, as soluções propostas empregam protocolos quase sempre proprietários na comunicação entre as aplicações de gerenciamento e os recursos que estão sendo gerenciados.

As áreas funcionais cobertas por cada um dos trabalhos variam, sendo o *Universal Manager* o mais amplo de todos. Os demais, por outro lado, têm seu foco no gerenciamento de configuração e falhas. Suportar as diferentes áreas funcionais é uma característica importante para oferecer um conjunto mais abrangente de funcionalidades em uma plataforma de gerenciamento.

Por fim, uma característica importante é a integração com sistemas de gerenciamento tradicionais. No caso do *Universal Manager*, o mesmo pode ser integrado com qualquer sistema baseado em SNMP através de seu *gateway* multi-protolo. Os demais trabalhos, porém, constituem soluções mais fechadas que necessitariam da implementação de um *gateway* para serem integradas com os agentes SNMP das redes tradicionais. A possibilidade de integração é um fator que pode ser determinante para a adoção de um sistema de gerenciamento.

	Universal Manager	WLAN-NMS	OTAHM
Protocolos utilizados	SNMP e SMS	SNMP e ICMP	CDMA
Foco de gerenciamento	Celulares, notebooks, PDAs e desktops	Pontos de acesso, swichtes, hubs e roteadores	Celulares CDMA
Suporte ao gerenciamento de aplicações	Não	Não	Não
Áreas funcionais de gerenciamento	Configuração, Contabilização, Desempenho, Falhas e Segurança	Configuração e Falhas	Configuração, Contabilização e Falhas
Integração com sistemas de gerenciamento tradicionais	Sim	Não	Não

Tabela 3.1 – Comparação das diferentes propostas de gerenciamento

Capítulo 4

Arquitetura MobiMan

Identificadas as limitações das diferentes propostas apresentadas no capítulo anterior, este capítulo introduz MobiMan (*Mobile Computing Management Architecture*), uma arquitetura para o gerenciamento de aplicações e recursos em um contexto de computação móvel [35, 36]. A MobiMan, conforme mencionado anteriormente, permite obter e disseminar, de forma flexível, informações sobre o funcionamento dos dispositivos e, sobretudo, das aplicações que executam nos mesmos. É importante ressaltar que, apesar do foco do trabalho ser o gerenciamento de dispositivos móveis, a arquitetura foi projetada de forma genérica o suficiente para que também possa ser empregada no gerenciamento de aplicações em ambientes tradicionais.

A Seção 4.1 descreve uma visão conceitual da arquitetura. A Seção 4.2 discorre sobre os diferentes componentes que fazem parte da mesma. Por fim, a Seção 4.3 apresenta como foi realizada a implementação desses componentes.

4.1 Visão conceitual

O principal elemento da arquitetura é um serviço presente no dispositivo móvel. As aplicações de gerenciamento, então, obtêm informações interagindo com esse serviço, conforme apresenta a Figura 4.1. A mesma mostra uma visão conceitual da arquitetura onde diferentes recursos e aplicações são gerenciados. O primeiro fluxo (1) indica o anúncio dos diferentes elementos gerenciáveis para um elemento intermediário (*service broker*). Esse é consultado pelas aplicações de gerenciamento (fluxo 2) para descobrir quais são os recursos presentes na rede em um determinado momento. Dessa forma, as mesmas podem efetuar operações de gerenciamento. Após a descoberta, as comunicações são feitas diretamente entre a aplicação de gerenciamento e os elementos gerenciáveis. Existem, basicamente, três tipos de comunicações exemplificadas na ilustração: requisição de informações, assinaturas e controle.

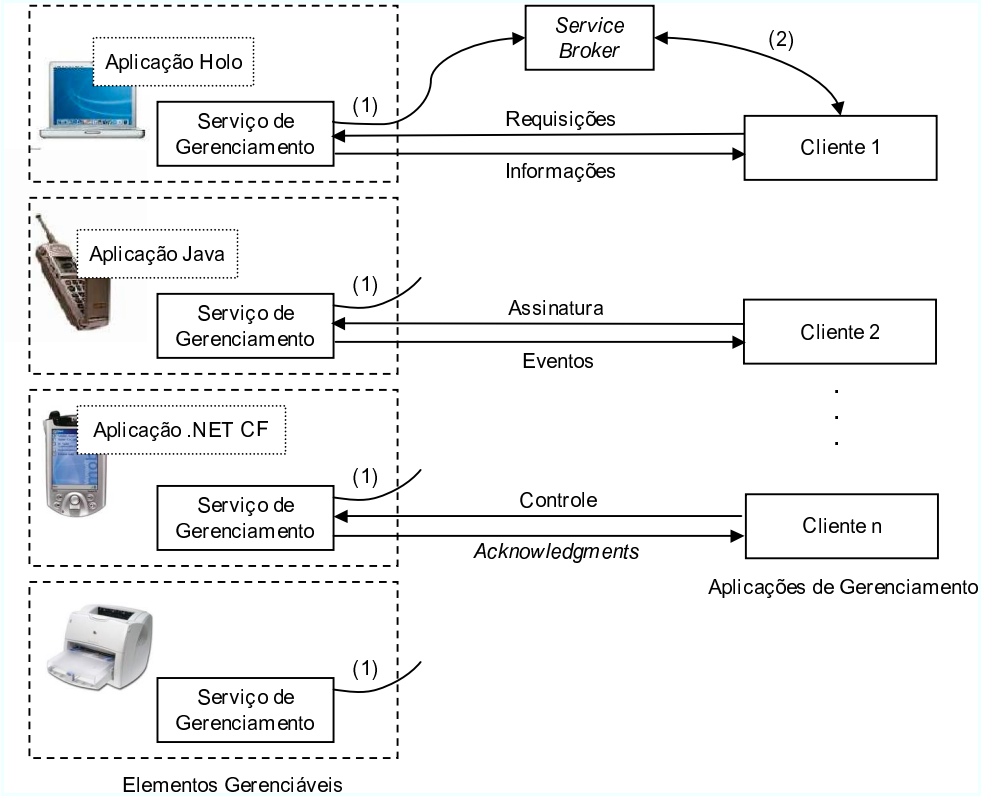


Figura 4.1 – Visão conceitual da arquitetura.

A requisição de informações permite que a aplicação de gerenciamento obtenha qualquer informação em que tenha interesse, de forma síncrona (os pedidos são recebidos e retornados imediatamente). Levando em consideração uma aplicação de videoconferência, por exemplo, poderia-se obter informações que indicassem a qualidade de vídeo e voz.

O segundo fluxo, por outro lado, mostra um mecanismo de *publish/subscribe* que proporciona uma forma assíncrona de requisição e obtenção de informações. Nesse caso, as informações são enviadas através de eventos. Tais eventos podem ocorrer em um determinado intervalo de tempo ou sempre que uma determinada condição seja satisfeita. Ainda se tratando da aplicação de videoconferência, se pode assinar por um evento que reporte alterações na qualidade de vídeo percebida pelo usuário. No momento em que a condição da qualidade do vídeo sofre alteração um evento é gerado de forma assíncrona para o administrador da rede. Dessa forma, o administrador fica ciente dessa dificuldade e pode tomar as decisões necessárias para resolver o problema.

Por fim, o fluxo de controle permite a execução de ações de gerenciamento nos dispositivos e nas aplicações. Esse controle permite que métodos possam ser invocados pelo administrador, de forma a mudar o comportamento da aplicação e dos dispositivos em tempo de execução. No caso da aplicação de videoconferência, o administrador pode solicitar que todas as aplicações diminuam a qualidade do vídeo

durante um período de tempo. Essa medida pode ser utilizada em situações que a rede não permite um tráfego concomitante com o adotado pelas aplicações.

Detalhando um pouco mais o trabalho, a Figura 4.2 apresenta uma visão mais completa da arquitetura. A ilustração mostra os três componentes principais que são imprescindíveis para o seu funcionamento: uma aplicação de gerenciamento, um serviço de gerenciamento (*Mobile Computing Management Service - MobiServ*) e uma API de instrumentação (*Mobile Computing Instrumentation Application Programming Interface - MobiAPI*). A aplicação de gerenciamento é utilizada pelo administrador da rede para interagir com os recursos que estão sendo gerenciados. A mesma está presente em uma máquina remota que será utilizada para gerenciar os dispositivos presentes no ambiente de computação móvel. O MobiServ e a MobiAPI, por outro lado, executam no dispositivo que está sendo gerenciado. O primeiro é responsável por responder às consultas feitas pela aplicação de gerenciamento e invocar ações. As informações necessárias para formulação da resposta, porém, estão armazenadas na MobiAPI. Esta é utilizada para instrumentar a aplicação sobre a qual se tenha interesse em coletar indicadores de funcionamento ou executar ações.

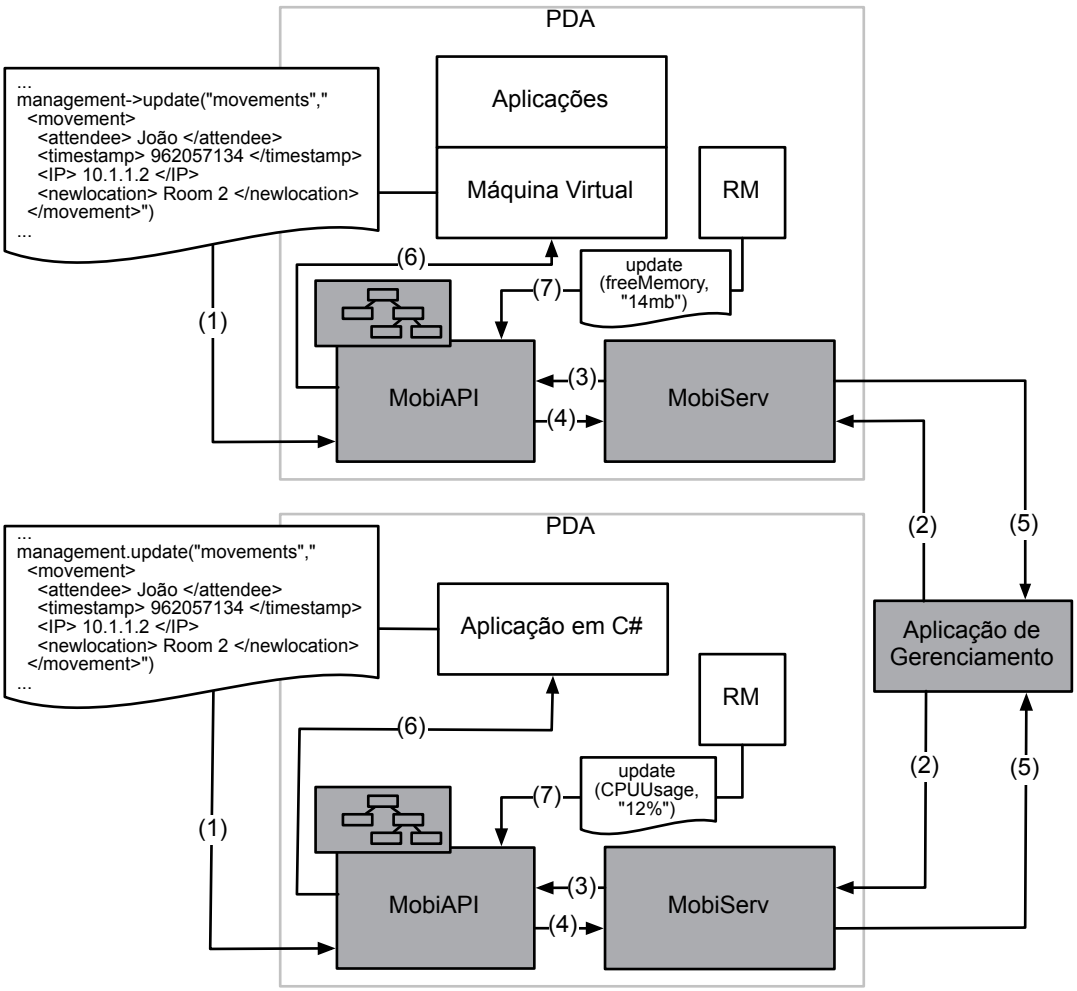


Figura 4.2 – Arquitetura MobiMan.

Duas formas de utilização da arquitetura de gerenciamento são possíveis, como demonstra a ilustração. A primeira forma se dá pela instrumentação de uma máquina virtual (*Virtual Machine - VM*) para fornecer suporte ao gerenciamento. Assim, a execução da aplicação é automaticamente gerenciável sem a adição de código na mesma. A segunda forma representada é a instrumentação de uma aplicação específica, como a apresentada em C#.

O primeiro fluxo ilustrado (Figura 4.2, fluxo 1) representa a invocação de uma função genérica oferecida pela MobiAPI para armazenar indicadores monitorados pela aplicação gerenciada na estrutura de dados local da própria API. A figura ilustra o mesmo método sendo executado tanto por uma máquina virtual, como por uma aplicação em C#. Em ambos os casos é informada a mobilidade de um usuário (João) passando como parâmetros o *timestamp*, o IP e a nova localização (*newlocation*) do usuário.

As diversas aplicações de gerenciamento interagem com a arquitetura através do serviço de gerenciamento presente no dispositivo móvel (fluxo 2). As informações são obtidas consultando a MobiAPI instanciada pela aplicação (fluxos 3 e 4). Após receber as informações da API, o MobiServ produz a resposta de acordo com o padrão WSDM e envia para aplicação que requisitou (fluxo 5). De forma semelhante ocorre a execução de uma ação. Nesse caso, o MobiServ repassa a operação para a MobiAPI. A última, então, se encarrega de notificar a aplicação que será responsável por tratar a solicitação de controle (fluxo 6).

O gerenciamento de recursos é realizado de forma semelhante. Para tal é preciso construir uma aplicação que obtenha os indicadores sobre o funcionamento dos recursos que se tenha interesse em gerenciar. A ilustração mostra um monitor de recursos (*Resource Monitor - RM*) que oferece suporte ao gerenciamento de informações como utilização de memória e processador do dispositivo. Assim como no gerenciamento das aplicações recém mencionadas, os dados também são armazenados na estrutura presente na MobiAPI através de chamadas ao método genérico.

4.2 Componentes

Esta seção detalha os componentes da MobiMan recém introduzida. Ressalta-se que a arquitetura é fundamentada no padrão WSDM [7]. A utilização do mesmo potencializa a integração da arquitetura com outros sistemas de gerenciamento. Nesse contexto, outras especificações associadas como o WS-Notification [28] e o WSRF (*Web Service Resource Framework*) [29] também são intensivamente utilizadas.

4.2.1 *Mobile Computing Management Service - MobiServ*

O MobiServ é um Web Service, presente no dispositivo móvel, que comunica tanto com a aplicação de gerenciamento como com a API de instrumentação. Vale mencionar, que por motivos de simplificação, a implementação desenvolvida desconsidera a existência do *service broker* e faz a comunicação direta entre o Web Service e a aplicação de gerenciamento. As três operações comentadas anteriormente são suportadas pelo serviço.

Os objetos gerenciados seguem o que recomenda o WSDM, ou seja, o serviço oferece capacidades de gerenciamento na forma de um conjunto de tópicos. Esses tópicos são estruturados na forma de uma árvore conforme mostra a Figura 4.3. A figura ilustra uma versão simplificada do modelo de informações proposto para gerenciar o MHolo. Uma versão mais completa é apresentada no Anexo A (Figura A.1). Nessa versão simplificada, estão representadas as entradas necessárias para gerenciar a mobilidade de um ente (*movements*). Além disso, a figura mostra algumas informações que podem ser armazenadas sobre um determinado ente, tais como ID (*beingID*), IP, ID da máquina virtual onde esse ente está executando (*holovmID*), nome do ente (*myName*) e nome do ente pai - utilizado para representar o nome do ente em que está inserido - (*fatherName*).

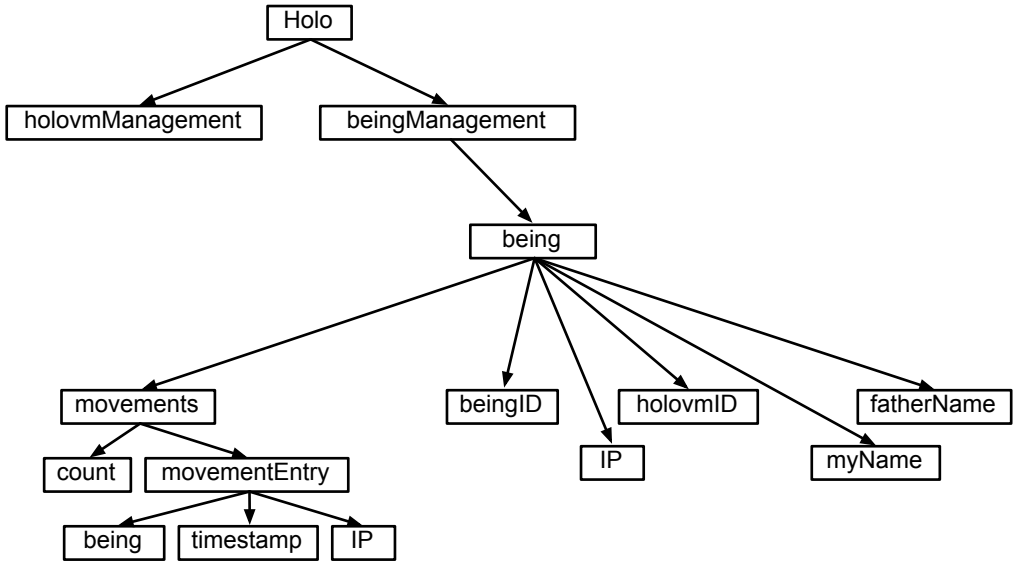


Figura 4.3 – Exemplo do modelo de informações proposto para o MHolo.

Em relação às operações que podem ser executadas sobre esses tópicos, a primeira é a requisição por informações de gerenciamento, feitas pela aplicação de gerenciamento ao MobiServ. As requisições são realizadas seguindo o padrão definido pelo WSRF. As funcionalidades oferecidas pelo MobiServ são: obter ou alterar o valor de uma determinada informação de gerenciamento (*GetResourceProperty ou SetResourceProperty*) e obter um conjunto de informações simultaneamente (*Get-*

MultipleResourceProperties). A Figura 4.4 apresenta uma síntese de como é uma requisição por múltiplas informações de gerenciamento através do método *GetMultipleResourceProperties*. No exemplo, é feita uma solicitação por três informações de gerenciamento presentes no modelo do MHolo recém introduzido (Figura 4.4a). A primeira instância de cada uma dessas informações, então, é retornada através de um XML semelhante (Figura 4.4b).

```
<Envelope>
  <holo:GetMultipleResourceProperties>
     <holo:ResourceProperty> holo:beingID </holo:ResourceProperty>
     <holo:ResourceProperty> holo:IP </holo:ResourceProperty>
     <holo:ResourceProperty> holo:myName </holo:ResourceProperty>
  </holo:GetMultipleResourceProperties>
</Envelope>
```

(a)

```
<Envelope>
  <holo:GetMultipleResourcePropertiesResponse>
     <holo:ResourceProperty> 1 </holo:ResourceProperty>
     <holo:ResourceProperty> 192.168.0.1 </holo:ResourceProperty>
     <holo:ResourceProperty> Holo </holo:ResourceProperty>
  </holo:GetMultipleResourcePropertiesResponse>
</Envelope>
```

(b)

Figura 4.4 – Exemplo de requisição com a primitiva *GetMultipleResourceProperties*.

A segunda operação oferecida é a requisição de informações - referenciadas como tópicos - de forma assíncrona, que se materializa através de assinaturas por tópicos de interesse efetuadas pela aplicação de gerenciamento. As assinaturas seguem o padrão WS-BaseNotification. Como exemplo, a Figura 4.5 ilustra uma assinatura por um tópico *holo:movementEntry*. Esse tópico, também presente no modelo de informações do MHolo, consiste em uma tabela atualizada toda vez que um ente realiza uma movimentação na árvore de execução do Holoparadigma. A mensagem de assinatura (Figura 4.5a) passa como parâmetro o tópico que quer assinar e um endereço (*ConsumerReference*) ao qual serão enviadas notificações sempre que o tópico assinado tiver seu valor alterado. Adicionalmente, se pode definir uma data na qual a assinatura deve expirar (*InitialTerminationTime*). Sempre que uma movimentação for realizada por um ente, então, a aplicação de gerenciamento receberá uma notificação com as informações dessa mobilidade (Figura 4.5b). No caso, o nome do ente, o *timestamp* e o IP da máquina virtual onde ocorreu a movimentação.

A terceira operação suportada é a execução de operações de gerenciamento na aplicação que está sendo gerenciada, aqui referenciada como controle. O controle é oferecido através de uma extensão no padrão do WSDM com a finalidade de suportar a execução de ações de gerenciamento (pela função *ExecuteControl*). Vale mencionar que o WSDM prevê a possibilidade de acrescentar novas funcionalidades

```
<Envelope>
  <wsnt:Subscribe>
    <wsnt:ConsumerReference>
      <wsa:Address> http://192.168.0.254:8080 </wsa:Address>
    </wsnt:ConsumerReference>
    <wsnt:TopicExpression>
      holo:movementEntry
    </wsnt:TopicExpression>
    <wsnt:InitialTerminationTime>
      2006-01-01T00:00:00.00000Z
    </wsnt:InitialTerminationTime>
  </wsnt:Subscribe>
</Envelope>
```

(a)

```
<Envelope>
  <wsnt:Notify>
    <wsnt:NotificationMessage>
      <wsnt:Topic> holo:movementEntry </wsnt:Topic>
        <wsnt:Message>
          <movementEntry>
            <being> Holo </being>
            <timestamp> 962057134 </timestamp>
            <IP> 192.168.0.1 </IP>
          </movementEntry>
        </wsnt:Message>
    </wsnt:NotificationMessage>
  </wsnt:Notify>
</Envelope>
```

(b)

Figura 4.5 – Exemplo de uma assinatura usando a primitiva *Subscribe*.

de gerenciamento sem tornar a plataforma incompatível com outros sistemas. A única consideração importante é que apenas os sistemas que implementem a extensão proposta farão uso do mecanismo de controle.

Outra questão importante do MobiServ é o suporte à utilização de filtros nas requisições por informações de gerenciamento feitas a ele. Dessa forma, pode-se aumentar a inteligência do nodo com o objetivo de reduzir a utilização dos recursos limitados oferecidos pelos dispositivos móveis. Os filtros são utilizados através de consultas por informações com o método *QueryResourceProperties* e a utilização de XPath [37]. Um exemplo de consulta XPath é apresentado na Figura 4.6. A mesma mostra a requisição pela penúltima movimentação realizada na máquina virtual do MHolo. A expressão *last() - 1* é empregada para fazer esse tipo de solicitação. Qualquer outro tipo de condição pode ser utilizada para fazer requisições condicionais por informações de gerenciamento.

```
<Envelope>
  <wsrp:QueryResourceProperties>
    <wsrp:QueryExpression>
      //holo:movementEntry[last() - 1]
    </wsrp:QueryExpression>
  </wsrp:QueryResourceProperties>
</Envelope>
```

Figura 4.6 – Exemplo de requisição com filtros XPath.

4.2.2 *Mobile Computing Instrumentation API - MobiAPI*

A MobiAPI é o módulo da arquitetura presente no dispositivo móvel capaz de suportar a comunicação entre a aplicação gerenciada e o MobiServ. A API tem uma estrutura de dados que segue rigorosamente os tópicos disponibilizados pelo MobiServ. No caso, existe um mapeamento um-para-um das informações disponibilizadas pelo MobiServ com as existentes na estrutura de dados da API, conforme mostra a Figura 4.7. É importante mencionar que a estrutura de dados de gerenciamento está presente na MobiAPI para fazer com que os dados não tenham que ser enviados constantemente para o MobiServ, a não ser que tenham sido solicitados. Além disso, a ilustração aqui apresentada é apenas conceitual. As informações de gerenciamento, na verdade, estão presentes apenas na MobiAPI.

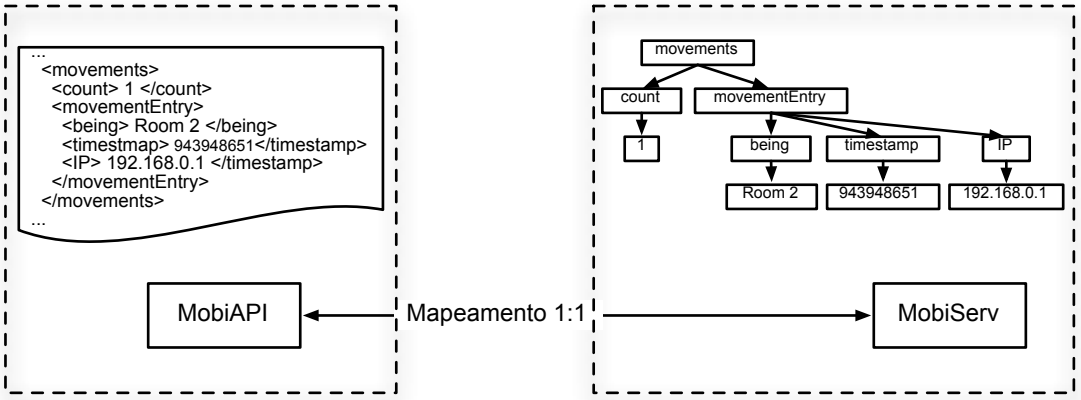


Figura 4.7 – Mapeamento 1:1 entra a MobiAPI e o MobiServ.

A API de instrumentação possui métodos que são acessíveis apenas pela aplicação ou recurso que estão sendo gerenciados e métodos que são acessíveis apenas pelo MobiServ, conforme mostra a Tabela 4.1. Os acessíveis pela aplicação são utilizados para alimentar as informações de gerenciamento presentes na estrutura de dados da MobiAPI. O método *MobiAPIUpdate* recebe o nome de um tópico *QName* e o valor com o qual esse tópico deve ser configurado.

Já os métodos acessíveis pelo MobiServ são empregados para recuperar as informações de gerenciamento junto à estrutura de dados (e para sua disponibilização para aplicação de gerenciamento). Sempre que a aplicação de gerenciamento solicita uma primitiva como *GetResourceProperty*, *GetMultipleResourceProperties* e *QueryResourceProperties*, a MobiAPI invoca a função equivalente da API para responder pela solicitação (respectivamente *MobiAPIGetResourceProperty*, *MobiAPIGetMultipleResourceProperties* e *MobiAPIQueryResourceProperties*). Em relação aos parâmetros, no caso de requisições por informações de gerenciamento únicas, o único parâmetro é o nome do tópico que se tem interesse em obter. A requisição

de um conjunto de informações, por outro lado, requer como parâmetro um vetor de tópicos os quais serão retornados em uma única mensagem. Por fim, o suporte a filtros é oferecido através do método *MobiAPIQueryResourceProperties* que, como argumento, recebe uma consulta do tipo XPath, conforme comentado anteriormente.

Além disso, a API apresenta suporte a execução de ações e configuração de informações de gerenciamento (pelos métodos *MobiAPIExecuteControl* e *MobiAPISetResourceProperty*). Nesse caso, fica a critério da aplicação que está sendo gerenciada como executar as operações de controle ou como tratar as operações de configuração.

Tabela 4.1 – Métodos disponibilizados pela MobiAPI

	Métodos acessíveis pela aplicação gerenciada
MobiAPIUpdate (QName, Novo Valor)	Atualiza informação de gerenciamento na estrutura de dados instanciada em memória no dispositivo móvel pela MobiAPI
	Métodos acessíveis pelo MobiServ
MobiAPIGetResourceProperty (QName)	Retorna o valor de uma determinada informação de gerenciamento
MobiAPIGetMultipleResourceProperties (QName[])	Retorna o valor de um conjunto de informações de gerenciamento
MobiAPIQueryResourceProperties(XPath Query)	Retorna o resultado de uma consulta XPath feita na estrutura de dados presente na MobiAPI
MobiAPISetResourceProperty(QName, Novo Valor)	Informa a aplicação ou recurso que está sendo gerenciado que uma alteração ocorreu
MobiAPIExecuteControl(OperationName, Args[])	Solicita à aplicação que está sendo gerenciada que execute uma determinada ação

4.2.3 Aplicação de gerenciamento

A aplicação de gerenciamento é o *frontend* utilizado para gerenciar as aplicações e recursos presentes no dispositivo móvel. A mesma deve suportar todas as funcionalidades oferecidas pelo MobiServ. Apesar de o exemplo ilustrado anteriormente (Figura 4.2) utilizar uma única aplicação para obter as informações de gerenciamento, pode-se utilizar duas ou mais aplicações. Além disso, um elemento intermediário poderia ser empregado para receber e agregar as informações coletadas dos diferentes recursos garantindo, assim, uma maior escalabilidade. A escalabilidade é uma característica que pode ser importante para adoção, em larga escala, de uma arquitetura de gerenciamento. Tal característica, porém, não é o foco desse trabalho.

A Figura 4.8 apresenta um *screenshot* da aplicação de gerenciamento desenvolvida. Além das funcionalidades descritas no MobiServ, a mesma apresenta suporte

para interromper e continuar uma assinatura por um determinado tópico. É importante mencionar que a aplicação, após fazer a assinatura por um tópico, precisa escutar de forma assíncrona por notificações que informem alterações nos tópicos assinados. Tanto as informações requisitadas como as notificações são exibidas para o usuário na janela de *log* da aplicação.

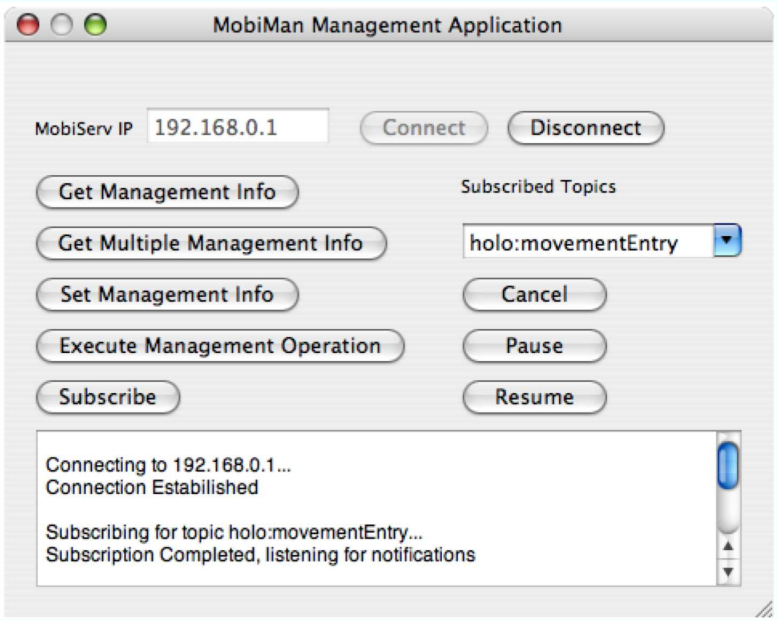


Figura 4.8 – Interface da aplicação de gerenciamento.

4.3 Implementação

O MobiServ foi desenvolvido como um Web Service na linguagem C# utilizando o .NET Compact Framework 2.0 (presente no Visual Studio 2005). O primeiro desafio na implementação de um Web Service foi a necessidade de contar com um servidor Web. Apesar de plataformas de desenvolvimento como o Visual Studio oferecerem esse tipo de suporte para *desktops*, o mesmo não ocorre com dispositivos móveis e o *compact framework* da Microsoft. A solução adotada foi empregar uma implementação de um servidor Web para PDAs feita na Universidade de Monash, Austrália. Essa implementação, disponibilizada na página da Microsoft, foi realizada de forma modular e pôde ser estendida para os propósitos da arquitetura. A implementação do serviço de gerenciamento, então, foi concebida com a adição de um novo módulo denominado WSDMModule.

Todas as configurações do MobiServ ficam presentes em um arquivo chamado *config.web*. Um exemplo desse arquivo pode ser observado na Figura 4.9. O mesmo consiste de um arquivo XML que contém as seguintes informações:

- *ipaddress*: endereço IP do dispositivo que está executando o MobiServ;

- *port*: porta TCP em que o mesmo deseja receber as requisições;
- *webroot*: diretório onde estará presente o conteúdo do servidor Web, como páginas HTML e arquivos WSDM;
- *security*: lista de IPs cujo acesso ao servidor Web não é autorizado;
- *modules*: definição de todos os módulos que devem ser carregados no processo de inicialização do MobiServ e o tipo de arquivo que os mesmos devem tratar.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <settings ipaddress="192.168.0.2" port="8080"
      webroot="\Program Files\CS\wwwroot\" />
   <security>
      <denied>
      </denied>
   </security>
   <modules>
      <module classname="WSDMModule.WSDMProcessor"
       modulepath="\Program Files\CS\WSDMModule.dll">
         <mimeElement mimetype="text/xml" extension="wsdm"/>
      <module>
   </modules>
</configuration>
```

Figura 4.9 – Arquivo de configuração do MobiServ.

O diagrama de classes da implementação do WSDMModule está ilustrado na Figura 4.10. A classe principal é a WSDMProcessor. A mesma herda a classe HttpBaseModule e implementa a interface IHttpModule. Sempre que uma requisição é feita a um arquivo WSDM, a mesma é chamada através de seu construtor. Caso seja uma requisição HTTP do tipo GET ou POST, os métodos ActivateGET e ActivatePOST serão chamados, respectivamente. Por outro lado, caso consista em uma requisição WSDM, como um GetResourceProperty, o método ActivateWSDM será chamado. Todo o processamento das mensagens WSDM é feito pela classe WSDMParser. A mesma, com utilização da classe XMLTools, faz *parsing* na requisição XML e responde com o XML adequado de acordo com as definições do padrão.

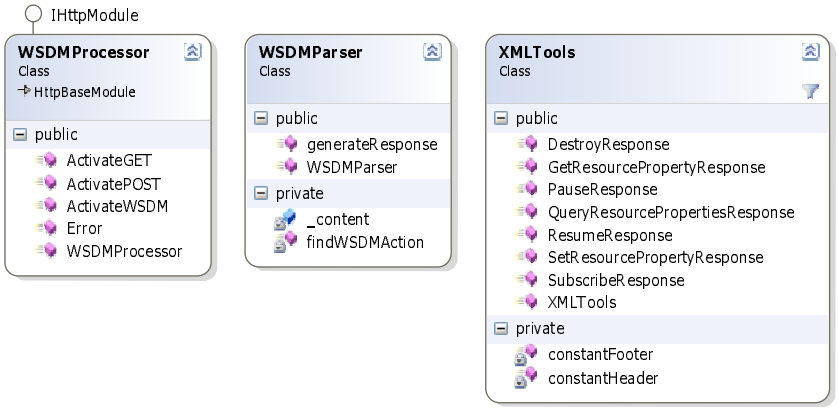


Figura 4.10 – Diagrama de classes.

O segundo componente da arquitetura, a MobiAPI, é uma biblioteca DLL que também foi desenvolvida em C#. A mesma deve ser inicializada pelo elemento gerenciável e comunica com o MobiServ através de comunicação inter-processo COM. O modelo de informações, conforme mencionado anteriormente, está presente na MobiAPI.

O WSDM não define quais informações o recurso deve prover na sua interface de gerenciamento. O padrão define, por outro lado, como expressar um modelo de informações em um *schema* XML e como acessá-lo utilizando Web Services. Dessa forma, um *schema* XML deve ser construído para representar as informações que se tenha interesse em gerenciar. Esse XML deve estar presente em um arquivo que é lido dinamicamente pela MobiAPI. Assim, a mesma cria uma instância desse arquivo na memória do dispositivo e utiliza essa instância para armazenar as informações recebidas da aplicação que estiver sendo gerenciada. Essa estrutura de dados viabiliza operações como *QueryResourceProperties* e *GetResourceProperty*, solicitadas pelo MobiServ.

Um exemplo de *schema* XML é apresentado na Figura 4.11. Esse exemplo representa de forma equivalente o modelo de informações proposto para o MHolo e apresentado anteriormente. O mesmo possui entradas para os entes (*being*) e diversas informações sobre cada um desses entes. Entre elas, pode-se destacar IP, movimentações, nome, entre outras. Cada uma das informações de gerenciamento que serão instanciadas na MobiAPI devem seguir a tipagem informada no modelo. No caso do identificador do ente (*beingID*), por exemplo, os valores aceitos são apenas do tipo inteiro (*xs:integer*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="being">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="movements"/>
      </xs:choice>
      <xs:attribute name="beingID" type="xs:integer"/>
      <xs:attribute name="IP" type="xs:string"/>
      <xs:attribute name="holovmID" type="xs:integer"/>
      <xs:attribute name="myName" type="xs:string"/>
      <xs:attribute name="fatherName" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="movements">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="movementEntry"/>
      </xs:choice>
      <xs:attribute name="count" type="xs:integer"/>
    </xs:complexType>

    <xs:complexType name="movementEntry>
      <xs:attribute name="being" type="xs:string"/>
      <xs:attribute name="timestamp" type="xs:long"/>
      <xs:attribute name="IP" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

Figura 4.11 – Modelo de informações do MHolo representado em um XML *schema*.

O último componente implementado foi a aplicação de gerenciamento. A mesma foi construída com a linguagem Java. As funcionalidades oferecidas pela

aplicação desenvolvida são todas àquelas ilustradas na seção anterior, desde a obtenção de informações simples e múltiplas de forma síncrona e assíncrona, até a execução de operações de controle. A aplicação de gerenciamento foi implementada com a utilização de uma API oferecida pela Apache. É importante mencionar que o emprego dessa API ajuda a testar a interoperabilidade da arquitetura proposta, pois a API também é compatível com o Apache Muse [38].

Capítulo 5

Avaliação Experimental

Com o intuito de avaliar a viabilidade técnica da arquitetura proposta foi instanciado um cenário de gerenciamento. Esse cenário consistiu do desenvolvimento de uma aplicação de computação móvel para o ambiente MHolo e da instanciação da arquitetura de gerenciamento nesse ambiente, feita com a instrumentação da HoloVM. Sobre esse cenário foram realizados experimentos visando identificar a sobrecarga introduzida pelo mecanismo de gerenciamento nos dispositivos típicos de um ambiente de computação móvel.

O capítulo está organizado em duas partes. A primeira apresenta o cenário instanciado, descrevendo a aplicação que foi escolhida, sua modelagem em Holo e as necessidades de gerenciamento existentes. A segunda parte descreve uma avaliação de desempenho da arquitetura de gerenciamento em cima do cenário instanciado.

5.1 Cenário de gerenciamento

Conforme já mencionado, para avaliar a MobiMan foi construída uma aplicação de computação móvel. A aplicação desenvolvida foi um assistente para conferências [10]. Um congresso é geralmente composto por um extensivo conjunto de apresentações e demonstrações que, muitas vezes, ocorrem de forma simultânea. Nesse contexto, o assistente de conferências atua como um *software* capaz de auxiliar o usuário durante esse tipo de evento.

A primeira funcionalidade da aplicação é indicar ao usuário apresentações em que ele possa ter interesse (através de tópicos informados previamente pelo usuário ao sistema). Caso o participante selecione uma determinada apresentação, o assistente deve ser capaz de ajudá-lo a chegar ao local onde a exposição irá ocorrer.

Quando o usuário entra na sala da conferência o assistente oferece um novo conjunto de informações. Entre essas, informações referentes ao apresentador, *URLs* relevantes e o número das páginas, nos anais, do artigo que vai ser apresentado. Além disso, existe uma funcionalidade para anotações; durante a apresentação, o

usuário vê *thumbnails* das lâminas que estão sendo apresentadas e pode tomar notas nas mesmas escrevendo em seu PDA. Essas miniaturas são também usadas para possíveis perguntas, pois o cliente do PDA pode mostrá-las no *datashow* durante seu questionamento. Outra funcionalidade oferecida pelo *software* é informar o usuário que versões de áudio e vídeo da conferência estão sendo gravadas. Dessa forma, o mesmo pode requisitar que essas informações estejam disponíveis para posterior recuperação.

Através do seu PDA, o usuário pode, ainda, verificar onde estão seus colegas e saber se eles estão gostando ou não da apresentação (os usuários informam ao assistente). Com base nessa informação, pode decidir para onde se deslocar caso sua apresentação seja mais curta que as outras apresentações que estão acontecendo no mesmo momento. Por fim, todos os dados anotados no dispositivo portátil (anotações, apresentações, áudios e vídeos) podem ser obtidos via Web.

A aplicação da conferência foi desenvolvida na linguagem Holo para ser executada sobre uma máquina virtual (HoloVM). Uma instanciação da mesma é ilustrada na Figura 5.1. A figura mostra uma aplicação composta por um ente *conferência* dentro do qual existem as diferentes sessões técnicas. Cada sessão possui um *chair* e um autor. Diferentes participantes se encontram dentro do ente *conferência* e das sessões técnicas. São apresentados, então, seis possíveis fluxos de ações sob a perspectiva de um usuário (participante 1) que está participando de uma conferência. O primeiro fluxo representa a mobilidade do participante 1 do ente conferência para dentro do ente sessão 1. O segundo fluxo mostra a obtenção de informações realizada pelo ente participante 1 da história do ente sessão 1. As informações obtidas podem ser, por exemplo, as transparências utilizadas pelo apresentador da sessão. O terceiro fluxo exemplifica uma comunicação direta entre dois participantes do evento. Nesse, o participante 1 envia uma mensagem para o participante 2 informando seu grau de satisfação sobre a sessão 1. O quarto fluxo representa um acesso do participante 1 a sua história, com o objetivo de gravar algumas anotações pessoais. O quinto fluxo, no qual o participante 1 acessa a história da sessão 1, exemplifica uma escrita na história, forma utilizada para modelar um questionamento de um participante ao apresentador. Por fim, o sexto fluxo modela a publicação de informações para a história do evento. Todas as anotações realizadas, por exemplo, podem ser enviadas para a história do ente *conferência* de forma que o usuário possa recuperá-las posteriormente na página Web do congresso.

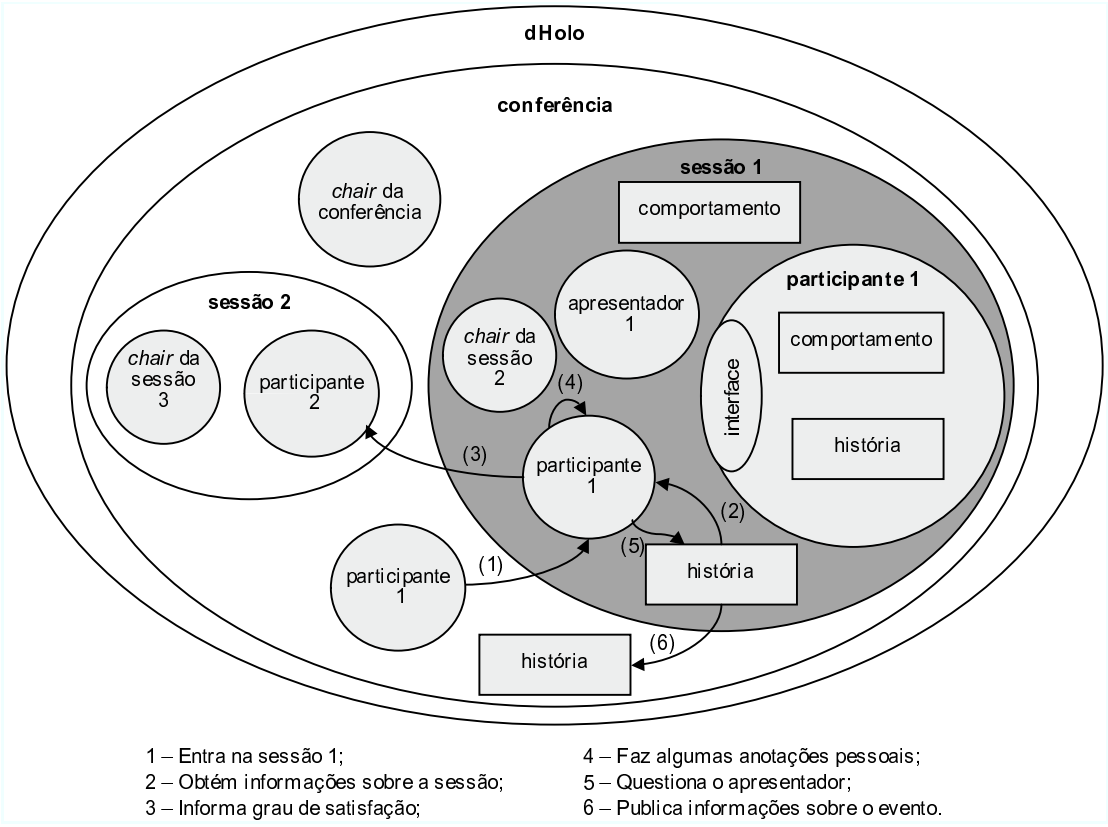


Figura 5.1 – Aplicação modelada em Holo.

Para o cenário recém descrito foram identificadas algumas necessidades de gerenciamento, das quais três são enumeradas a seguir. A primeira consiste em determinar quantos participantes estão presentes em uma determinada sessão técnica. Com essa informação, obtida em tempo de execução, o gerente pode detectar excesso de pessoas em algumas salas e, até mesmo, propor reorganização dos locais que estão sendo utilizados para as diferentes apresentações. A segunda necessidade de gerenciamento identificada é caracterizar a movimentação do usuário entre as sessões técnicas que compõem o evento, permitindo que o administrador tenha uma visão da dinâmica das pessoas. Por fim, a terceira necessidade consiste em gerenciar alguns aspectos de contabilização que permitem avaliar o desempenho da aplicação na recuperação de materiais sobre as sessões técnicas que sejam sensíveis ao tempo, como *streams* de vídeo. Esse tipo de informação permite que o administrador fique ciente se a infra-estrutura disponibilizada é suficiente para suprir as necessidades dos usuários do sistema.

A Figura 5.2 apresenta uma versão simplificada do cenário instanciado com a arquitetura proposta. Nesse cenário existem dois entes participantes (P_1 e P_2). Esses entes estão inicialmente dentro das sessões técnicas, presentes no ente Conferência. Três dispositivos são representados na figura: um servidor (HNS) e dois PDAs. O servidor, conforme mencionado anteriormente, possui uma visão geral de toda a

árvore de entes do Holo. Já os PDAs apresentam apenas o ente que representa o participante. A aplicação de gerenciamento desenvolvida foi configurada para consultar tanto o MobiServ executando no servidor (fluxo 1), como sua instância em cada um dos dispositivos móveis (fluxos 2 e 3). Uma das ações realizadas foi a assinatura pela movimentação dos entes P_1 e P_2 nos diferentes Web Services. Sempre que os participantes entraram ou saíram de uma determinada sessão técnica a informação foi enviada de forma assíncrona para a aplicação de gerenciamento.

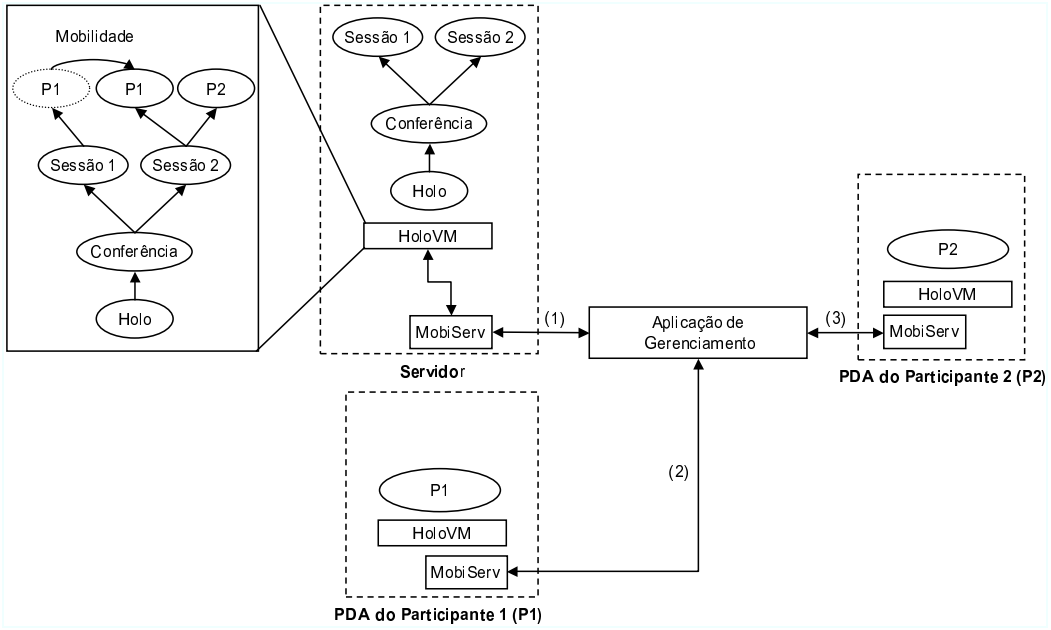


Figura 5.2 – Instanciação da arquitetura em um ambiente real.

A primeira etapa realizada para validar a arquitetura foi instanciar a aplicação em Holo e instrumentar a HoloVM v2.0 para fornecer suporte ao gerenciamento. Tendo o cenário instanciado, a segunda etapa foi utilizar a aplicação de gerenciamento para assinar por informações como *movementEntry*, descrita no modelo de informações do MHolo. Essa informação permitiu não só identificar a movimentação dos participantes nas diferentes sessões técnicas, como também contabilizar o número de usuários em cada uma dessas sessões. Essa última informação foi calculada controlando todas as entradas e saídas dos entes que representam os participantes nas sessões técnicas. Alguns aspectos de contabilização também foram gerenciados nessa aplicação. Os tópicos *averageReadTime* e *averageWriteTime*, descritos na versão completa do modelo MHolo (Anexo A), permitiram obter os tempos médios de leitura e escrita na história de cada ente de sessão técnica. Essa informação pode ser utilizada para, por exemplo, identificar se um determinado usuário está recebendo um vídeo da conferência em um tempo adequado.

5.2 Avaliação de desempenho

Para realizar a avaliação de desempenho restringiu-se o cenário anterior a um dispositivo móvel executando a aplicação de conferência e os respectivos componentes de gerenciamento, conforme ilustra a Figura 5.3. O dispositivo móvel empregado foi um PDA HP iPAQ hx4700 com um processador Intel PXA270 de 624 MHz e 192MB de memória. O *schema* empregado para avaliação contempla todas operações disponibilizadas pela Hololinguagem. O XML foi instanciado na MobiAPI e alimentado pela aplicação gerenciada até totalizar 100 informações de gerenciamento na estrutura de dados da API.

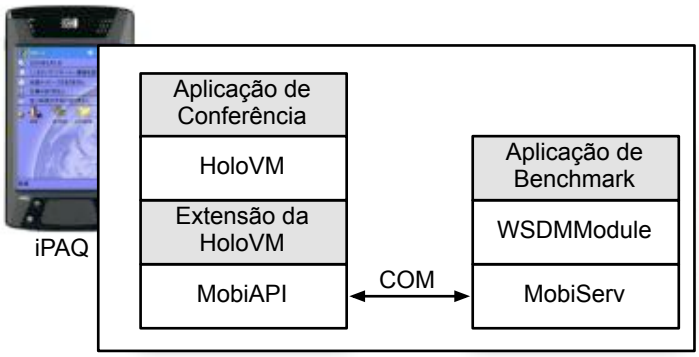


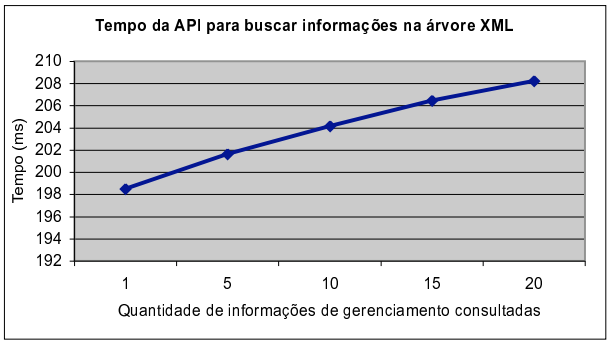
Figura 5.3 – Cenário de gerenciamento empregado na avaliação de desempenho.

A avaliação de desempenho realizada analisou três funcionalidades oferecidas pela arquitetura proposta: obtenção de uma informação de gerenciamento (*GetResourceProperty*); obtenção de várias informações de gerenciamento de forma simultânea (*GetMultipleResourceProperties*); e obtenção de uma informação filtrada por XPath (*QueryResourceProperties*). Sobre as duas primeiras operações foi avaliado o tempo de resposta da API para buscar pela informação solicitada na memória e o tempo do MobiServ para produzir uma resposta em XML seguindo os padrões do WSDM. Na terceira operação, por outro lado, avaliou-se apenas o tempo gasto pela MobiAPI para encontrar uma ou mais informações na estrutura de dados presente na memória.

As medições de tempo (em milisegundos) realizadas foram feitas no dispositivo apenas, sem considerar o tempo consumido pela rede de comunicação. Cada uma das avaliações foi repetida 20 vezes com intuito de obter a média e o desvio padrão dos resultados.

A Figura 5.4 mostra o tempo consumido pela MobiAPI para buscar dados em sua estrutura de dados. O MobiServ marca o tempo antes da requisição pela informação à API e depois que essa informação é retornada. A diferença desse tempo é, então, apontada como o tempo da API para buscar a informação na árvore XML. Os resultados apontam que a arquitetura não consome um tempo que se acredita

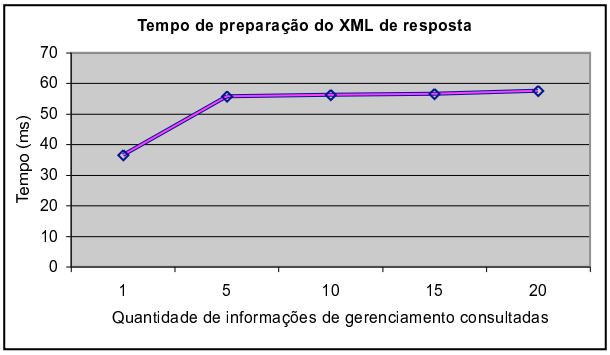
ser excessivo. O tempo máximo foi 208ms para responder por 20 informações de gerenciamento de forma simultânea, o que se considera aceitável em se tratando de um PDA. Outra consideração importante é o crescimento não exponencial dos resultados. O gráfico aponta que o tempo é muito próximo independente do número de objetos solicitados. Acredita-se que isso se deve à maneira como os mecanismos de busca em XML foram implementados pelo .NET CF. Tendo instanciado o XML em memória, as buscas por múltiplas informações de gerenciamento não consomem um tempo muito maior do que por uma única. Tal fator potencializa o uso da arquitetura, pois oferece uma forma eficiente de obter um conjunto de informações do serviço de gerenciamento sem gerar uma sobrecarga grande nos recursos do PDA.



Informações consultadas	Tempo da API	
	Média	Desvio Padrão
1	198,50	2,04
5	201,65	2,89
10	204,15	1,87
15	206,45	2,39
20	208,25	2,15

Figura 5.4 – Tempo consumido pela MobiAPI para buscar informações na árvore XML.

A Figura 5.5 mostra o tempo para o MobiServ preparar a resposta à aplicação de gerenciamento. Essa preparação é feita com base na resposta da API seguindo o padrão do WSDM. Pode-se perceber que, assim como na figura anterior, os tempos são consideravelmente pequenos. Outra questão importante a ser destacada é que existe um custo inicial para produzir respostas de duas ou mais informações. Esse custo, porém, é muito similar para cinco ou mais informações de gerenciamento.

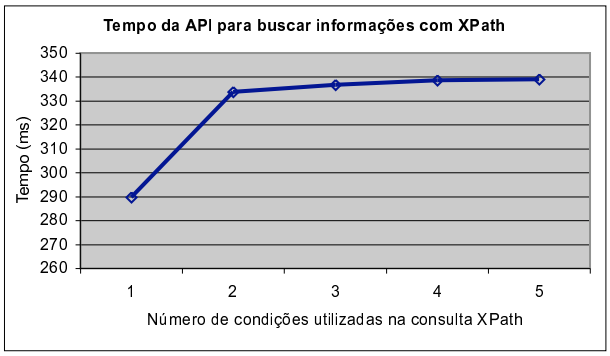


Informações consultadas	Tempo de preparação	
	Média	Desvio Padrão
1	36,55	1,90
5	55,80	1,54
10	56,30	1,66
15	56,60	1,88
20	57,65	3,03

Figura 5.5 – Tempo consumido pelo MobiServ para produzir o XML de resposta.

A Figura 5.6 ilustra o desempenho da API para responder a consultas XPath. Tais consultas são feitas através do suporte oferecido pelo .NET CF 2.0 para buscas

em estruturas XML instanciadas em memória. O teste foi feito variando o número de condições de filtragem entre 1 e 5. A ilustração aponta que esse tipo de construção não consome os recursos do dispositivo de forma impactante e, ao mesmo tempo, possibilita ao administrador escolher as informações de que necessita sem gerar uma sobrecarga grande nos recursos do dispositivo móvel. Uma última consideração sobre essa análise é que a partir de duas condições o tempo consumido pela MobiAPI é similar.



Número de condições	Tempo da API	
	Média	Desvio Padrão
1	289,63	2,43
2	333,79	2,32
3	336,63	2,34
4	338,53	1,90
5	339,11	2,79

Figura 5.6 – Tempo consumido pela MobiAPI para responder por consultas XPath.

Capítulo 6

Considerações Finais

Em ambientes de computação móvel, onde cada vez mais encontra-se dispositivos portáteis executando um extensivo número de aplicações, o gerenciamento é fundamental. A obtenção de indicadores de funcionamento dos dispositivos e, sobretudo, das aplicações, podem ser decisivos para assegurar o bom funcionamento do ambiente em questão.

As soluções existentes para o gerenciamento de ambientes de computação móvel têm seu foco em infra-estrutura. O gerenciamento das aplicações, nesse caso, acaba sendo negligenciado. Além disso, muitos dos mecanismos empregados por essas soluções para comunicação com os dispositivos são proprietários, dificultando sua interoperação com sistemas de gerenciamento já consagrados.

Esta dissertação apresentou MobiMan, uma arquitetura orientada a serviços Web para o gerenciamento de ambientes de computação móvel. A arquitetura permite gerenciar, de forma flexível, informações sobre o funcionamento dos dispositivos e das aplicações que executam nos mesmos. Em consonância com tendências atuais de gerenciamento, a MobiMan foi desenvolvida com base no padrão WSDM. Esse padrão potencializa a integração da arquitetura com outros sistemas de gerenciamento. Outra consideração importante é que a mesma pode ser utilizada para instrumentar máquinas virtuais, garantindo assim uma forma transparente de gerenciar as aplicações já implementadas para executar sobre as mesmas, a exemplo da instrumentação realizada na HoloVM.

Embora os resultados obtidos não sejam conclusivos, é importante enfatizar que a arquitetura não gera uma sobrecarga muito acima do esperado no PDA, satisfazendo a necessidade de ser enxuta e não onerar de forma considerável os recursos desses dispositivos. Um aspecto que chamou atenção é que, a partir de um certo número de informações de gerenciamento, não existe uma diferença significativa no tempo consumido para obter as informações da MobiAPI. Além disso, pôde-se perceber que o emprego de XPath consumiu um tempo similar às consultas por informações sem filtragem. Esse tipo de construção é importante, pois permite que

o administrador obtenha apenas as informações que realmente lhe interessam sem sobrecarregar a rede e os dispositivos dos clientes.

Como trabalho futuro pretende-se aprofundar os experimentos realizados para precisar melhor a sobrecarga gerada pela MobiMan nos dispositivos típicos de um ambiente de computação móvel. Além disso, novas aplicações em Holo devem ser implementadas para certificar o correto funcionamento da instrumentação feita na HoloVM. Por fim, outra questão importante seria instrumentar uma outra máquina virtual ou aplicação para identificar a generalidade da solução proposta nesta dissertação.

Bibliografia

[1] S. Adwankar, S. Mohan, and V. Vasudevan. Universal manager: Seamless management of enterprise mobile and non-mobile devices. *IEEE International Conference on Mobile Data Management (MDM04)*, pages 320–331, 2004.

[2] Boo. Jeon, E. Ko, and G. H. Lee. Network management system for wireless lan service. *10th International Conference on Telecommunications (ICT)*, 2:948–953, 2003.

[3] P. Oommen. A framework for integrated management of mobile-stations over the air. *Integrated Network Management Proceedings (IEEE/IFIP)*, pages 247–256, 2001.

[4] J. L. V. Barbosa, A. C. Yamin, P. K. Vargas, I. Augustin, and C. F. R. Geyer. Holoparadigm: a multiparadigm model oriented to development of distributed systems. *International Conference on Parallel and Distributed Systems (ICPADS)*, pages 165–170, 2002.

[5] Robert Grimm. *System support for pervasive applications.* PhD thesis, University of Washington, December 2002.

[6] Microsoft. *The .NET Compact Framework – Overview.* Microsoft Corporation, http://msdn.microsoft.com/vstudio/device/compactfx.asp, April 2002. Último acesso em Janeiro de 2006.

[7] OASIS. OASIS Web Services Distributed Management (WSDM) homepage, 2005. Disponível em: http://www.oasis-open.org/. Último acesso em Janeiro de 2006.

[8] Robert Grimm. One.world project home page, 2005. Disponível em: http://cs.nyu.edu/rgrimm/one.world/. Último acesso em Janeiro de 2006.

[9] Jorge L. V. Barbosa. *Holoparadigma: Um Modelo Multiparadigma Orientado ao Desenvolvimento de Software Distribuído.* Tese (doutorado em ciência da computação), Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002. 213p.

[10] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction Journal (HCI)*, 16:2–4, 2001.

[11] H. G. Hegering, A. Küpper, C. Linhoff-Popien, and H. Reiser. Management challenges of context-aware services in ubiquitous environments. *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 246–259, 2003.

[12] Archan Misra, Subir Das, and McAuley Anthony. Autoconfiguration, registration, and mobility management for pervasive computing. *IEEE Personal Communications*, 8(4):24–31, August 2001.

[13] P. J. McCann and G. C. Roman. A mobile-unity specification of the mobile IP protocol. Technical Report WUCS-96-15, Washington University, Department of Computer Science, 1996.

[14] M. Chakrabarty, D. Saha Misra, and et al. A comparative study of existing protocols supporting IP mobility.

[15] M. Balazinska and P. Castro. Characterizing mobility and network usage in a corporate wireless local-area network. *MobiSys 2003 - The First International Conference on Mobile Systems, Applications, and Services. San Francisco, California*, pages 303–316, 2003.

[16] A. Balachandran, G. Voelker, P. Bahl, and P. Rangan. Characterizing user behavior and network performance in a public wireless lan. *ACM SIGEMTRICS*, pages 195–205, 2002.

[17] D. Schwab and R. B. Bunt. Characterising the use of a campus wireless network. *Infocom 2004*, pages 862–870, 2004.

[18] T. Buchholz, A. Küpper, and M. Schiffers. Quality of context: What it is and why we need it. In *Proceedings of the Workshop of the HP OpenView University Association 2003 (HPOVUA 2003)*, 2003.

[19] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, third edition, 1997.

[20] William Stallings. Snmp and snmpv2: The infrastructure for network management. *IEEE Communications Magazine*, 36 n. 3:37–46, 1998.

[21] Mark Miller. *Managing Internetworks with SNMP*. USA: M&T Books, second edition, 1997.

[22] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta. On management technologies and the potential of web services. *IEEE Communications Magazine*, 42:58–66, 2004.

[23] Aiko Pras, Thomas Drevers, Remco van de Meent, and Dick Quartel. Comparing the performance of snmp and web services-based management. *eTransactions on Network and Service Management*, 1(2):11, 2004.

[24] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web services architecture. *IBM Systems Journal*, 41:170–177, 2002.

[25] W3C: World-Wide Web Consortium. Disponível em: http://www.w3.org/. Último acesso em Janeiro de 2006.

[26] Thomas Erl. *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, April 2004.

[27] Steve Graham. *Web Services Addressing*. Akamai, Fujitsu, CA Intl, HP Developement Company, IBM, SAP, SSC, Univ. Chicago, Tibco, http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/, 1.0 edition, 10 August 2004. Último acesso em Janeiro de 2006.

[28] Steve Graham. *Web Services Base Notification*. Akamai, Fujitsu, CA Intl, HP Developement Company, IBM, SAP, SSC, Univ. Chicago, Tibco, http://www.ibm.com/developerworks/library/ws-resource/ws-basenotification.pdf, 1.0 edition, 5 March 2004. Último acesso em Janeiro de 2006.

[29] I. et al Foster. Modeling and managing state in distributed systems: the role of OGSI and WSRF. In *Proceedings of the IEEE*, volume 93, pages 604–612, 2005.

[30] William Vambenepe. MUWS Specification Part 1, 2005. Disponível em: http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf. Último acesso em Janeiro de 2006.

[31] William Vambenepe. MUWS Specification Part 2. Disponível em: http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf. Último acesso em Janeiro de 2006.

[32] Igor Sedukhin. MOWS Specification, 2005. Disponível em: http://docs.oasis-open.org/wsdm/2004/12/wsdm-mows-1.0.pdf. Último acesso em Janeiro de 2006.

[33] Steve Graham. *Web Services Topics.* Akamai, Fujitsu, CA Intl, HP Developement Company, IBM, SAP, SSC, Univ. Chicago, Tibco, http://www.ibm.com/developerworks/library/ws-resource/ws-topics.pdf, 1.0 edition, 5 March 2004. Último acesso em Janeiro de 2006.

[34] Winston Bumpus, John W. Sweitzer, Patrick Thompson, Andrea R. Westerinen, and Raymond C. Williams. *Common information model: implementing the object model for enterprise management.* John Wiley & Sons, Inc., New York, NY, USA, 2000.

[35] Fernando Costa, Luciano Paschoal, Gerson Cavalheiro, Jorge Barbosa, and José Dirceu. Towards a framework for mobile, context-aware applications management. *IEEE International Workshop on Management Issues and Challenges in Mobile Computing (MICMC)*, 2005.

[36] Fernando Costa and Luciano Paschoal Gaspary. Gerenciamendo flexível de ambientes de computação móvel através de uma arquitetura baseada em serviços web. *Submetido ao Simpósio Brasileiro de Redes de Computadores (SBRC)*, 2006.

[37] W3C Recommendation. XML Path Language (XPath) Version 1.0, November 1999. Disponível em: http://www.w3.org/TR/xpath. Último acesso em Janeiro de 2006.

[38] Apache Software Foundation. Apache Muse. Disponível em: http://ws.apache.org/muse/. Último acesso em Janeiro de 2006.

Apêndice A

Modelo de Informações de Gerenciamento do MHolo

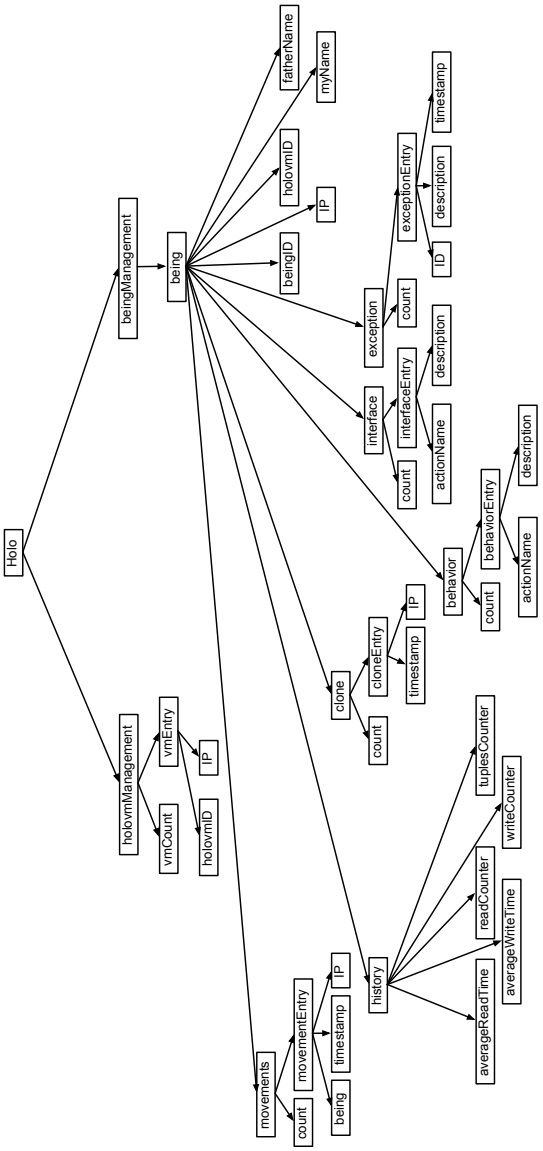


Figura A.1 – Modelo de informações proposto para o MHolo.