UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Infra-estrutura para o Desenvolvimento de Aplicações Pervasivas Auto-configuráveis

Mário Hozano Lucas de Souza

Campina Grande, Paraíba, Brasil
Maio de 2008

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Infra-estrutura para o Desenvolvimento de Aplicações Pervasivas Auto-configuráveis

Mário Hozano Lucas de Souza

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande - Campus I, como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Angelo Perkusich
(Orientador)
Hyggo Oliveira de Almeida
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S725i

2008 Souza, Mário Hozano Lucas de

Uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis / Mário Hozano Lucas de Souza – Campina Grande: 2008.

74f.: il. col.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientadores: Angelo Perkusich e Hyggo Oliveira de Almeida

1. Computação Pervasiva. 2. Computação Autônoma. 3. Adaptação 4. Auto-configuração. I-Título

CDU 004.416.3(043)

"UMA INFRA-ESTRUTURA PARA O DESENVOLVIMENTO DE APLICAÇÕES PERVASIVAS AUTO-CONFIGURÁVEIS"

MÁRIO HOZANO LUCAS DE SOUZA

DISSERTAÇÃO APROVADA EM 05.05.2008

PROF. ANGELO PERKUSICH, D.Sc
Orientador

PROF. HYGGO OLIVEIRA DE ALMEIDA, D.Sc
Orientador

PROF. LEANDRO DIAS DA SILVA, D.Sc
Examinador

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr.
Examinador

CAMPINA GRANDE – PB

Resumo

No paradigma de computação pervasiva, a automatização de operações favorece a realização de uma das principais características encontradas nos ambientes inteligentes: a invisibilidade. Enquanto isso, o paradigma da computação autônoma tem apresentado significativos avanços na concepção de sistemas com a capacidade de auto-gerência. Nesse sentido, a utilização de aspectos da computação autônoma em aplicações pervasivas permite automatizar tarefas cuja necessidade pode ser verificada por informações contextuais. Entretanto, a heterogeneidade e a dinamicidade dos ambientes pervasivos revelam alguns problemas que devem ser considerados na automatização destas tarefas. Neste trabalho, propõe-se uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis, que reúne aspectos de computação pervasiva e computação autônoma. Esta infra-estrutura apresenta uma arquitetura extensível onde diversos mecanismos de raciocínio podem ser utilizados na adaptação do ambiente a partir de informações contextuais. Como prova de conceito, um estudo de caso foi implementado para prover auto-configuração no Laboratório de Sistemas Embarcados e Computação Pervasiva.

Abstract

In the pervasive computing paradigm, the automation of tasks promotes an important feature of smart environments: the invisibility. On the other hand, researches on autonomic computing have achieved significant advances in the design of self-management systems. In this context, improving pervasive computing systems with autonomic features allows automating tasks based on contextual information. However, the heterogeneity and dynamism of pervasive environments create some challenges that need to be considered in the automation of those tasks. In this work, it is proposed an infrastructure for the development of self-configuring pervasive computing applications, which brings together techniques of pervasive and autonomic computing. Such infrastructure provides an extensible architecture where reasoning mechanisms can be used to adapt the environment based on contextual information. As proof of concept, an application example has been implemented to provide self-configuring for the Embedded Systems and Pervasive Computing Laboratory environment.

Agradecimentos

Agradeço ao meu pai José Ferreira, por personificar o “exemplo a seguir” nas situações adversas que passei. À minha mãe Elza, pela sua irreverência e bondade ilimitada, não medindo esforços para garantir o sorriso e a felicidade de seus filhos.

Aos meus irmãos Arlan, que, apesar da distância, foi o principal motivador para a minha entrada neste curso; Kelton, que me mostrou que em algumas situações da vida a teoria é diferente da prática, e que a ousadia e a confiança podem alcançar objetivos praticamente inatingíveis; Duda, que tem me ensinado a ser uma pessoa menos egoísta e mais prestativa e; Guenda, que me ensinou a nunca desistir de meus objetivos.

À Teófila, minha segunda mãe, que sempre soube fazer as melhores comidas de acordo com as minhas restrições alimentícias.

Aos meus sobrinhos Igor e Mateus que sempre perderam de mim nos jogos de computador, e os mais novos, Breno e Débora, que também perderão de mim quando crescerem.

À minha namorada Berel, por suportar minhas chatices nesta minha vida de cigano com idas e vindas entre Maceió-Campina Grande.

Agradeço também aos meus amigos Satoro, Henrique, Théo e Ricardinho, que, juntamente comigo, foram protagonistas de diversas histórias que contarei aos meus filhos (que nascerão daqui a uns 10 anos).

Aos meus amigos Felipe (Vigia), Leandro Sales, Glauber, Marcos Fábio, Leandro Cabelo, Fred, Milena e Xambinho que fizeram os dias em Campina se tornarem um pouco mais interessantes com reuniões, quebra-cabeças, futebol e cervejadas.

Aos meus orientadores Angelo Perkusich e Hyggo Almeida, por me darem a oportunidade de trabalhar num ambiente como o Embedded. Aos meus novos amigos Olympio, Zé Luís, Danilo, Ádrian, Yuri, Diego e André Felipe, que durante dois anos, agüentaram as minhas piadas no laboratório.

Agradeço à CAPES pelo apoio financeiro.

Por fim, agradeço a todas as pessoas que contribuíram direta ou indiretamente para que eu chegasse até aqui.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Nos últimos anos tem-se observado um grande avanço das tecnologias para comunicação sem fio. A cada dia se torna mais fácil encontrar objetos do dia a dia que apresentam tecnologias como *Bluetooth* [6] e *Wi-Fi*[1] [41], favorecendo a comunicação entre os aparelhos eletrônicos. Dessa forma, aparelhos como televisores, aparelhos de som, lâmpadas, etc., contidos numa sala ou escritório, podem ser manipulados por outros dispositivos que não estejam fisicamente conectados aos mesmos.

Além de estarem presentes em aparelhos com nenhuma ou baixa mobilidade, as tecnologias de comunicação sem fio vêm sendo amplamente utilizadas em dispositivos portáteis como *PDAs*[2], *Internet tablets* e aparelhos celulares. Este nicho de dispositivos móveis conta ainda com um grande esforço na miniaturização de seus componentes e em pesquisas relacionadas ao gerenciamento de bateria, memória e processamento [3; 49; 32; 9].

Ao mesmo tempo em que *PDAs*, *Internet tablets* e celulares apresentam características comparáveis aos computadores tradicionais, a disputa entre as montadoras destes aparelhos juntamente com a concorrência entre as inúmeras operadoras de telecomunicação, favorece o barateamento e, conseqüentemente, a popularidade dos mesmos no mundo atual. Tal popularidade é comprovada por pesquisas realizadas pelo Instituto Brasileiro de Geografia e Estatística (IBGE). Na Pesquisa Nacional por Amostras de Domicílios em 2006, foi verificada a evolução da posse de telefone móvel celular, registrando números que superam a

[1] *Wireless Fidelity*

[2] *Personal Digital Assistants*

posse de telefone fixo nas residências brasileiras. Como apresentado no gráfico da Figura 1.1, em 2006, 63,3% das residências brasileiras possuíam um aparelho celular, enquanto apenas 10,9% das residências possuíam apenas o suporte para telefonia fixa[3].

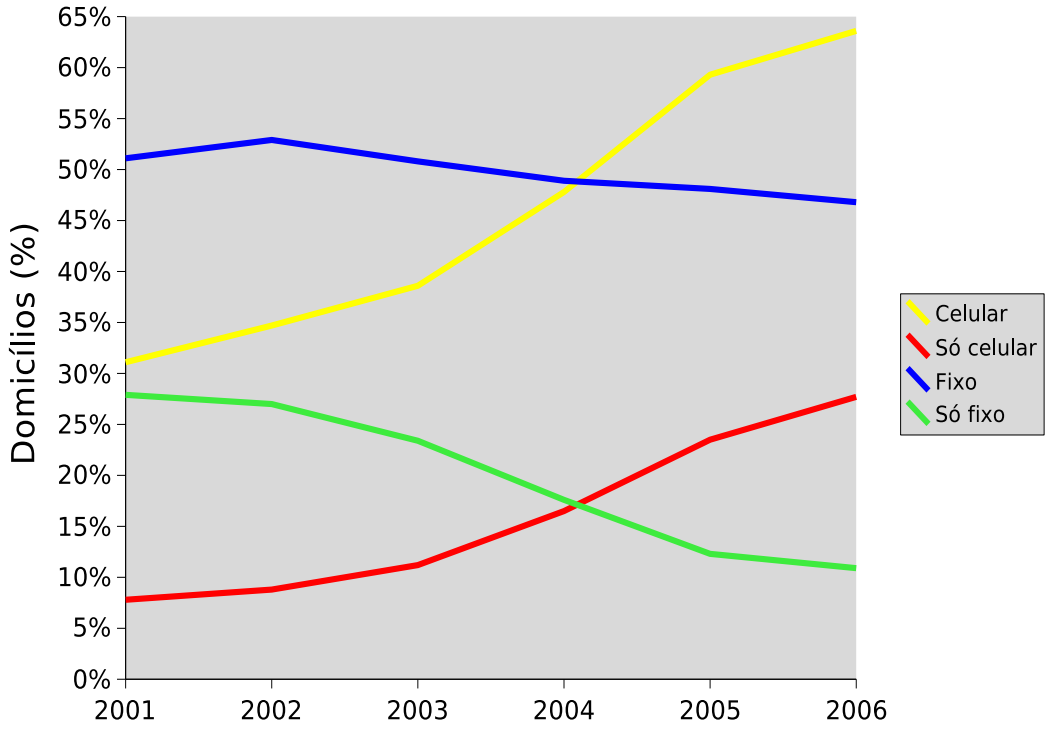

Figura 1.1: Presença do telefone fixo e do celular nas residências brasileiras

Diante deste cenário, os exemplos visionários do paradigma de computação pervasiva, concebido por Mark Weiser [59], estão cada vez mais próximos de se tornar realidade. Aplicações ou serviços podem interagir com dispositivos dispersos no ambiente, realizando operações de acordo com as percepções do contexto em que estão inseridos, mantendo determinada discrição de seu funcionamento [51]. Deste modo, o usuário pode se concentrar apenas na atividade que deseja realizar, não se preocupando com tarefas secundárias como configuração de rede, instalação de componentes, carregamento de dependências, adaptação ao contexto, etc. Tais tarefas podem ser desempenhadas automaticamente, a partir do momento em que se determina uma necessidade identificada através de informações contextuais.

De acordo com a viabilidade técnica e econômica, muitas empresas têm dedicado in-

[3]Fonte: Instituto Brasileiro de Geografia e Estatística - PNAD 2006. Os dados coletados até 2003 não incluem a população da área rural de Rondônia, Acre, Amazonas, Roraima, Pará e Amapá.

vestimentos para a criação de ambientes de computação pervasiva onde usuários possam realizar tarefas de forma fácil e transparente, minimizando, cada vez mais, a necessidade de intervenção humana [50]. Entretanto, a concepção de um ambiente pervasivo real apresenta alguns problemas que são comuns a grande parte dos sistemas computacionais. Apesar do aparato tecnológico atual dispor de diversas soluções para satisfazer requisitos de comunicação sem fio, memória, processamento e consumo de bateria, necessários aos ambientes pervasivos, os diferentes tipos de dispositivos, protocolos, ferramentas, linguagens, sistemas, etc., denotam uma grande complexidade na integração das entidades envolvidas e na manutenção do sistema criado [25].

Embasado nesta visão, Paul Horn, então vice-presidente do grupo de pesquisas da IBM[4], concebeu um novo paradigma conhecido como computação autônoma [25]. Neste novo paradigma, as aplicações possuem a capacidade de realizar operações como configuração, otimização, proteção e tratamento automaticamente, baseando-se em instruções definidas em uma linguagem de alto nível e facilitando a integração e manutenção das mesmas. Assim, uma aplicação capaz de adaptar-se automaticamente ao contexto na qual está inserida apresenta a capacidade de ser auto-configurável [28].

A união das características destes dois paradigmas permite desenvolver aplicações pervasivas auto-configuráveis, proporcionando uma noção de invisibilidade no funcionamento do sistema, automatizando a adaptação das aplicações ao contexto. Um notável exemplo no uso desta solução pode ser caracterizado quando um usuário se dirige para uma reunião e no meio do evento o seu aparelho celular recebe uma ligação emitindo um som desapropriado para o momento. Em vez do usuário precisar configurar o seu aparelho toda vez que vai para uma reunião, o sistema pervasivo auto-configurável poderia configurar o aparelho para não emitir sons durante a reunião, evitando o constrangimento do usuário. É nesse contexto de auto-configuração que se insere este trabalho.

1.1 Problemática

As características inerentes à computação pervasiva trazem algumas dificuldades com relação ao desenvolvimento e manutenção de aplicações. Neste sentido, na construção de

[4]http://www.ibm.com

aplicações pervasivas auto-configuráveis, deve-se abordar características como as descritas a seguir:

- **Ciência de contexto**: o conceito de ciência de contexto permite que as aplicações possam adquirir e interpretar as informações do ambiente para tomar decisões baseadas nas mesmas [7]. Alguns problemas podem ser observados na utilização deste conceito em sistemas pervasivos, tais como a padronização de um modelo para caracterização e o reconhecimento de situações onde a configuração automática é necessária [14].

- **Ambientes heterogêneos**: devido à crescente necessidade de integração e convergência das soluções computacionais, os ambientes pervasivos possuem a característica de contemplar entidades de hardware e software variadas, o que causa um aumento na complexidade gerencial do sistema. Afinal, dispositivos como *notebooks*, celulares, *PDAs*, etc., possuem características individuais de configuração e utilização dos recursos oferecidos no ambiente.

- **Ambientes dinâmicos**: a natureza dinâmica e imprevisível dos ambientes pervasivos, onde dispositivos móveis podem entrar ou sair dos ambientes a qualquer hora, introduz novos níveis de complexidade na gerência do sistema. Muito suscetível a falhas de comunicação, um sistema com características autônomas deve definir dentre várias ações, qual é a mais indicada para tratar diferentes situações que venham a ocorrer.

- **Dispositivos limitados**: mesmo com evolução considerável nos últimos anos, os dispositivos móveis atuais ainda possuem determinada limitação quando comparados a computadores de uso comum. Sendo assim, aplicações embarcadas nestes dispositivos devem considerar suas limitações de processamento, memória e armazenamento. As rotinas que tratam estas limitações devem ser desempenhadas automaticamente.

Apesar dos notáveis avanços e pesquisas nas áreas de computação pervasiva e computação autônoma, as características acima citadas apresentam novos desafios no desenvolvimento de aplicações que combinam características dos dois paradigmas.

1.2 Objetivo

Neste trabalho, tem-se como objetivo disponibilizar uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Esta arquitetura contempla conceitos da computação autônoma, que possibilita a adaptação automática de aplicações ou componentes de aplicações pervasivas ao contexto, a partir de informações contextuais disponíveis no ambiente. Estas informações podem ser coletadas por instruções pré-definidas, sensores, através de outras aplicações ou ainda por dedução lógica.

Assim, esta infra-estrutura deve apresentar uma implementação suficientemente flexível, capaz de simplificar sua adaptação num ambiente heterogêneo. Nesse sentido, sua arquitetura deve oferecer mecanismos de extensão capazes de facilitar a utilização da infra-estrutura no desenvolvimento de aplicações pervasivas em diversos domínios.

Para a validação do trabalho, esta infra-estrutura foi implementada com a linguagem Java, sobre uma arquitetura baseada em serviços. Esta arquitetura utiliza a tecnologia *Universal Plug and Play* (UPnP) [55], que provê mecanismos de descoberta e comunicação entre os dispositivos do ambiente.

1.3 Relevância

Ao mesmo tempo em que se investiga a utilização da computação autônoma na área de computação pervasiva, assunto pouco detalhado na literatura, neste trabalho propõe-se considerar aspectos que ainda não foram diretamente abordados no contexto de pesquisa definido. Trabalhos como [37] e [45] apresentam arcabouços para o desenvolvimento de aplicações auto-configuráveis. Entretanto, estes trabalhos não são dedicados a aplicações pervasivas que possuem as limitações citadas anteriormente. Já o trabalho proposto em [8] provê suporte à auto-configuração em aplicações pervasivas, mas este suporte é restrito a serviços de rede. A utilização da infra-estrutura proposta neste trabalho tende a minimizar os esforços de gerenciamento de aplicações pervasivas a partir de soluções autônomas, facilitando a criação dos ambientes pervasivos.

1.4 Estrutura da Dissertação

O restante deste trabalho está organizado da seguinte forma:

- No **Capítulo 2** é feita uma fundamentação teórica sobre os conceitos e tecnologias utilizados neste trabalho. Mais precisamente, são abordados os conceitos de Computação Pervasiva e Computação Autônoma, apresentando características, arquiteturas e detalhes destes paradigmas. Após é apresentada uma visão geral da tecnologia UPnP, utilizada na implementação deste trabalho e por fim é realizado um levantamento sobre os trabalhos relacionados.

- No **Capítulo 3** é descrita a infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Sua arquitetura, modelagem e funcionamento também são apresentados neste capítulo.

- No **Capítulo 4** é apresentado o estudo de caso que implementa a infra-estrutura proposta no Capítulo 3.

- No **Capítulo 5** são apresentadas as conclusões do trabalho, descrevendo suas contribuições e trabalhos futuros.

Capítulo 2

Fundamentação

Com a intenção de prover uma melhor compreensão dos conceitos envolvidos neste trabalho, apresenta-se, neste capítulo, uma breve descrição sobre computação pervasiva e computação autônoma, apresentando características, funcionamento e tecnologias envolvidas na concepção dos sistemas que seguem estes paradigmas. Por fim, é apresentada uma combinação dos dois paradigmas, criando o conceito de Computação Pervasiva Autônoma.

2.1 Computação Pervasiva

Desde a criação dos primeiros computadores pessoais na década de 70, a interação entre homem e máquina é realizada de forma reativa. Sempre que um usuário deseja executar alguma atividade com o computador ele realiza o processo de preparação do ambiente de trabalho, indicando explicitamente quais aplicações serão usadas e o que as mesmas devem fazer. Por exemplo, supondo que um escritor chamado João gosta de ouvir música clássica enquanto escreve seus livros em seu computador, ele precisaria realizar as seguintes tarefas para preparar o seu ambiente de trabalho:

1. Executar a aplicação apropriada para a reprodução de músicas;
2. Informar à aplicação que deseja ouvir uma música (ou coleção);
3. Indicar o arquivo de música desejado;
4. Informar à aplicação para começar a reproduzir o arquivo de música indicado;

5. Executar a aplicação para a edição de textos;

6. Abrir o arquivo para a escrita do texto.

Neste exemplo João precisou realizar seis tarefas para preparar o seu ambiente de trabalho. Dado que diariamente ele realiza as mesmas tarefas no mesmo horário, o computador de João poderia preparar o ambiente automaticamente, deixando que João se concentre apenas em sua atividade principal, que é escrever textos. Esta interação pró-ativa poderia ser realizada de acordo com as preferências e as experiências de João com o computador. Estendendo essa idéia para o contexto dos dispositivos eletrônicos dispostos na sala de trabalho de João, outras tarefas poderiam ser realizadas automaticamente na preparação do ambiente. Algumas destas tarefas poderiam ser:

- O condicionador de ar poderia climatizar a sala em uma temperatura agradável, de acordo com as preferências de João;
- As luzes da sala poderiam ser ligadas;
- O telefone celular de João poderia ser configurado para não receber chamadas enquanto ele trabalha;
- A cafeteira poderia ser ligada.

Durante as últimas décadas, a maioria dos sistemas de software e dos dispositivos eletrônicos tem apresentado comportamento reativo como no exemplo supracitado. Eles não "sabem" onde estão, o que há no ambiente, nem tampouco as preferências de quem os utilizam. Entretanto, nos últimos anos, tem sido possível perceber o início de uma mudança no conceito do comportamento destes dispositivos. Eles começaram a se tornar um pouco mais "inteligentes", agindo de acordo com as características do contexto em que estão inseridos.

Esta mudança de comportamento revela uma migração da computação tradicional, baseada em computadores pessoais, para uma nova era, onde o poder computacional está embarcado em diversos dispositivos eletrônicos espalhados nos ambientes do dia-a-dia, onde estes se integram de forma transparente na vida das pessoas. Esta nova era condiz com as idéias expostas por Mark Weiser em 1991, quando ele concebeu o paradigma da Computação Ubíqua ou Pervasiva [59].

Em sua concepção, Weiser afirmou que a computação pervasiva poderia ser alcançada a partir de três elementos principais: dispositivos baratos e com baixo consumo de energia, infra-estrutura de rede sem fio e sistemas que implementassem aplicações pervasivas. Na época da afirmação, a tecnologia de hardware não estava preparada para suportar a computação pervasiva. A tecnologia de redes sem fio ou não estava disponível, ou não estava embarcada nos dispositivos móveis. Como conseqüência, aplicações pervasivas não podiam ser desenvolvidas [35]. Entretanto, o cenário tecnológico atual permite dizer que as idéias visionárias de Weiser estão se tornando realidade, afinal:

- No campo de redes sem fio, tecnologias como Bluetooth [6] e Wi-Fi [41] vêm sendo amplamente utilizadas em diversos aparelhos eletrônicos do dia-a-dia, como celulares, aparelhos de som, condicionadores de ar, etc. Além disso, têm sido definidos protocolos de comunicação como *UPnP* [55], *ZeroConf* [22] e *Mobile IP* [46] que propiciam a criação dos ambientes de computação pervasiva;

- A evolução dos dispositivos móveis no que se refere à capacidade de memória, processamento e consumo de bateria tem permitido a execução de aplicações mais complexas nos mesmos e o suporte às tecnologias de rede sem fio supracitadas. Além disso, o barateamento destes dispositivos tem ajudado a difundi-los em todo o mundo.

- Como conseqüência dos avanços observados nas tecnologias de rede sem fio e dispositivos móveis, a comunidade científica pôde conceber as primeiras aplicações pervasivas e construir os primeiros ambientes "inteligentes" (ou pervasivos). Estes têm sido implementados através de projetos como Aura [18], Oxygen [23], Portolano [15] e Wings [34].

Um ambiente de computação pervasiva é caracterizado pela saturação de dispositivos computacionais com capacidade de comunicação, de modo que tais dispositivos suportem a mobilidade e se integrem às atividades do usuário de forma transparente [51]. Esta comunicação permite que inúmeros recursos sejam compartilhados entre os nós na rede pervasiva, fazendo do ambiente pervasivo um grande repositório de recursos. Não obstante, o alto grau de mobilidade destes nós, característica da computação pervasiva, faz com que os dispositivos entrem e saiam da rede constantemente e de forma não previsível. A dinamici-

dade da topologia de rede faz com que os recursos oferecidos se tornem disponíveis ou não de acordo com a presença dos dispositivos na rede.

Esta ausência de garantia da disponibilidade de recursos fez com que a comunidade científica investigasse abordagens para minimizar este problema, definindo e desenvolvendo soluções que envolvem o paradigma de Computação Orientada a Serviços (*Service-Oriented Computing*) [42; 27]. Afinal, neste paradigma são definidos elementos que permitem a descoberta de serviços sob demanda, o que se torna uma potencial solução para a disponibilização de recursos nos ambientes de computação pervasiva. Entretanto, a heterogeneidade e dinamicidade encontrada nos ambientes pervasivos revelam algumas dificuldades na implantação da descoberta de serviços existentes. A seguir descreve-se como a Computação Orientada a Serviços pode lidar com a disponibilização de recursos provendo serviços em ambientes de computação pervasiva. Por fim, é realizada uma breve descrição sobre a tecnologia UPnP.

2.1.1 Provisão de Serviços em Ambientes de Computação Pervasiva

A Computação Orientada a Serviços define um paradigma onde um serviço é elemento fundamental para a criação de aplicações [42]. Neste paradigma, um serviço é concebido como uma parte de software que implementa uma função e pode ser integrado com outros serviços na criação de aplicações mais complexas. Aplicações desenvolvidas com base neste paradigma devem oferecer mecanismos que satisfaçam as seguintes operações [43]:

1. **Publicação**: a publicação é feita para que os provedores de serviço indiquem quais serviços eles disponibilizam. Assim, tais serviços podem ser utilizados por outras entidades de aplicações orientadas a serviços;

2. **Descoberta**: uma vez publicados, os serviços podem ser descobertos pelos consumidores. Esta descoberta pode ser feita, por exemplo, com palavras-chave que indicam a funcionalidade do serviço;

3. **Seleção**: a seleção do serviço é utilizada nos casos em que mais de um serviço é indicado na fase de descoberta. Neste caso, o consumidor do serviço precisa definir qual serviço se adequa às suas necessidades;

4. **Ligação**: após descoberto e selecionado o serviço requerido, o consumidor inicia uma comunicação direta com o serviço, fazendo com que o mesmo possa ser executado.

Estas operações normalmente são desempenhadas com uma arquitetura que apresenta três entidades principais: provedor, cliente e registro [16]. Nesta arquitetura, conhecida como Arquitetura Orientada a Serviços e ilustrada na Figura 2.1, cada elemento possui responsabilidades diferentes. O provedor disponibiliza seus serviços publicando-os na entidade de registro. O cliente é a entidade que utiliza os serviços disponibilizados pelos provedores fazendo a descoberta através da comunicação com o registro. Após a descoberta, o cliente pode fazer a seleção e a ligação com o serviço disponibilizado pelo provedor.

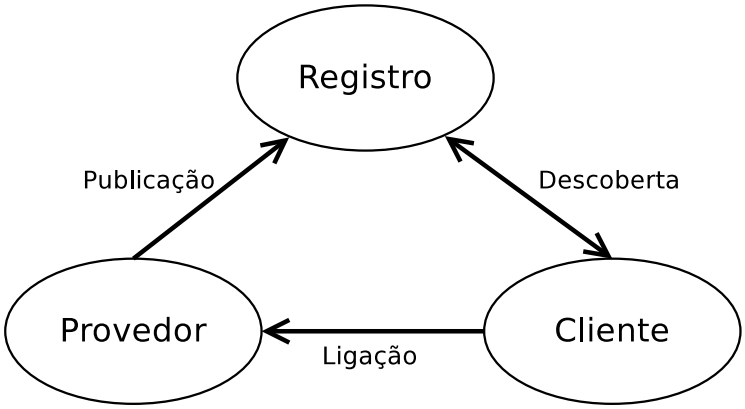

Figura 2.1: Arquitetura Orientada a Serviços

De acordo com a arquitetura supracitada, o processo de publicação, descoberta, seleção e ligação das arquiteturas orientadas a serviços revela uma potencial solução para o compartilhamento de recursos nos ambientes de computação pervasiva. Uma vez que os serviços são descobertos sob demanda, é possível lidar com a heterogeneidade e dinamicidade encontrada nestes ambientes, utilizando a descoberta de serviços sobre os dispositivos que estão dispersos no ambiente.

Nos últimos anos, a comunidade científica e a indústria têm apresentado diversos protocolos que implementam a descoberta de serviços. Enquanto na academia foram definidos protocolos como *Intentional Naming System* (INS) [1] e *Ninja Service Discovery Service* [24], a indústria apresentou tecnologias como Jini [57], criado pela *Sun Microsystems* e a tecnologia *Universal Plug and Play* (UPNP) [55], que possui contribuições de empresas como Microsoft, Intel e Nokia.

Apesar dos protocolos para a descoberta de serviços existentes apresentarem notáveis soluções que facilitam a interação entre as entidades de software, aumentando a usabilidade, os mesmos ainda não contemplam funcionalidades que satisfaçam as exigências dos ambientes de computação pervasiva. A descoberta de serviços sob demanda facilita a disponibilização de serviços nas situações em que os dispositivos podem conectar ou desconectar da rede a qualquer momento. Entretanto, a ligação para estes serviços não é realizada automaticamente, fazendo com que sua utilização seja definida *a priori* ou indicada pelo usuário quando preciso. Os serviços providos pelos dispositivos agem de forma reativa às requisições dos usuários através dos clientes.

Neste cenário, seria interessante que o ambiente pervasivo provesse uma infra-estrutura para definição automática das ligações e uso dos serviços disponíveis, sempre que as situações no ambiente demandassem a execução dos mesmos. Tomando como base o exemplo do escritor João, apresentado no início deste capítulo, uma vez que existissem serviços para ligar a cafeteira, luzes e condicionador de ar, tal infra-estrutura poderia conectar os dispositivos ligando seus serviços, de modo que o ambiente se adequasse ao horário de trabalho de João. Esta infra-estrutura deveria então realizar uma melhor integração entre os dispositivos e as pessoas que estejam no ambiente de computação pervasiva inferindo situações onde a ligação e utilização automática dos serviços ajam de forma transparente e eficaz, facilitando a vida das pessoas [61]. Na Seção 2.1.2, detalha-se a tecnologia UPnP apresentando sua estrutura e analisando como ela promove a descoberta de serviços. Esta tecnologia é utilizada na implementação da infra-estrutura proposta neste trabalho que é apresentada no Capítulo 3.

2.1.2 *Universal Plug and Play* (UPnP)

A tecnologia *Universal Plug and Play* (UPnP) [55] define uma arquitetura que permite a conectividade em redes pervasivas ponto-a-ponto sem a necessidade de configurações. Utilizando protocolos conhecidos, como TCP, IP, UDP e HTTP, os dispositivos que suportam UPnP podem, dinamicamente, entrar na topologia da rede, obter um endereço IP, disponibilizar suas capacidades (serviços) e ainda verificar a presença e capacidade dos dispositivos que estão na mesma rede. Utilizando mensagens baseadas em XML, estas operações são realizadas automaticamente de forma transparente para o usuário, deixando a infra-estrutura

de rede "invisível".

Na arquitetura UPnP são definidas duas entidades principais: dispositivo (*device*) e ponto de controle (*control point*), onde os dispositivos implementam serviços e apenas respondem as requisições dos pontos de controle, agindo como controladores. Por serem definidos por padrões abertos e amplamente utilizados, tanto dispositivos quanto pontos de controle podem ser implementados com diferentes linguagem e implantados em diversas plataformas, incluindo computadores pessoais e sistemas embarcados. Além disso, múltiplos dispositivos e pontos de controle podem ser implementados em um mesmo dispositivo físico.

Ao entrar na rede UPnP, um dispositivo obtém um endereço IP automaticamente através do protocolo DHCP[1]. A partir deste ponto, a rede UPnP pode caracterizar cinco fases distintas, apresentadas a seguir:

1. **Descoberta**: a descoberta é feita quando o dispositivo entra na rede. Quando é adicionado, o dispositivo anuncia seus serviços a todos os pontos de controle da rede, utilizando o protocolo para a descoberta de serviços SSDP [19]. O SSDP também permite que a descoberta de serviços seja realizada pelo ponto de controle através de buscas aos dispositivos que estejam conectados momentaneamente;

2. **Descrição**: após a descoberta, os pontos de controle ainda não possuem informações detalhadas dos dispositivos presentes na rede. Para isto, eles podem requisitar a descrição dos dispositivos através de uma URL[2] específica conhecida na fase de descoberta. Com esta descrição, os pontos de controle obtêm informações específicas do dispositivo como fabricante, fornecedor, nome e modelo. A descrição apresenta ainda uma lista com todos os serviços (detalhando a interface dos serviços) e dispositivos incorporados ao dispositivo consultado, além de prover URLs para Controle, Eventos e Apresentação, detalhadas a seguir;

3. **Controle**: na fase de descrição, os pontos de controle obtêm uma lista dos serviços disponibilizados na rede através dos dispositivos. Cada serviço indica suas ações (funcionalidades) apresentando sua interface com seus parâmetros de entrada e saída. Com o conhecimento desta interface, os pontos de controle podem executar um determi-

[1] *Dynamic Host Configuration Protocol*
[2] *Uniform Resource Locator*

nado serviço fazendo uma requisição à URL de Controle obtida também na fase de descrição. Esta requisição é expressa em XML, usando o protocolo SOAP[3].

4. **Eventos**: a descrição obtida no passo 3, inclui, além da lista de ações, uma lista de variáveis que modelam o estado do serviço em sua execução. Neste caso, os pontos de controle podem se inscrever no serviço requerendo informações sobre estas variáveis. O serviço publica atualizações quando essas variáveis mudam e um ponto de controle pode inscrever-se para receber essas informações.

5. **Apresentação**: a URL de apresentação obtida, também, na fase de descrição, permite que usuário possa carregá-la em um navegador para obter mais informações sobre o dispositivo. Assim, o usuário pode ter informações de estado do dispositivo e, possivelmente, executar os serviços oferecidos pelo mesmo.

Baseando-se em padrões amplamente utilizados, a tecnologia UPnP tira proveito da experiência e do conhecimento difundido na comunidade, fazendo com que o desenvolvimento de dispositivos e pontos de controle se torne algo comum para os desenvolvedores. Para facilitar a interoperabilidade, algumas especificações foram definidas para a concepção de dispositivos UPnP com finalidade comum. Estas especificações definem como os dispositivos UPnP devem ser implementados, padronizando, entre outras coisas, um identificador comum, nomes de serviços, parâmetros de entrada e saída. Atualmente existem especificações que definem dispositivos de multimídia, impressoras, iluminação, pontos de acesso, etc. Estas especificações já vêm sendo implementadas e embarcadas em dispositivos físicos comuns como aparelhos de som, lâmpadas e televisores. Algumas destas especificações podem ser encontradas em [56].

2.2 Computação Autônoma

A necessidade de integração de sistemas computacionais heterogêneos, observada no mercado atual, tem dificultado as operações de gerência e manutenção nos sistemas corporativos. As milhares de linhas de código desses sistemas têm apresentado níveis de complexidade que desafiam a capacidade da compreensão humana [28].

[3] Simple Object Access Protocol

Em 2001, Paul Horn, então vice-presidente da área de pesquisas da IBM, definiu o termo "Computação Autônoma" para descrever uma solução para a crescente complexidade envolvida nos sistemas de software [25]. Inspirado no sistema nervoso autônomo dos humanos, a computação autônoma apresenta uma visão onde os sistemas de software possuem a capacidade de auto-gerência baseados em informações contextuais e guiados por políticas descritas em alto nível por seus administradores [38]. Assim, as complexidades de baixo-nível seriam abstraídas dos humanos, deixando com que os administradores de software deixem de lado as rotineiras operações de manutenção, concentrando-se em políticas descritas em alto nível.

Nos últimos anos a computação autônoma tem chamado a atenção da comunidade científica que, por sua vez, tem incentivado as pesquisas nessa área em busca de uma padronização de conceitos, modelos e aplicações que estejam inseridas na visão de Horn. Apesar de existirem algumas interpretações e adaptações da visão inicial, a definição mais aceita na comunidade classifica a computação autônoma em quatro aspectos principais detalhados a seguir [33; 28]:

1. **Auto-configuração**: o aspecto de auto-configuração define que os sistemas de software devem se adaptar automaticamente de acordo com as mudanças ocorridas no ambiente em que estão inseridos. Entre outras coisas, isto permite que o sistema continue funcionando normalmente quando uma nova entidade de software se integra ao sistema.

2. **Auto-tratamento**: em computação autônoma, a capacidade de auto-tratamento é responsável por detectar, diagnosticar e reparar problemas resultantes de *bugs* ou falhas de hardware e software. Aplicações com esta capacidade se baseiam em dados de *log* e monitores que indicam falhas no sistema. Uma vez diagnosticado o problema, o sistema deve aplicar as mudanças necessárias para reparar a falha ou indicar o tratamento apropriado para sanar o problema.

3. **Auto-otimização**: sistemas de banco de dados como Oracle e DB2 apresentam centenas de parâmetros de configuração que devem ser ajustados para otimizar o funcionamento. Para isso, os administradores devem ter o conhecimento das responsabilidades de cada parâmetro e verificar qual a melhor combinação para utilizar no sistema em que está administrando. Em computação autônoma, tais ajustes deveriam ser desem-

penhados de forma automática, tanto em tempo de instalação, quanto em tempo de execução, com a finalidade de melhorar o desempenho do sistema.

4. **Auto-proteção**: apesar da existência de *firewalls* e ferramentas de detecção de intrusão, em muitas situações os humanos devem estar presentes para decidir como proteger o sistema a partir de ataques maliciosos. Por exemplo, ao identificar um grande número de requisições com um determinado IP em um curto espaćo de tempo, servidores de nomes poderiam caracterizar uma situação de ataque e bloquear as requisições originadas desta máquina automaticamente. Neste sentido, aplicações com a capacidade de auto-proteção preservam o funcionamento do sistema se defendendo de ataques maliciosos e falhas em cascata, além de antecipar problemas com base em relatórios de *log*, requisições, histórico, etc.

A automatização das operações relacionadas a estes aspectos se baseia em informações identificadas no próprio sistema e no ambiente em que o mesmo está inserido através de mecanismos que têm ciência do contexto, como apresentado no Capítulo 1. Sistemas que desempenham estes aspectos são conhecidos como sistemas autônomos que possuem a característica de **auto-gerência**. O que não quer dizer que os aspectos de configuração, tratamento, otimização e proteção estejam inseridos em dimensões ortogonais de uma característica maior de auto-gerência. Em algumas situações estes aspectos se combinam e se sobrepõem na busca da auto-gerência [38].

2.2.1 Considerações Arquiteturais

Embora haja diferentes arquiteturas para sistemas autônomos, a idéia geral é que eles sejam projetados em uma arquitetura de componentes onde os mesmos devem ser gerenciados por um gerente de autonomia. Neste sentido, um ou mais componentes podem ser gerenciados por um único gerente de autonomia, formando assim um elemento autônomo [53]. Laços de controle com sensores e atuadores, aliados ao conhecimento sobre o sistema com políticas de planejamento, permitem que um elemento autônomo tenha ciência de si, de seu ambiente e também se auto-gerencie [11]. Com esta arquitetura, o gerente de autonomia pode gerenciar seus componentes seguindo um ciclo com etapas de **M**onitoração, **A**nálise, **P**lanejamento e **E**xecução. Este ciclo, conhecido como ciclo MAPE, é ilustrado na Figura 2.2.

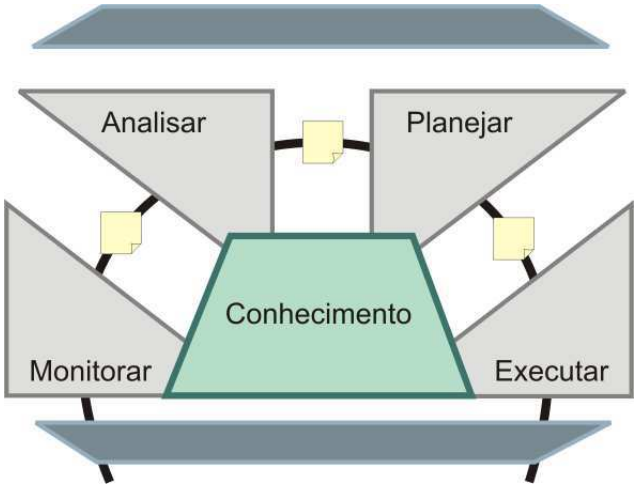

Figura 2.2: Laço de controle MAPE

As partes de monitoração e análise do ciclo processam informação de sensores para prover tanto auto-consciência quanto consciência do ambiente externo. As partes de planejamento e execução decidem o comportamento de auto-gerência necessário que será executado pelos atuadores. Os componentes do MAPE usam correlações, regras, expectativas, histórias e outras informações conhecidas pelo elemento autônomo ou disponibilizadas para ele através de um repositório de conhecimento acessado pelo gerente autônomo.

2.2.2 Elementos Autônomos

De maneira geral, sistemas autônomos podem ser vistos como coleções interativas de elementos autônomos. Cada elemento autônomo é um constituinte individual do sistema que contém recursos e entrega serviços a humanos ou outros elementos autônomos. Esses elementos gerenciam seu comportamento interno e suas relações com outros elementos autônomos verificando se estão de acordo com políticas que humanos ou outros elementos estabeleceram. Para dar suporte aos elementos autônomos e suas interações, sugere-se uma infra-estrutura distribuída e orientada a serviços [28].

Um elemento autônomo irá tipicamente consistir de um ou mais elementos gerenciados e um gerente de autonomia que os controla e representa, como ilustrado na Figura 2.3. O elemento gerenciado vai ser essencialmente equivalente ao que encontramos em sistemas não autônomos. Ele pode corresponder a um recurso de hardware, como armazenamento ou CPU, ou pode ser um recurso de software como um banco de dados, um serviço de

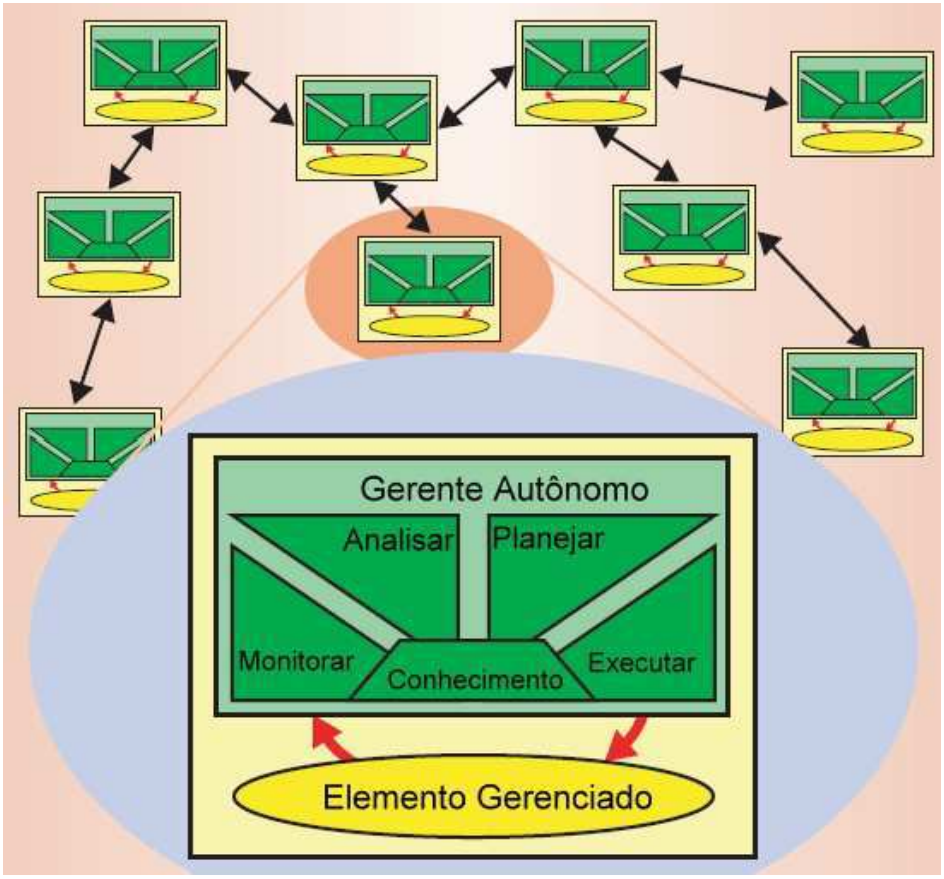

Figura 2.3: Elemento autônomo

diretórios ou um grande sistema legado. Se observado sob um nível mais alto, um elemento gerenciado pode ser visto como um serviço, ou um negócio individual. De forma geral, elementos autônomos serão dotados de ciclos de vida complexos, tratando normalmente com várias *threads* de atividades e continuamente monitorando e respondendo ao ambiente em que estão situados.

2.2.3 Políticas

A transformação de diretivas em ações a serem realizadas pelos elementos é de grande importância para o comportamento dos sistemas autônomos. Os comportamentos são determinados através de políticas descritas em alto nível por administradores e usuários do sistema autônomo. Uma política é uma representação, em uma forma padrão, do comportamento de-

sejado e, também, das restrições sobre o comportamento. Neste caso, os sistemas autônomos devem considerar duas questões importantes [29]: como os administradores do sistema deveriam expressar estas políticas e como o sistema autônomo usaria tais políticas para gerenciar o seu próprio comportamento?

Neste sentido, os sistemas autônomos utiliza abordagens amplamente discutidas nas disciplinas de Inteligência Artificial.Em [60] são apresentados três tipos de políticas que podem ser utilizadas na concepção de sistemas autônomos:

- **Políticas de ação**: é a forma de especificação mais básica, a qual é tipicamente expressa na forma *SE <condição> ENTÃO <ação>*. Um elemento autônomo que emprega políticas de ação deve medir ou sintetizar as quantidades presentes nas condições e deve executar as ações sempre que as condições são satisfeitas;

- **Políticas de objetivos**: descrevem condições que podem ser atingidas sem especificação de "como". Neste caso, em vez de informar o que o sistema deve fazer, as políticas de objetivos descrevem o estado final que o sistema deve atingir em determinadas situações. Neste sentido, é possível afirmar que as políticas de objetivos são mais robustas do que políticas de ação, pois um humano ou outro elemento autônomo pode dar direções para outro elemento sem que haja requisição do conhecimento detalhado das atividades internas do elemento.

- **Políticas de função de utilidade**: especificam o quão desejáveis são estados alternativos, um com relação ao outro. Essa relação entre os estados pode ser obtida através de atribuição numérica ou ordenação parcial ou total dos estados. Funções de utilidade são definidas como extensões das políticas de objetivo, pois determinam automaticamente o objetivo mais importante em uma dada situação.

2.3 Computação Pervasiva Autônoma

A Computação Pervasiva idealiza a criação de ambientes carregados de dispositivos computacionais que se integram à vida dos humanos de forma transparente, adaptando-se automaticamente de acordo com as situações observadas no contexto em que estão inseridos. Dez

anos depois, a Computação Autônoma (2.2) indicava que os sistemas computacionais deveriam desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo tarefas complexas por políticas descritas em alto nível por usuários e administradores. Apesar dos dois paradigmas apresentarem focos distintos, é possível perceber que eles possuem características em comum. Por exemplo, a capacidade de adaptação ao contexto de forma transparente para os usuários está presente nos sistemas de computação pervasiva e de computação autônoma. Com isso, alguns autores já definem o termo de Computação Pervasiva Autônoma nos casos em que as tecnologias desenvolvidas apresentam características dos dois paradigmas [47; 2; 40].

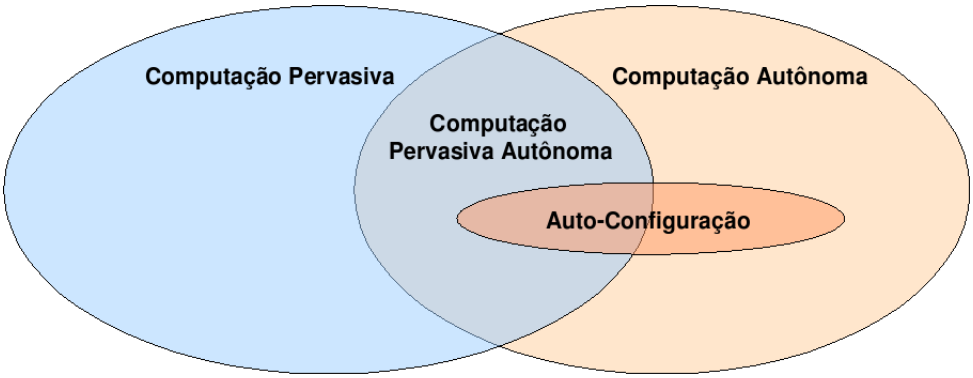

Figura 2.4: Relação entre Computação Pervasiva e Autônoma

A partir da definição da Computação Pervasiva Autônoma, é possível dizer que este trabalho está inserido na característica de auto-configuração desta nova computação que combina a Computação Pervasiva e Autônoma. Na Figura 2.4, apresenta-se uma ilustração da combinação destes paradigmas.

2.4 Trabalhos Relacionados

Recentes pesquisas nos campos de computação autônoma e computação pervasiva têm apresentado interessantes trabalhos nestas duas áreas. Alguns destes trabalhos permitem a inserção de aspectos de computação autônoma em ambientes de computação pervasiva reunindo características destes dois paradigmas. A seguir são apresentados alguns destes trabalhos, que serviram de base para a concepção dos objetivos da infra-estrutura definida nesta dissertação.

2.4.1 O Arcabouço *Autonomia*

No trabalho descrito em [12], os autores apresentam um arcabouço, conhecido por *Autonomia*, que permite a introdução de características autônomas em aplicações distribuídas. Como prova de conceito, um protótipo foi implementado com suporte às propriedades de auto-tratamento e auto-configuração.

Apesar de se mostrar como um arcabouço capaz de dar suporte às propriedades de computação autônoma, o referido trabalho não apresenta como são definidas e tratadas políticas de alto nível nem o mecanismo de decisão e não utiliza informações contextuais. Além disso, o arcabouço proposto não foi diretamente projetado para ser utilizado no desenvolvimento de aplicações pervasivas.

Já em [8] os autores utilizam o *Autonomia* em uma abordagem de suporte à auto-configuração em sistemas pervasivos. Entretanto, neste trabalho não são apresentadas características que suprem as deficiências do *Autonomia*, bem como, os resultados obtidos no experimento realizado.

2.4.2 Um Arcabouço de Suporte à Auto-otimização

O arcabouço descrito em [48] permite o desenvolvimento de aplicações autônomas para ambientes de computação pervasiva. Este arcabouço utiliza informações de contexto, definindo preferências do usuário para desempenhar características de auto-otimização (*self-optimizing*) nos ambientes pervasivos. Além de não apresentar suporte para o desenvolvimento de aplicações auto-configuráveis, este trabalho não revela características de seu funcionamento, bem como, sua arquitetura, detalhes de implementação e um estudo de caso que valide sua utilização.

Em [47], os mesmos autores do trabalho supracitado desenvolveram um arcabouço de suporte à auto-otimização apresentam uma abordagem que combina técnicas de Inteligência Artificial com a especificação de objetivos para realizar operações em ambientes de computação pervasiva. A partir das preferências do usuário, de informações de contexto e de políticas de segurança, um arcabouço de auto-otmização planeja a melhor forma para atingir um objetivo do usuário expresso em linguagens de alto nível. Mais uma vez, os esforços deste trabalho não estão direcionados ao desenvolvimento de aplicações pervasivas

auto-configuráveis. Contudo, tanto neste trabalho como em [29], o uso de funções de utilidade [54] tem apresentado bons resultados na implementação de mecanismos de decisão de sistemas autônomos.

2.4.3 Um Arcabouço para o Desenvolvimento de Componentes Auto-configuráveis

No trabalho apresentado em [45], define-se um arcabouço que permite a construção de componentes auto-configuráveis. Sua arquitetura modularizada permite que componentes de computação autônoma sejam obtidos e instalados automaticamente, em tempo de execução. Estas operações se baseiam em preferências do usuário, regras de negócio e meta-dados para os componentes.

Apesar de apresentar algumas características interessantes para a utilização de componentes auto-configuráveis, como a composição e modificação dinâmica de componentes, o trabalho não deixa explícito como são tratadas as informações de contexto e aquisição de informações. Além disto a a implementação do trabalho não foi feita para trabalhar com as limitações dos dispositivos móveis encontrados nos ambientes de computação pervasiva.

2.4.4 Sistemas de Auto-tratamento

O trabalho definido em [44] propõe um sistema de auto-tratamento utilizando abordagens multi-agentes. Baseado em dados verificados nos históricos das aplicações (*log*), juntamente com a monitoração dos níveis de utilização de memória e processador, o sistema multi-agentes procura possíveis erros ocorridos na aplicação. Após a verificação estes dados são coletados e repassados por uma série de agentes finalizando com um diagnóstico indicando as ações que devem ser tomadas para a reparação dos erros da aplicação. A utilização de políticas restritas, mapeando quatro principais tipos de erro, restringe o domínio de atuação do sistema, tornando-o inadequado para utilização em domínios de auto-configuração em ambientes de computação pervasiva.

Nesse sentido, o trabalho apresentado em [2] propõe um modelo arquitetural para a concepção de sistemas de auto-tratamento em ambientes de computação pervasiva. Neste modelo os dispositivos móveis realizam o auto-tratamento através de uma rede *ad-hoc*, sem

a necessidade de uma infra-estrutura de suporte. A ausência de um modelo de contexto faz com que os próprios dispositivos indiquem a necessidade de auto-tratamento, deixando a criação das políticas de alto nível individuais e isoladas.

2.4.5 O Uso de Ontologias para Modelar Contexto e Raciocínio em Computação Pervasiva

O trabalho definido em [13] apresenta uma abordagem baseada em ontologias para modelar contexto e raciocínio em aplicações de computação pervasiva. Em sua arquitetura, um modelo de entidades serve de base para representação e gerenciamento das informações do ambiente de computação pervasiva contemplando entidades comuns aos ambientes de computação pervasiva, como Pessoa, Localidade e Dispositivo. A partir deste modelo, a representação do ambiente é feita através de uma ontologia comum que indica relações e atributos das entidades do modelo. Esta representação permite que usuários possam cadastrar regras que, dada um determinado estado na representação do ambiente, uma ação deve ser executada. Um estudo de caso foi apresentado para que a configuração de toque de um aparelho celular fosse gerenciada através de regras definidas pelo usuário. Entretanto, os mecanismos de aquisição de informação do ambiente e a forma de atuação no mesmo não foram detalhados no trabalho.

O uso de ontologias na representação do conhecimento permite um grau de detalhamento de entidades e relações mais preciso por prover um vocabulário adicional através de uma semântica formal para a definição de classes, propriedades, relações e axiomas. Neste sentido, as ontologias podem ser obtidas através de informações descritas em triplas RDF[4] [30], apresentando sujeito, predicado e objeto para indicar informações do ambiente.

2.4.6 A Utilização de Arquiteturas UPnP na Construção de Ambientes Inteligentes

O trabalho apresentado em [52] descreve um sistema que implementa um ponto de controle UPnP que provê acesso integrado e intuitivo aos dispositivos UPnP dispersos no ambiente.

[4] *Resource Description Framework*

Apoiado por informações que indicam as dimensões do espaço físico do ambiente, este sistema, nomeado PECo, gera uma visualização 3D do ambiente, exibindo-a através de uma interface gráfica interativa em aparelhos pessoais como PDAs e celulares. Assim usuários podem executar ações dos dispositivos UPnP dispersos no ambiente indicando dispositivo e ação diretamente no seu PDA.

Apesar de não prover uma forma transparente para o usuário interagir com o ambiente, fazendo com que o mesmo tenha que indicar a ação a ser executada, este trabalho contribui com as pesquisas relacionadas à tecnologia UPnP, que a cada dia tem se tornado mais presente na concepção dos ambientes de computação pervasiva.

2.4.7 Considerações sobre os Trabalhos Relacionados

De acordo com os trabalhos acima descritos é possível observar que as implementações de sistemas autônomos, em geral, apresentam mecanismos de raciocínio específicos para soluções individuais. Estes mecanismos utilizam funções de utilidade, regras, e outros formalismos de Inteligência Artifical para automatizar operações de acordo com as informações contextuais e as políticas de computação autônoma. Em nenhum caso os sistemas autônomos apresentados utilizam abordagens com múltiplos mecanismos de raciocínio, facilitando a utilização de políticas diferentes para os ambientes heterogêneos das aplicações pervasivas.

De maneira geral, os modelos de contexto utilizados apresentam entidades comuns aos ambientes de computação pervasiva como Localidade, Dispositivo e Pessoa. Estas entidades serão mantidas na concepção de modelo base para a infra-estrutura definida nesta dissertação. A utilização de triplas RDF pelo trabalho definido em [13], também foi considerada na implementação da infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis.

Por fim, o trabalho descrito em [52], revela que a tecnologia UPnP têm sido utilizada em diversos trabalhos que implementam ambientes de computação pervasiva. Neste sentido, esta tecnologia também foi utilizada na implementação do estudo de caso apresentado no Capítulo 4.

Capítulo 3

Infra-estrutura para o Desenvolvimento de Aplicações Pervasivas Auto-configuráveis

Neste capítulo é detalhada a infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Inicialmente apresenta-se uma visão geral, seguida da arquitetura e o detalhamento do projeto de baixo nível. Por fim é ilustrado o funcionamento da infra-estrutura.

3.1 Visão Geral da Infra-estrutura

A infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis se baseia em conceitos conhecidos nos paradigmas de Computação Pervasiva e Computação Autônoma, apresentados no Capítulo 2. Nesse sentido, o Ciclo de autonomia MAPE é adaptado para funcionar sobre uma arquitetura orientada a serviços, amplamente utilizada na criação dos ambientes de computação pervasiva. Neste sentido, os passos de *monitoração*, *análise*, *planejamento* e *execução* são adaptados e atrelados a um modelo de contexto com entidades que definem algumas relações entre si.

Utilizando uma abordagem centralizada, a implementação do ciclo MAPE define que a fase de *monitoração* é iniciada a partir de dispositivos no ambiente que provêem informações sobre as interações e mudanças de estado para a infra-estrutura. Estas informações são traduzidas em uma representação formal do ambiente, que apoiará o raciocínio dos mecanismos

que utilizam políticas de alto nível para prover a auto-configuração. Esta formalização é padronizada a partir de um modelo que define as entidades e interações comuns realizadas nos ambientes onde a infra-estrutura definida neste trabalho será implantada.

Após a monitoração, as informações enviadas são repassadas para um módulo de tratamento de políticas, onde é iniciada a fase de *análise*. Devido às diversas abordagens utilizadas no tratamento de políticas, a infra-estrutura definida neste trabalho provê uma arquitetura extensível que permite a utilização de inúmeras abordagens para este fim. Dessa forma, o raciocínio utilizado para prover a auto-configuração poderá utilizar abordagens com políticas de ação, políticas de objetivo, funções de utilidade, entre outras. A verificação de tais políticas deve indicar que um determinado serviço disponível no ambiente deve ser executado, promovendo a auto-configuração.

Uma arquitetura que utiliza diversas abordagens para o tratamento de políticas de auto-configuração deve considerar a situação em que os mecanismos de tratamento de políticas apontem um mesmo serviço a ser executado no ambiente. Assim, a fase de *planejamento* é iniciada, verificando redundâncias, erros e otimizando os serviços indicados pelas políticas verificadas na fase anterior. Uma vez terminada a fase de planejamento, estes serviços são repassados para um componente atuador que inicia a fase de *execução*. Como em um ambiente de computação pervasiva comum os dispositivos entram e saem a todo instante e, conseqüentemente, os serviços que eles disponibilizam podem estar disponíveis ou não. O componente atuador deve verificar a disponibilidade dos serviços indicados na fase de planejamento para que estes possam ser definitivamente executados. Ao término do ciclo, os casos de sucesso e falha na execução dos serviços, em busca da auto-configuração, são armazenados num histórico para que um administrador humano possa otimizar o processo de auto-configuração, baseado nestas informações.

A execução do ciclo MAPE de acordo com a infra-estrutura proposta neste trabalho ataca diretamente os problemas indicados na Seção 1.1. Afinal, a interoperabilidade conseguida com a arquitetura orientada a serviços difundida entre os dispositivos móveis suprime a dificuldade de se trabalhar com ambientes heterogêneos e dinâmicos; a utilização de um modelo de entidades e relações que trabalha em conjunto com um mecanismo flexível para o tratamento de políticas de alto nível promove a ciência de contexto na infra-estrutura e; a utilização da infra-estrutura de forma centralizada, acessível por todos os dispositi-

vos os dispositivos inseridos na arquitetura orientada a serviços, faz com que os dispositivos possam participar do sistema de auto-configuração sem requerer a instalação de uma aplicação específica no mesmo. A arquitetura da infra-estrutura para aplicações pervasivas auto-configuráveis é descrita a seguir.

3.2 Arquitetura

A arquitetura da infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis define um gerente de autonomia que interage com sensores, aplicações e serviços dispostos em um ambiente de computação pervasiva. Esta arquitetura é ilustrada na Figura 3.1 e seus módulos são descritos a seguir.

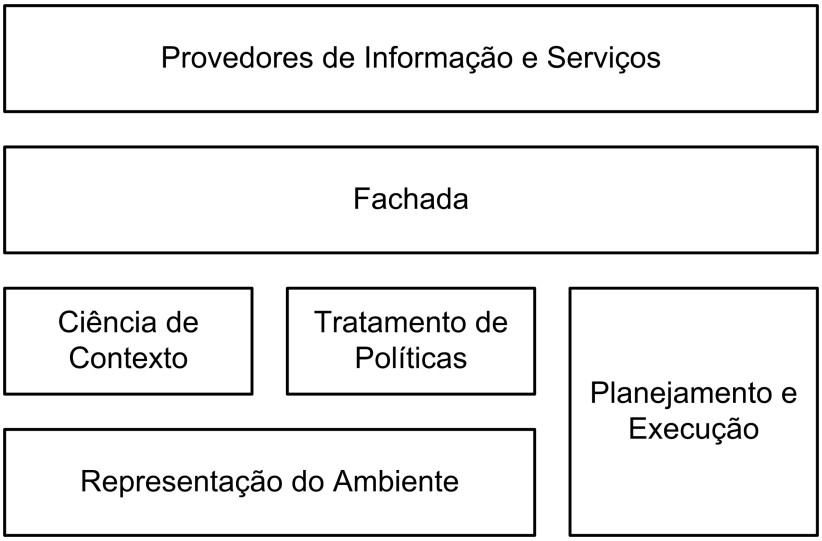

Figura 3.1: Arquitetura de alto nível da Infra-estrutura

3.2.1 Representação do Ambiente

Como apresentado no Capítulo 2, as características da computação pervasiva idealizam a criação de ambientes que se integram às vidas das pessoas de forma transparente. Assim, os dispositivos eletrônicos dispostos em bibliotecas, escritórios e *shoppings*, poderiam interagir com as pessoas se adaptando de acordo com as relações comuns nestes ambientes e com as preferências das pessoas. Entretanto, é possível perceber que tais ambientes possuem relações e entidades distintas, fazendo com que as interações observadas em um escritório

sejam diferentes das de um *shopping*. Por exemplo, o ambiente barulhento de um *shopping* permite que os usuários configurem seus aparelhos celulares para emitir um som alto ao receber chamadas telefônicas, o que não é observado em escritórios em que há eventuais reuniões.

As características diferentes dos ambientes físicos motivam a criação de um modelo de entidades e relações específico para o ambiente em que a infra-estrutura para o desenvolvimento de aplicações pervasivas será executada. Este modelo define as principais entidades e relações que estão presentes no ambiente, facilitando a representação formal do mesmo. Neste sentido, em um ambiente como um *shopping*, as entidades Visitante, Loja, Promoção e Lanchonete poderiam ser definidas junto a um conjunto de relações como: um Visitante compra produtos em uma Loja; uma Loja tem Promoções que interessam algum Visitante; uma determinada Lanchonete está fechada, etc. Suprimindo os sujeitos e os objetos das orações supracitadas, pode-se definir relações através de predicados como: "compra produto em", "tem promoção", "está".

Na infra-estrutura proposta neste trabalho, o Módulo de Representação do Ambiente deve ser definido de acordo com o ambiente em que a mesma será implantada. Tal módulo contém um conjunto mínimo de **entidades** e **predicados** de acordo com as características de domínio presentes no ambiente real e permite que relações conhecidas sejam armazenadas em um **banco de dados** utilizando um modelo comum. Por exemplo, a entidade "Livro" e a relação que representa o empréstimo do mesmo poderiam definir uma entidade e uma relação do modelo utilizado em uma biblioteca pervasiva. Ao mesmo tempo, esta biblioteca poderia cadastrar todos os livros disponíveis para que eles fossem identificados através de um código único.

A representação do estado atual do ambiente, realizada pelo Módulo de Representação do Ambiente permite que os mecanismos (engenhos) de tratamento de políticas (detalhados a seguir) possam inferir sobre as situações onde a auto-configuração se faça necessária. Além disso, a representação formal do ambiente de computação pervasiva permite que aplicações administrativas divulguem o estado atual do ambiente, facilitando as tarefas de um administrador humano. Portanto, é necessário que na instanciação deste modelo seja possível identificar situações em que a auto-configuração é desejável. Além disso, o cadastramento das informações de pessoas e dispositivos que estão comumente presentes no ambiente irá

facilitar a representação e a inferência sobre o mesmo. Este módulo serve de base para os módulos descritos a seguir.

3.2.2 Ciência de Contexto

O Módulo de Ciência de contexto é responsável pela aquisição, identificação e tradução das informações do contexto que serão preservadas em um banco de dados de representação do ambiente. Estas informações representam ações que ocorreram no ambiente e são enviadas em forma de mensagem por Provedores de Informação que estão dispersos no mesmo. Tais mensagens devem ser mapeadas de acordo com o modelo de entidades e predicados provido pelo Módulo de Representação do Ambiente. Os provedores são implementados na forma de sensores, aplicações e serviços que se comunicam com um **Gerente de Monitoração**, através de um serviço de interface entre a infra-estrutura e os dispositivos do ambiente. Este serviço define que a entrada das informações de contexto deve ser baseada em XML com o formato de uma tripla RDF[1] [30] representando uma situação como $< Sujeito, Predicado, Objeto >$, onde:

- **Sujeito**: define o sujeito que realizou a ação representada pela mensagem. Normalmente, este sujeito indica alguma entidade presente no modelo de entidades definido no Módulo de Representação do Ambiente.
- **Predicado**: define a ação representada pela mensagem. Normalmente, este predicado condiz com algum predicado presente no conjunto de predicados definido no Módulo de Representação do Ambiente.
- **Objeto**: devem ser indicados valores que complementem a mensagem para um melhor entendimento da ação representada pela mensagem.

A junção das informações recebidas pelos provedores de informação permite que a infra-estrutura para o desenvolvimento de aplicações pervasiva possa representar formalmente o estado atual do ambiente no banco de dados provido pelo Módulo de Representação do Ambiente. Entretanto, é necessário fazer um mapeamento das informações recebidas no formato RDF para a representação do atual estado do ambiente neste banco de dados. Este

[1] *Resource Description Framework*

mapeamento das informações recebidas pelo Gerente de Monitoração para um mecanismo de persistência é realizado por um componente definido como **Gerente de Tradução**. Este gerente agregará componentes de tradução que atualizarão o banco de dados a partir das informações RDF.

Considerando que a mensagem $< Maria, Entrou, Sala104 >$ foi recebida, o Gerente de Tradução poderia mapear tal ação em um banco de dados relacional onde uma instância da entidade Pessoa (no caso "Maria") teria relação com uma instância da entidade do tipo Sala (no caso "Sala 104"). Uma vez que tais informações foram traduzidas para um banco de dados relacional, será possível, por exemplo, consultar este banco utilizando uma linguagem como SQL[2]. A mensagem RDF recebida pelo Gerente de Monitoração, além de utilizada para a atualização do banco de dados, é repassada integralmente para o Módulo de Tratamento de Políticas. Assim os mecanismos de tratamento de políticas podem, também, representar o estado do ambiente com uma estrutura adeqüada ao seu funcionamento. O detalhamento deste processo é apresentado a seguir.

3.2.3 Módulo de Tratamento de Políticas

O Módulo de Tratamento de Políticas define o ponto principal da infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Neste módulo, as políticas são cadastradas e utilizadas para prover a auto-configuração de acordo com as modificações ocorridas no ambiente informadas pelos Provedores de Informação. Para realizar estas funções o Módulo de Tratamento de Políticas encapsula um conjunto de **Engenhos de Políticas** e um **Gerente de Políticas**. Estes componentes são descritos a seguir.

Engenhos de Políticas

Como apresentado no Capítulo 2, um sistema autônomo se comporta de acordo com um conjunto de políticas descritas em alto nível pelos administradores do sistema. Desta forma, a complexidade envolvida no gerenciamento do sistema deve ser abstraída ao máximo, facilitando o trabalho dos administradores. Nesse sentido, a utilização de regras, raciocínio baseado em casos, objetivos e funções de utilidade vêm sendo adotadas na implementação

[2] *Structured Query Language*

de políticas para sistemas autônomos [58; 39; 47; 29]. Tais implementações revelam que o raciocínio é bem empregado em determinadas situações que se adequam à abordagem utilizada. Por exemplo, em situações onde o estado dos dispositivos é facilmente caracterizado, a utilização de abordagens com raciocínio baseado em objetivos e funções de utilidade, normalmente, apresentam desempenho satisfatório. Em outros casos, onde o estado dos dispositivos não é conseguido, a utilização de abordagens baseadas em regras é aplicável [39].

Diante das inúmeras abordagens para adoção de raciocínio sobre as informações de contexto com a finalidade de criar aspectos autônomos, a concepção de uma infra-estrutura que pudesse embarcar mecanismos de raciocínio distintos poderia unir as situações em que tais mecanismos apresentem melhor desempenho, deixando o sistema autônomo aplicável em diversos domínios de atuação. Esta característica se torna mais importante quando tal sistema está inserido em um ambiente de computação pervasiva, pois a interação não antecipada entre humanos e dispositivos promove situações imprevisíveis.

Neste sentido, a infra-estrutura para o desenvolvimento de aplicações pervasivas autoconfiguráveis apresenta uma arquitetura onde diversos mecanismos de raciocínio, chamados de **engenhos de políticas**, podem ser utilizados, inclusive simultaneamente. Assim, é possível a criação de um engenho que utiliza o raciocínio baseado em regras enquanto outro engenho utiliza uma abordagem com funções de utilidade. Através de uma interface única, estes engenhos devem tratar a inserção de novas políticas e, com elas, desempenhar o raciocínio de acordo com as informações recebidas pelo Módulo de Ciência de Contexto. Como apresentando na seção anterior, este raciocínio pode ser apoiado utilizando consultas ao banco de dados que o Módulo de Representação do Ambiente utiliza para representar o estado atual do ambiente.

A utilização de diversos engenhos de políticas simultaneamente requer um componente que possa distribuir as mensagens recebidas pelo Módulo de Ciência de Contexto bem como direcionar as políticas que são cadastradas no sistema. Afinal, não faz sentido que um engenho com raciocínio baseado funções de utilidade para desempenhar seu raciocínio utilize uma regra comum no seu trabalho. O componente responsável por tratar este cadastro de políticas de acordo com o engenho específico, denominado um Gerente de Políticas, é detalhado a seguir.

Gerente de Políticas

Como descrito na seção anterior, o Gerente de Políticas é o componente responsável por prover uma interface para o cadastramento de políticas para a infra-estrutura proposta neste trabalho. Esta interface indica que o cadastramento de políticas deve descrever a política a ser cadastrada e o mecanismo que deve tratar a mesma. Para isso, o gerente de políticas provê um identificador único para cada engenho de política utilizado pela infra-estrutura. Além disso, o gerente de políticas divulga as informações recebidas pelo Módulo de Ciência de Contexto para todos os engenhos de políticas. Isto permite que os engenhos possam implementar seu próprio mecanismo de representação do estado do ambiente fazendo com que ele desempenhe o raciocínio sobre as políticas cadastradas toda vez que o estado se altere.

Para que as políticas cadastradas na infra-estrutura tenham uma finalidade comum na busca da auto-configuração em ambientes de computação pervasiva, toda política deve indicar o que deve ser feito caso seja satisfeita. Como a infra-estrutura será executada em uma arquitetura orientada a serviços, o resultado da verificação de uma determinada política deve indicar um serviço a ser executado. Por exemplo, uma política baseada em regra pode indicar que, quando uma pessoa entrar numa sala, a infra-estrutura deve executar o serviço para acender as luzes da sala. Assim, ao receber uma tripla RDF do Módulo de Ciência de Contexto que indique a entrada de uma pessoa nesta sala, o Gerente de Políticas repassa tal informação para o engenho que detém esta regra, que por sua vez indica ao gerente de políticas que o serviço para acender as luzes da sala deve ser executado. É certo que, em situações como esta, vários engenhos de política poderiam inferir que as luzes da sala deveriam ser acesas. Nestes casos, é necessário um componente que faça o planejamento correto para que o serviço para acender as luzes não seja executado várias vezes. Para desempenhar este papel, a infra-estrutura proposta neste trabalho define um Módulo de Planejamento e Execução, que será detalhado a seguir.

3.2.4 Módulo de Planejamento e Execução

A utilização de inúmeros engenhos permite que diversas políticas sejam utilizadas no raciocínio de uma mesma situação ocorrida no ambiente de computação pervasiva. No exem-

plo apresentado na seção anterior, onde as luzes de uma sala são ligadas quando alguém entra na mesma, podem ser cadastradas diversas políticas, para diversos engenhos, que realizem a operação para executar o serviço que acende as luzes da sala. Para evitar a redundância na execução de serviços, diminuindo atitudes errôneas e aumentando o desempenho da infra-estrutura, é definido o Módulo de Planejamento e Execução. Este módulo apresenta dois componentes principais: um **Gerente de Planejamento** e um **Gerente de Execução**.

O Gerente de Planejamento é o componente responsável por tratar as requisições de execução de serviços enviadas pelo Gerente de Políticas no Módulo de Tratamento de Políticas. Estas requisições são originadas a partir da verificação de políticas em diversos engenhos definidos neste módulo. Uma vez que diversos engenhos verifiquem políticas que indiquem a execução de um mesmo serviço o Gerente de Planejamento repassa os dados do serviço a ser executado uma única vez para o Gerente de Execução.

Definidos os serviços a serem executados, o Gerente de Execução verifica se os mesmos estão disponíveis no ambiente de computação pervasiva, afinal a dinamicidade dos dispositivos neste ambiente fazem com que os serviços estejam disponíveis ou não para serem executados. Esta verificação deve ser realizada com algum protocolo de descoberta de serviços utilizado na arquitetura orientada a serviços apresentada no Capítulo 2. Uma vez verificados, o Gerente de Execução executa os serviços dispostos no ambiente e persiste a informação em um banco de dados de histórico indicando o sucesso na execução. Caso o serviço não seja encontrado é inserida a informação de falha no histórico. Estas informações permitem que o administrador do sistema possa, caso necessário, atualizar as políticas cadastradas no Módulo de Tratamento de Políticas a fim de se obter um melhor desempenho nos requisitos de auto-configuração do sistema.

3.2.5 Módulo de Fachada

O Módulo de Fachada tem como função prover um único ponto de comunicação entre os provedores de informação e serviços com os módulos de Ciência de Contexto e Tratamento de Políticas. Esta comunicação será satisfeita através de serviços providos pelo Módulo de Fachada que permitirão a entrada de informações e políticas. Neste caso, as informações enviadas e as políticas para cadastramento serão direcionadas para o Gerente de Monitoração e para o Gerente de Políticas respectivamente.

A utilização de um módulo de fachada permite ainda que a comunicação entre as entidades externas (Provedores de Informação e Serviços) e a infra-estrutura apresentada neste trabalho seja implementada utilizando diversas tecnologias de comunicação existentes, como RMI [5], XML-RPC [31] e CORBA [36].

3.3 Projeto de Baixo Nível

Nesta seção, descreve-se como a infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis deve ser implementada apresentando um projeto de baixo nível. Neste projeto são detalhados os módulos da infra-estrutura, apresentando classes, métodos e relacionamentos entre eles.

3.3.1 Módulo de Representação do Ambiente

Como apresentado na seção anterior, o Módulo de Representação do Ambiente define um conjunto mínimo de entidades e predicados capaz de representar formalmente o estado e as interações presentes no ambiente de computação pervasiva. Para proporcionar esta representação são definidos dois elementos principais que se complementam: o Modelo de Entidades e um conjunto de predicados. Além destes, este módulo provê um Mecanismo de Persistência que armazena o estado atual do ambiente de computação pervasiva.

Como relação entre pessoas, dispositivos e localidades está presente em grande parte dos ambientes de computação pervasiva, um modelo base foi definido na infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis utilizando estas entidades principais. O Modelo de Entidades, o conjunto de predicados e o mecanismo de persistência são descritos a seguir.

Modelo de Entidades

Na Figura 3.2, ilustra-se um diagrama de classes com as entidades definidas na infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Estas classes são detalhadas a seguir:

- ***Context Entity***: classe abstrata que contém atributos comuns a todas as entidades pre-

sentes no ambiente de computação pervasiva. Como atributos principais estão presentes um identificador único para a representação da entidade e um conjunto de atividades vinculadas a entidade de contexto. Estas atividades são definidas pela classe descrita a seguir.

- ***Activity***: classe que representa uma determinada atividade ou estado de acordo com a entidade concreta que está vinculada à mesma. Como exemplo, uma *Activity* poderia informar que uma pessoa está em reunião, que as luzes de uma determinada sala estão ligadas ou ainda que um celular está configurado com alerta vibratório.

- ***Location***: classe que representa um espaço físico no ambiente de computação pervasiva. Esta classe possui um nome e deve representar uma sala ou cômodo de uma edificação.

- ***Person***: classe que representa uma pessoa presente no ambiente de computação pervasiva. Esta classe possui um atributo que representa o nome da pessoa.

- ***Device***: classe abstrata que representa um dispositivo disposto no ambiente. Um *Device* possui referência para uma entidade *Location*, informando que o dispositivo está presente em uma determinada localidade. Esta classe contém um atributo que representa o modelo do dispositivo e é estendida pelas classes *Personal Device* e *Equipment* descritas a seguir.

- ***Personal Device***: classe que representa um dispositivo de uso pessoal como um *notebook*, celular ou PDA. Um *Personal Device* possui uma referência para uma instância de *Person*, definindo que o dispositivo de uso pessoal está de posse de uma determinada pessoa. É importante observar que esta relação permite inferir a localidade em que a pessoa está uma vez que seu *Personal Device* possui referência para uma *Location*.

- ***Equipment***: classe que representa um equipamento que não possui mobilidade e que está em uma determinada localidade. Equipamentos como lâmpadas, condicionadores de ar e fechaduras podem ser instanciados através desta classe.

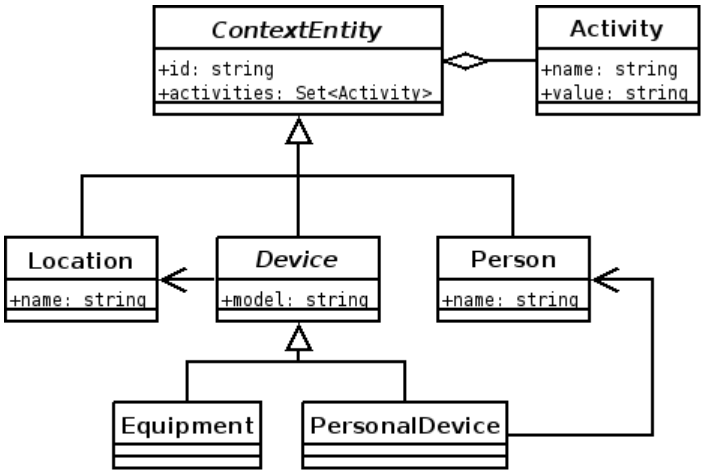

Figura 3.2: Modelo de entidades

Conjunto de Predicados

O Módulo de Representação do Ambiente define um conjunto de predicados de acordo com as ações ocorridas no ambiente de computação pervasiva. Como descrito anteriormente, estes predicados, quando combinados com um Sujeito e um Objeto, definem uma tripla RDF que representa uma informação ocorrida no ambiente. Estas informações são enviadas pelos provedores de informação para o Gerente de Monitoração. Neste caso a infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis provê um conjunto mínimo de predicados comuns a todos os ambientes de computação pervasiva. Estes predicados são descritos na Tabela 3.1.

Mecanismo de Persistência

A utilização do Modelo de Entidades supracitado permite a criação de um banco de dados com as entidades para pessoas, localidades e dispositivos, representando o estado atual do ambiente de computação pervasiva. Este banco de dados pode ser definido através de um esquema armazenado com um Sistema Gerenciador de Banco de Dados (SGBD) relacional comum. Utilizando este sistema, o Mecanismo de Persistência mantém um componente responsável por conectar a infra-estrutura com o SGBD relacional através de uma classe que provê uma conexão única utilizando o padrão *Singleton* [17]. Através desta classe é possível:

1. Atualizar o banco de dados através das informações recebidas pelo Módulo de Ciência

Predicado	Descrição
`enter_location`	Indica que uma determinada pessoa (*Person*) ou dispositivo (*Device*) está entrando em uma determinada localidade (*Location*).
`exit_location`	Indica que uma determinada pessoa (*Person*) ou dispositivo (*Device*) está saindo de uma determinada localidade (*Location*).
`turn_on`	Indica que um determinado dispositivo (*Device*) foi ligado.
`turn_off`	Indica que um determinado dispositivo (*Device*) foi desligado.
`available`	Indica que uma determinada pessoa (*Person*) ou dispositivo (*Device*) está disponível para interagir com os dispositivos do ambiente.
`busy`	Indica que uma determinada pessoa (*Person*) ou dispositivo (*Device*) não está disponível para interagir com os dispositivos do ambiente.
`moving`	Indica que uma determinada pessoa (*Person*) ou equipamento (*Equipment*) está se movimentando em direção à uma determinada localidade (*Location*).

Tabela 3.1: Lista dos predicados utilizados na infra-estrutura

de Contexto;

2. Acessar o banco de dados através de uma interface de consulta. Esta interface apoiará os engenhos de políticas no raciocínio sobre as condições do ambiente e com as políticas que devem ser tomadas;

3. Manipular as informações do banco de dados a partir das informações inferidas pelos engenhos de políticas;

4. Manipular as informações do banco de dados através de alguma aplicação específica de manutenção por um administrador humano.

Na Figura 3.3 apresenta-se a classe *DatabaseSingleton* que implementa o padrão *Singleton* com as funcionalidades necessárias para a manipulação do banco de dados. Seus métodos são detalhados a seguir:

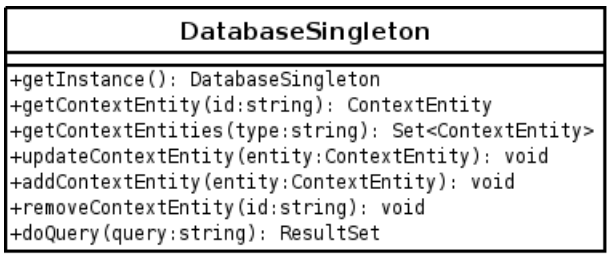

Figura 3.3: Classe *DatabaseSingleton*

- `getInstance`: método fábrica para a implementação do *Singleton*.
- `getContextEntity`: resgata do banco uma entidade do contexto através do identificador passado por parâmetro.
- `getContextEntities`: resgata todas as entidades do banco de acordo com o tipo passado por parâmetro. O tipo deve indicar uma entidade do modelo.
- `updateContextEntity`: atualiza no banco a entidade de contexto repassada como parâmetro.
- `removeContextEntity`: remove do banco a entidade de contexto que possui o identificador passado como parâmetro.
- `doQuery`: invoca uma consulta ao banco de dados e retorna o resultado da mesma.

É importante observar que os métodos desta classe não estão acoplados à nenhuma entidade do Modelo de Entidades. Todas as referências às classes do modelo são realizadas por parâmetros do tipo `string`, permitindo que a infra-estrutura evolua sem mudanças nesta classe. Além disso, o mecanismo de persistência apresenta uma estrutura estensível o suficiente para ser implementado utilizando diversas abordagens para a representação do ambiente. Neste caso, abordagens que utilizam bancos de dados orientados a objetos e até ontologias poderiam ser utilizadas como forma de representação do ambiente. Para isto, seria necessária apenas a criação de tradutores e mecanismos de consulta apropriados.

3.3.2 Módulo de Ciência de Contexto

Como discutido anteriormente, o Módulo de Ciência de Contexto é responsável por prover mecanismos para a aquisição, identificação e tradução das informações enviadas pelos pro-

vedores de informação dispersos no ambiente de computação pervasiva. Para desempenhar tal papel, este módulo define três classes principais que se relacionam como ilustrado no diagrama de classes apresentado na Figura 3.4. A seguir são detalhadas as classes *MonitorManager*, *TraslatorManager* e *Translator* que compõem o Módulo de Ciência de Contexto.

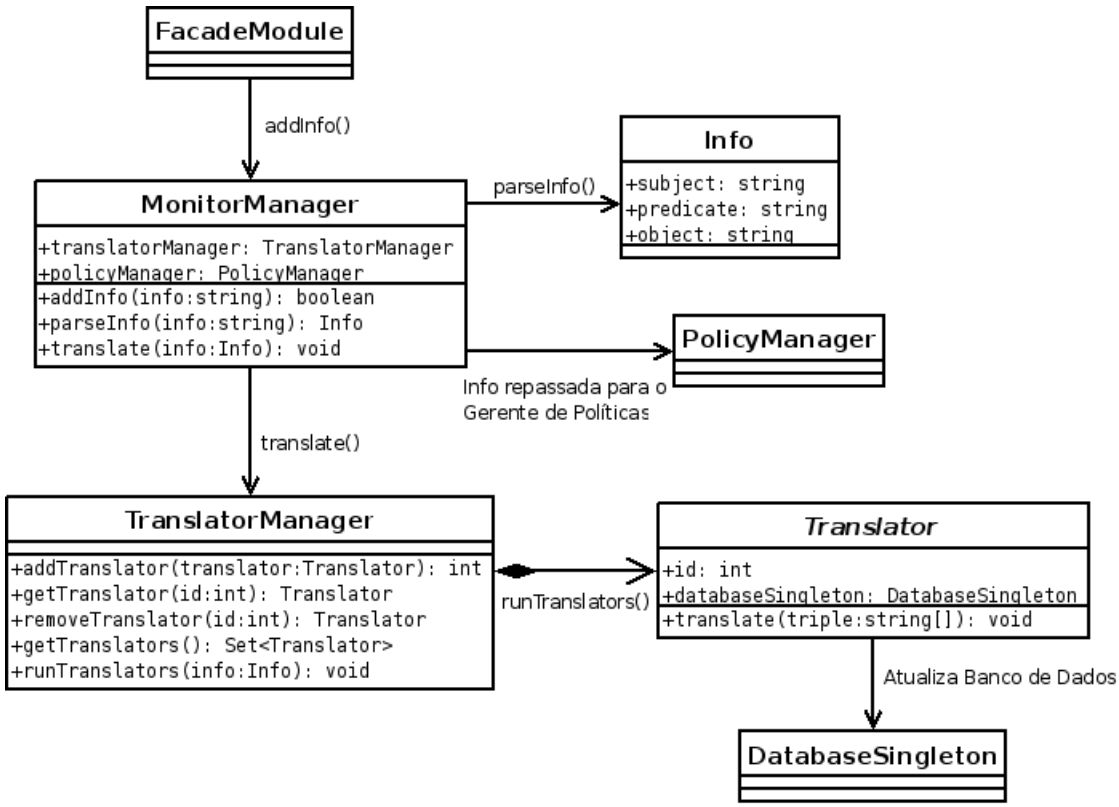

Figura 3.4: Principais classes do módulo de Ciência de Contexto

MonitorManager

A classe *MonitorManager* define o Gerente de Monitoração apresentado anteriormente. Ela é responsável por: receber as informações dos provedores de informação; validá-las de acordo com o formato da mensagem recebida; identificar sujeito, predicado e objeto na mensagem; e repassar esta tripla tanto para o Gerente de Tradução quanto para o Gerente de Políticas. Os métodos da classe *MonitorManager* são descritos a seguir:

- `addInfo`: implementa a entrada de informações originadas pelos provedores de informação no ambiente. Ele é chamado a partir de uma classe do Módulo de Fachada que será apresentada posteriormente. Como parâmetro deve ser indicada uma

`string` contendo a tripla RDF. Após o recebimento, o método deve verificar se *string* de entrada contém uma tripla RDF válida. O retorno deste método define um atributo *boolean* que indica o sucesso ou falha na validação da tripla.

- `parseInfo`: implementa um *parser* que será utilizado para validar e identificar os elementos da tripla RDF recebida pelo módulo. Após a verificação é retornada uma instância da classe *Info* que encapsula sujeito, predicado e objeto da tripla RDF.

- `translate`: responsável por repassar a tripla obtida através do método `parseInfo` para o Gerente de Tradução.

TranslatorManager

A classe *TranslatorManager* define o Gerente de Tradução apresentado na Seção 3.2.2. Ela é responsável por atualizar o banco de dados que representa o estado do ambiente a partir das informações enviadas pelos provedores de informação e trabalhadas no Gerente de Monitoração. Sua implementação mantém que um conjunto de Tradutores deve ser implementado para que a tradução seja realizada. Estes tradutores são implementados estendendo a classe abstrata *Translator* e podem ser adicionados e removidos do *TranslatorManager*. Isto permite que o gerenciamento de tradutores possa ser feito sem a necessidade de modificações complexas na infra-estrutura apresentada neste trabalho. Os métodos da classe *TranslatorManager* são detalhados a seguir:

- `addTranslator`: adiciona um novo tradutor ao Gerente de Tradução. Seu retorno é um número inteiro que identifica o tradutor.

- `getTranslator`: resgata uma referência para o tradutor a partir do identificador passado como parâmetro.

- `removeTranslator`: remove o tradutor que possui o identificador passado como parâmetro. Uma referência do tradutor é retornada caso ele exista.

- `getTranslators`: resgata todos os tradutores cadastrados no Gerente de Tradução.

- `runTranslators`: repassa a tripla recebida para todos os tradutores cadastrados, para que eles façam o mapeamento para a atualização do banco de dados que representa o ambiente.

Translator

A classe abstrata *Translator* representa um tradutor que pode ser adicionado ao Gerente de Tradução. As classes que estendem *Translator* devem implementar o método descrito a seguir:

- `translate`: realiza a tradução da tripla RDF recebida para atualização do banco de dados que representa o ambiente de computação pervasiva. Esta atualização deve ser feita utilizando a referência da classe *DatabaseSingleton* definida como atributo da classe *Translator*.

3.3.3 Módulo de Tratamento de Políticas

O Módulo de Tratamento de Políticas é o principal módulo na busca da auto-configuração em ambientes de computação pervasiva. Com ele é possível cadastrar engenhos de políticas, cadastrar políticas e com estes realizar o raciocínio necessário para prover a auto-configuração nos sistemas pervasivos. Na Figura 3.5, ilustra-se o relacionamento entre as classes do Módulo de Tratamento de Políticas. As principais classes deste módulo (*PolicyManager* e *PolicyEngine*) são detalhadas a seguir:

PolicyManager

A classe *PolicyManager* representa o Gerente de Políticas definido anteriormente. O Gerente de Políticas implementa interfaces para o cadastramento de engenhos de política, o cadastramento de políticas e para a divulgação de informações originadas do Módulo de Ciência de Contexto para os engenhos cadastrados. Neste caso, as informações enviadas pelo Gerente de Monitoração são repassadas para os Engenhos de Políticas a fim de encontrar alguma política que indique que a auto-configuração é necessária. Caso encontre alguma política, estas são repassadas para o Módulo de Planejamento e Execução, para que eles reflitam a auto-configuração no ambiente. Os principais métodos desta classe são apresentados abaixo:

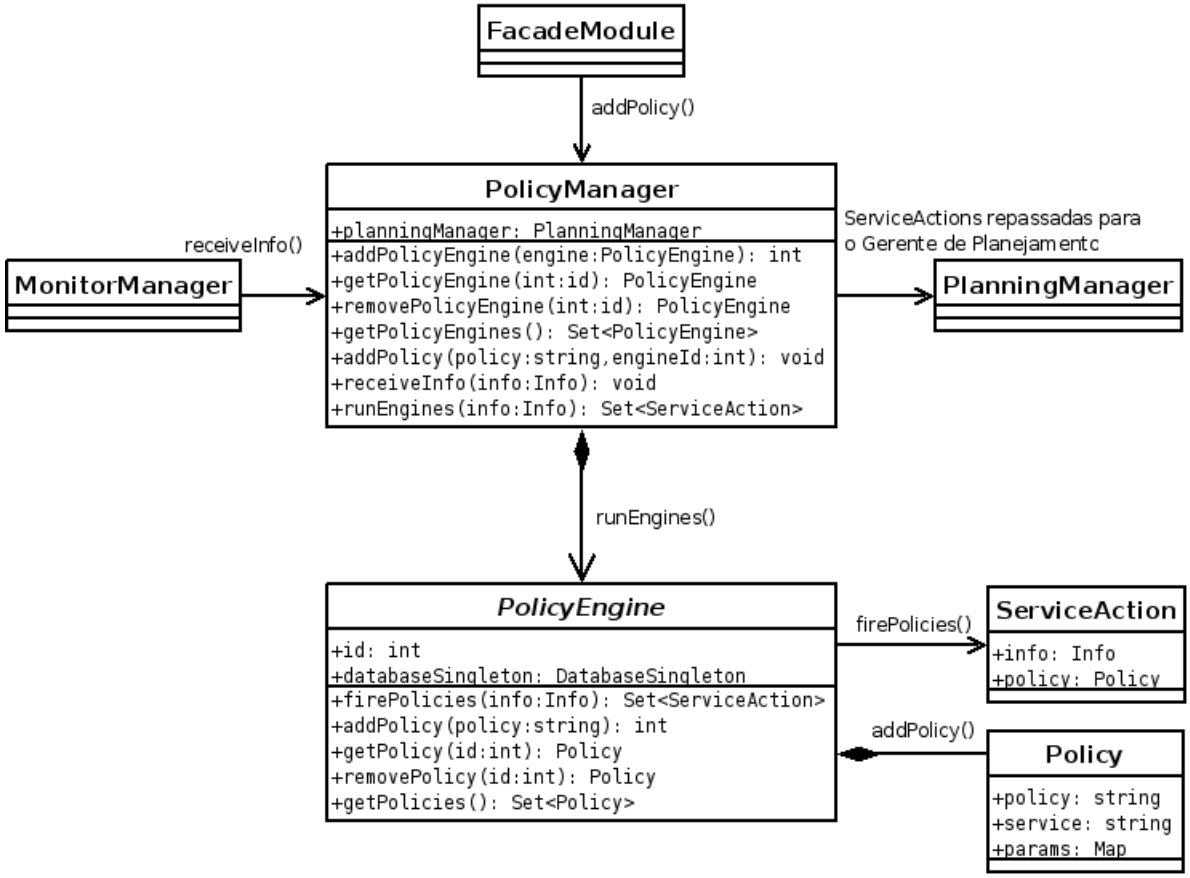

Figura 3.5: Principais classes do módulo de Tratamento de Políticas

- `addPolicyEngine`: cadastra um novo Engenho de Política (*PolicyEngine*) no Gerente de Políticas. Este método retorna um número inteiro que identifica o engenho cadastrado.

- `getPolicyEngine`: resgata um específico engenho de política cadastrado que possua o identificador passado por parâmetro.

- `removePolicyEngine`: remove um engenho de política cadastrado que possua o identificador passado por parâmetro. Uma referência do engenho é retornada caso exista.

- `getPolicyEngines`: resgata todos os engenhos de política cadastrados no Gerente de Políticas.

- `addPolicy`: adiciona uma nova política ao engenho de política cadastrado que pos-

sui o identificador passado pelo parâmetro `engineId`. Neste ponto, a política é passada na forma de `string` e deve ser verificada, validada e interpretada pelos engenhos.

- `receiveInfo`: recebe uma informação enviada pelo Gerente de Monitoração.
- `runEngines`: indica que todos os engenhos cadastrados devem procurar alguma política de auto-configuração de acordo com a informação passada pelo parâmetro. Na execução deste método, as políticas disparadas para a realização da auto-configuração são encapsuladas junto com a informação com instâncias da classe *ServiceAction*. As instâncias de *ServiceAction* criadas por todos os engenhos de políticas, são reunidas pelo Gerente de Políticas e repassadas para o Gerente de Tradução (*PlanningManager*).

PolicyEngine

A classe *PolicyEngine* representa o Engenho de Políticas definido na Seção 3.2.3. Um Engenho de Política é responsável por verificar as situações em que a auto-configuração é necessária através de políticas que podem ser adicionadas a ele. O engenho de políticas deve utilizar as informações recebidas pelo Gerente de Políticas e, juntamente com consultas ao banco de dados que representa o ambiente, verificar as situações onde a auto-configuração é indicada, baseado nas políticas cadastradas no mesmo. A seguir são detalhados os métodos desta classe:

- `addPolicy`: cadastra uma nova política ao engenho de políticas. Este método retorna um número inteiro que identifica a política cadastrada.
- `getPolicy`: resgata uma política cadastrada que possua o identificador passado pelo parâmetro.
- `removePolicy`: remove uma política cadastrada que possua o identificador passado pelo parâmetro. Uma referência para a política é retornada caso ela exista.
- `getPolicies`: resgata todas as políticas cadastradas no engenho de políticas.

- `firePolicies`: verifica quais políticas são satisfeitas indicando a necessidade de auto-configuração no ambiente. Tais políticas são retornadas junto com a informação enviada como parâmetro em uma instância da classe *ServiceAction*.

3.3.4 Módulo de Planejamento e Execução

Após a verificação no módulo anterior, as políticas de auto-configuração são repassadas para o Módulo de Planejamento e Execução. Este módulo é responsável por executar as ações (serviços), definidas nas instâncias de *ServiceAction*, no ambiente de computação pervasiva. Antes da execução, são verificadas as situações de conflito, redundância e erro nas instâncias recebidas pelo Gerente de Políticas. Para realizar este papel, duas classes principais são implementadas no Módulo de Planejamento e Execução: *PlanningManager* e *ExecutionManager*, como ilustrado na Figura 3.6. Estas classes são detalhadas a seguir.

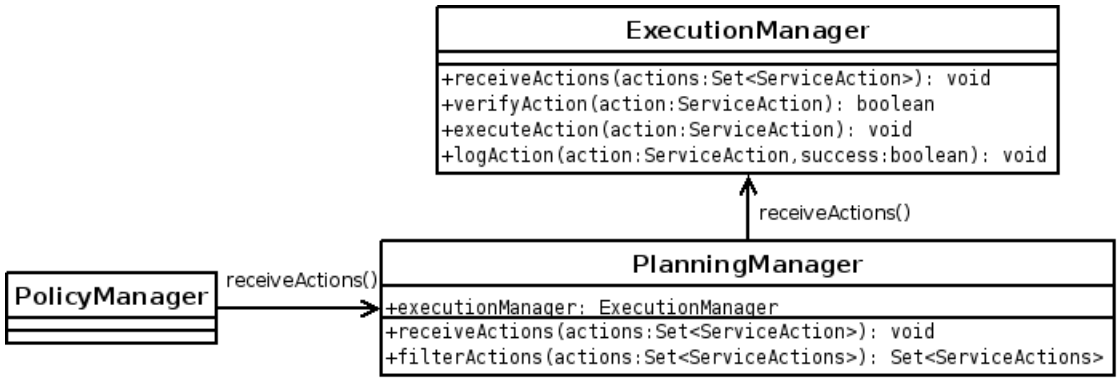

Figura 3.6: Principais classes do módulo de Planejamento e Execução

PlanningManager

A classe *PlanningManager* implementa o Gerente de Planejamento descrito na Seção 3.2.4. Ela é responsável por receber as ações enviadas pelo Gerente de Políticas, verificar os possíveis conflitos que existam nas mesmas, repassando um subconjunto de ações sem conflitos para o Gerente de Execução. Seus métodos são descritos a seguir:

- `receiveActions`: recebe as ações definidas pelos Engenhos de Política e repassadas pelo Gerente de Políticas do Módulo de Tratamento de Políticas.

- `filterActions`: verifica se ações recebidas pelo parâmetro possuem algum conflito caso fossem executadas no ambiente. Normalmente é verificado se existe a redundância de ações, uma vez que os engenhos de política trabalham de forma independente, sem o conhecimento dos outros engenhos. Um subconjunto de ações com os conflitos resolvidos é retornado pelo método.

ExecutionManager

A classe *ExecutionManager* implementa o Gerente de Execução descrito na Seção 3.2.4. Esta classe verifica se os serviços, definidos nas ações repassadas pelo Gerente de Planejamento, estão disponíveis no ambiente de computação pervasiva. Após a verificação, os serviços são finalmente executados, provendo a auto-configuração no ambiente. Além disto, é criado um histórico com as ações de sucesso e de falha, para que um administrador humano possa verificar o funcionamento da infra-estrutura. Os principais métodos desta classe são descritos a seguir:

- `receiveActions`: recebe as ações sem conflitos repassadas pelo Gerente de Planejamento.
- `verifyAction`: verifica se o serviço definido na ação passada como parâmetro está disponível no ambiente de computação pervasiva.
- `executeAction`: executa o serviço definido na ação recebida pelo parâmetro.
- `logAction`: persiste a informação de execução da ação recebida pelo parâmetro definindo seu sucesso ou falha no histórico do sistema.

3.3.5 Módulo de Fachada

Como apresentado na Seção 3.2.5, o Módulo de fachada é responsável por disponibilizar uma interface de comunicação entre os dispositivos dispersos no ambiente e a infra-estrutura definida neste trabalho. Através de serviços que permitem a entrada de informações e cadastramento de políticas, este módulo provê a comunicação entre o ambiente externo e os Gerentes de Monitoração e Políticas. A classe de fachada deste módulo é ilustrada na Figura 3.7 e suas funcionalidades são detalhadas a seguir:

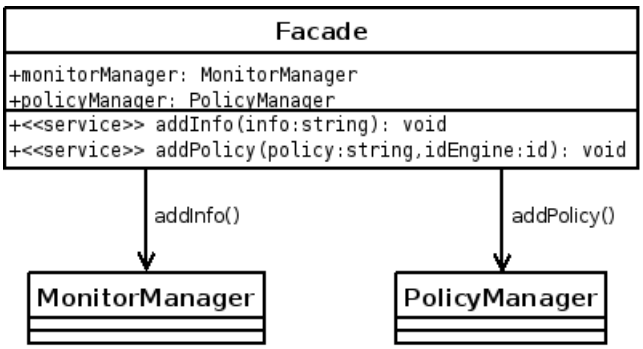

Figura 3.7: Principais classes do módulo de Fachada

- `addInfo`: adiciona uma nova informação que deve ser repassada para o Gerente de Monitoração. Esta funcionalidade é implementada na forma de serviço que recebe uma `string` de informação como entrada.

- `addPolicy`: adiciona uma nova política a um determinado engenho de política. Tanto a política, como o identificador do engenho (recebidos como parâmetros) são repassados para o Gerente de Políticas.

3.4 Funcionamento da Infra-estrutura

Como discutido anteriormente, o funcionamento da infra-estrutura definida neste trabalho depende de uma fase de configuração que compreende a inserção de tradutores, engenhos e cadastramento de políticas de alto nível. Nesse sentido, a utilização da infra-estrutura pode ser realizada através de três atores diferentes: o desenvolvedor, o administrador e o usuário.

Para o desenvolvedor é necessária a implementação de engenhos de política que trabalhem com os provedores de informação utilizados na infra-estrutura. Neste caso, o desenvolvedor terá que verificar todas as aplicações que podem publicar informações relevantes sobre o ambiente e adaptar os engenhos de política para trabalhar com estas informações. Caso estes engenhos utilizem o mecanismo de persistência em seu trabalho, é necessária a implementação de tradutores que façam o mapeamento necessário das informações obtidas para a representação do ambiente.

Já o administrador do sistema deve considerar os engenhos de política utilizados pela infra-estrutura para cadastrar políticas de alto nível que ele julgue necessárias para prover a

auto-configuração no ambiente de computação pervasiva. O cadastramento destas políticas, normalmente deve ser feito através de uma interface de administração que se comunica diretamente com o módulo de fachada da infra-estrutura.

Uma vez realizados os trabalhos do desenvolvedor e do administrador, os usuários poderão usufruir e interagir com o ambiente de forma transparente, de acordo com as políticas de auto-configuração definidas. As políticas cadastradas pelo administrador, normalmente respeitam regras e comportamentos adequados ao ambiente que a infra-estrutura está sendo executada. Entretanto, a infra-estrutura permite que os próprios usuários cadastrem políticas para uso próprio. Questões de privacidade e permissões na execução dos dispositivos dispersos no ambiente não são contempladas pela infra-estrutura, mas estão sendo consideradas como trabalhos futuros.

Após a definição de engenhos e o cadastramento de políticas, o funcionamento da infra-estrutura pode ser ilustrado de acordo com a Figura 3.8, seguindo os passos descritos abaixo:

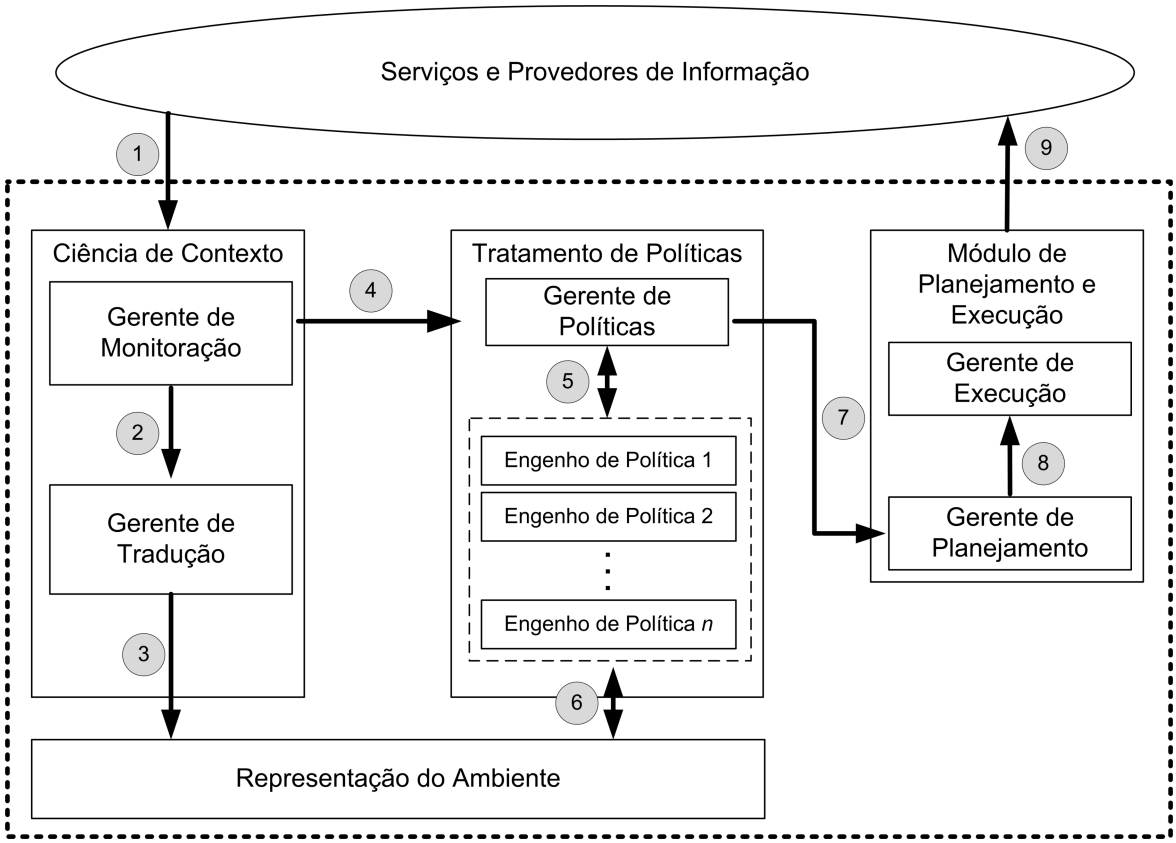

Figura 3.8: Funcionamento da Infra-estrutura

1. Os provedores de informação dispersos no ambiente enviam informações para o Ge-

rente de Monitoração utilizando o serviço provido no Módulo de Fachada. Estas informações, enviadas como tripla RDF no formato `string`, são verificadas, validadas e interpretadas para um formato que indique o sujeito, predicado e objeto da tripla;

2. Após a identificação da informação, a tripla é repassada para o Gerente de Tradução. Neste, os tradutores cadastrados são notificados para traduzir a tripla recebida;

3. Neste ponto o banco de dados que representa o ambiente é atualizado de acordo com a informação traduzida;

4. Depois do banco ser atualizado, a mesma tripla é repassada para o Módulo de Tratamento de Políticas, para que ele analise se esta nova informação deve ser refletida com alguma ação de auto-configuração, de acordo com as políticas de alto nível;

5. O gerente de políticas repassa a tripla para os Engenhos de Políticas cadastrados;

6. Os engenhos, por sua vez, verificam se alguma política cadastrada condiz com a informação contida na tripla recebida. Neste ponto, o raciocínio desempenhado pelo engenho de políticas é apoiado por consultas ao banco de dados que representa o ambiente;

7. Caso os engenhos verifiquem alguma política cadastrada, eles indicam tais serviços para o Gerente de Políticas que, por sua vez, notifica o Gerente de Planejamento para verificá-los. Neste, o conjunto de serviços é reduzido para um subconjunto de serviços sem redundâncias, erros e conflitos;

8. Após a verificação do Gerente de Planejamento, os serviços sem redundâncias, erros e conflitos são repassados para o Gerente de Execução;

9. Por fim, o Gerente de Execução verifica se os serviços recebidos pelo Gerente de Planejamento estão disponíveis no ambiente. Caso estejam disponíveis, eles são executados promovendo a auto-configuração. Os casos de sucesso e falha na execução de serviços são armazenados num histórico do sistema para análise de um administrador humano.

Capítulo 4

Estudo de Caso

Uma vez descritas as características da infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis, através de seus componentes, estrutura e funcionamento, a criação de um estudo de caso que implemente esta infra-estrutura permite validar o trabalho proposto neste documento. Este estudo de caso implementa tal infra-estrutura no Laboratório de Sistemas Embarcados e Computação Pervasiva (*Embedded*) da Universidade Federal de Campina Grande.

Esta implementação, nomeada *Adapt!*, foi desenvolvida através da linguagem Java definindo modelo de entidades, conjunto de predicados, tradutores, engenhos de políticas e provedores de informação na busca da auto-configuração no ambiente físico do *Embedded*. Utilizando uma infra-estrutura de rede UPnP, o *Adapt!* provê uma aplicação web para a administração do sistema, onde é possível verificar o estado do banco de dados que representa o ambiente, adicionar políticas, adicionar informações e verificar quais dispositivos UPnP estão presentes no ambiente.

O *Adapt!* é implementado de acordo com as classes, métodos e relacionamentos apresentados no projeto de baixo nível descrito na Seção 3.3. Entretanto, algumas modificações foram realizadas para que a infra-estrutura definida utilizasse uma infra-estrutura de rede baseada na tecnologia UPnP, detalhada na Seção 2.1.2. A seguir são apresentadas as definições utilizadas na concepção do *Adapt!*, ilustrando dispositivos, tradutores, engenhos de política e a aplicação de administração.

4.1 Modelo de Entidades e Conjunto de Predicados

O Modelo de Entidades considerado no *Adapt!* utiliza o conjunto mínimo de entidades descrito na Seção 3.3 com uma simples alteração na classe abstrata *Device* que, agora, apresenta um identificador do dispositivo UPnP que ele representa. Nesse sentido, a especificação UPnP define que todo dispositivo deve apresentar um identificador único através do atributo `UDN` (*Unique Device Name*). Assim, tal atributo foi adicionado à classe *Device* para que o banco de dados que representa o ambiente de computação pervasiva indique o estado dos dispositivos UPnP dispersos no ambiente. Esta alteração do modelo é ilustrada na Figura 4.1.

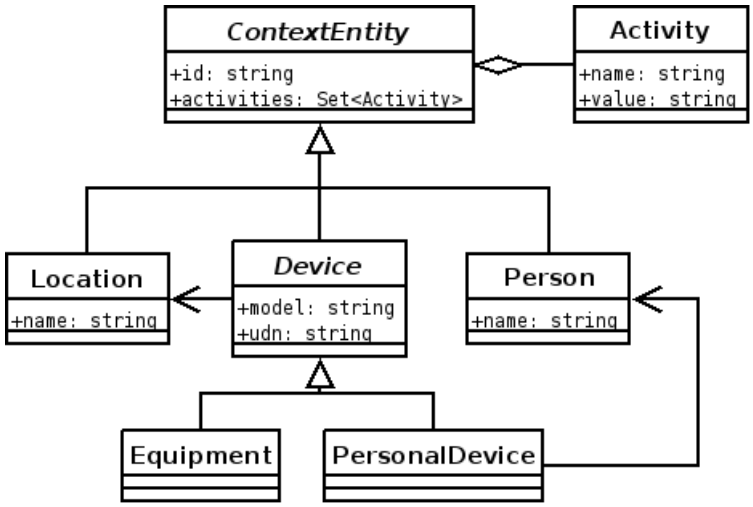

Figura 4.1: Modelo de entidades utilizado no estudo de caso

Através deste modelo o mecanismo de persistência do *Adapt!* foi implementado utilizando o Sistema Gerenciador de Banco de Dados HSQLDB [26] criando tabelas e relacionamentos de acordo com o modelo apresentado na Figura 4.1. Nesse sentido, tradutores e engenhos de políticas podem verificar a situação do banco de dados através de consultas SQL.

Diferentemente do Modelo de Entidades, o Conjunto de Predicados definido pelo *Adapt!* conserva os mesmos predicados definidos na Tabela 3.1.

4.2 Tradutores

Para realizar a tradução das informações recebidas pelos provedores de informação, um simples tradutor foi implementado através da classe java *TranslatorImpl*. Esta classe traduz a informação recebida com os predicados `enter_location` e `exit_location` descritos na Tabela 3.1 e utilizados no conjunto de predicados do *Adapt!*. Ao receber uma informação que define o sujeito como o UDN de um dispositivo UPnP, o predicado é a *string* `enter_location` e o valor é o identificador de uma localidade (*Location*), o tradutor atualiza o banco de dados utilizando a linguagem SQL como apresentado no Código 4.1.

Código Fonte 4.1: Classe *TranslatorImpl*

```
1  package context;
2
3  public class TranslatorImpl extends Translator {
4
5      public void translate(Info info) {
6          // Traducao para o predicado "enter_location"
7          if (info.getPredicate().equals("enter_location")) {
8              String query = "UPDATE PERSONAL_DEVICE pd "+
9                      " SET location_id = " + info.getValue() +
10                     " WHERE pd.udn = " + info.getSubject();
11             this.getDatabase().doQuery(query);
12         }
13         // Outras traducoes
14     }
15
16 }
```

4.3 Engenhos de Políticas

Dois Engenhos de Políticas foram definidos para a utilização do *Adapt!* neste estudo de caso. Um engenho baseado em consultas SQL (*QueryPolicyEngine*) e outro engenho baseado nas informações recebidas pelos provedores de informação (*InfoPolicyEngine*). Estes Engenhos de Políticas são descritos a seguir.

4.3.1 *QueryPolicyEngine*

O Engenho de Políticas baseado em consultas SQL é implementado através da classe *QueryPolicyEngine*. As políticas cadastradas neste engenho, devem indicar uma consulta SQL que deve ser executada sobre o modelo de dados definido no mecanismo de persistência. Caso a consulta definida pela política retorne algum valor como resultado da consulta, a política é verificada e o o serviço indicado na mesma deve ser executado. Esta funcionalidade é definida através do método *firePolicies* apresentado no Código 4.2.

Código Fonte 4.2: Classe *QueryPolicyEngine*

```
1  package policy;
2
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.util.HashSet;
6  import java.util.Set;
7
8  import context.Info;
9
10 public class QueryPolicyEngine extends PolicyEngine {
11
12   private Set<Policy> policies = new HashSet<Policy>();
13
14   public QueryPolicyEngine() {
15     super("Query Policy Engine");
16   }
17
18   public void addPolicy(Policy policy) {
19     this.policies.add(policy);
20   }
21
22   public Set<ServiceAction> firePolicies(Info info) {
23     HashSet<ServiceAction> actions = new HashSet<ServiceAction>();
24     ResultSet rs = null;
25     for (Policy policy : this.policies) {
26       rs = this.getDatabase().doQuery(policy.getPolicy());
27       if (rs.next()) {
```

```
28            actions.add(new ServiceAction(info, policy));
29          }
30        }
31        return actions;
32      }
33  }
```

4.3.2 *InfoPolicyEngine*

O Engenho de Políticas baseado em informações é implementado através da classe *InfoPolicyEngine*. Este engenho, confronta as políticas cadastradas com as informações enviadas pelos Provedores de Informação. Se os valores para sujeito, predicado e valor identificados na tripla enviada pelo provedor forem iguais aos valores descritos em uma política cadastrada, o serviço indicado na mesma deve ser executado. Os elementos sujeito, predicado e objeto são passados nessa ordem em uma *string*, separados pelo marcador (#). Estes elementos também podem assumir o valor (*), indicando que todos os valores são válidos na verificação deste parâmetro. Por exemplo, uma política que define (*) para sujeito, `enter_location` como predicado e "sala-3" como valor define que todas as informações que indicam a entrada de um dispositivo na sala 3 devem ser disparadas pelo engenho. Esta funcionalidade é definida através do método *firePolicies* apresentado no Código 4.3.

Código Fonte 4.3: Classe *InfoPolicyEngine*

```
1  package policy;
2
3  import java.util.HashSet;
4  import java.util.Set;
5  import context.Info;
6
7  public class InfoPolicyEngine extends PolicyEngine {
8
9    private Set<Policy> policies = new HashSet<Policy>();
10
11   public InfoPolicyEngine() {
12     super("Rule Policy Engine");
13   }
```

```
14
15    public void addPolicy(Policy policy) {
16      this.policies.add(policy);
17    }
18
19    public Set<ServiceAction> firePolicies(Info info) {
20      HashSet<ServiceAction> actions = new HashSet<ServiceAction>();
21      String[] parsedPolicy;
22      boolean fired = true;
23      for (Policy policy : this.policies) {
24        fired = true;
25        parsedPolicy = this.parsePolicy(policy.getPolicy());
26        if (!info.getSubject().equals("*") &&
27            !info.getSubject().equals(parsedPolicy[0])) {
28          fired = false;
29        } else if (!info.getPredicate().equals("*") &&
30                   !info.getPredicate().equals(parsedPolicy[1])) {
31          fired = false;
32        } else if (!info.getPredicate().equals("*") &&
33                   !info.getPredicate().equals(parsedPolicy[2])) {
34          fired = false;
35        }
36
37        if (fired) {
38          actions.add(new ServiceAction(info, policy));
39        }
40      }
41      return actions;
42    }
43
44    public String[] parsePolicy(String policy) {
45      return policy.split("#");
46    }
47  }
```

4.4 Planejamento e Execução

O Gerente de Planejamento implementado para o *Adapt!* apenas verifica e remove a redundância entre as instâncias de *ServiceAction* recebidas pelo Gerente de Políticas. Após esta remoção o novo conjutno de ações é repassado para o Gerente de Execução.

Neste ponto, o Gerente de Execução implementa um ponto de controle UPnP, de acordo com a biblioteca *CyberLink* [10], para a execução das ações recebidas pelo Gerente de Planejamento. Os casos de sucesso e falha são armazenados no histórico da aplicação.

4.5 Provedores de Informação

O Laboratório de Sistemas Embarcados e Computação Pervasiva possui diversos dispositivos que podem ser utilizados em ambientes de computação pervasiva. Entre estes dispositivos encontram-se aparelhos celulares, *internet tablets*, roteadores de rede sem fio, etiquetas e leitores de RFID[1] e ainda aparelhos que implementam dispositivos UPnP.

A partir dos recursos oferecidos pelo *Embedded*, alunos de graduação, mestrado e doutorado têm criado projetos que motivam a criação dos ambientes de computação pervasiva. Neste estudo de caso um destes projetos, conhecido como *Embedded Location*, será utilizado como provedor de informação para o *Adapt!*.

O *Embedded Location* implementa um servidor que indica a localidade de pessoas e dispositivos que estão nas salas do *Embedded*. Neste estudo de caso, este servidor foi adaptado para informar ao *Adapt!* a movimentação de dispositivos UPnP entre as salas e corredores do *Embedded*. Estas informações são repassadas como triplas RDF para o *Adapt!* contendo:

- **Sujeito**: indica o `UDN` do dispositivo UPnP que entrou ou saiu de uma determinada localidade;
- **Predicado**: indica o predicado `enter_location` ou `exit_location` para as informações em que o dispositivo tenha entrado ou saído de uma localidade;
- **Valor**: indica a sala ou corredor em que o dispositivo definido no sujeito tenha entrado ou saído de acordo com o predicado indicado.

[1] *Radio-Frequency IDentification*

4.6 Dispositivos UPnP

Uma vez definidos os principais componentes para o *Adapt!* a infra-estrutura está pronta para tratar informações, adicionar políticas e, assim, prover a auto-configuração sobre os dispositivos UPnP dispersos no ambiente de computação pervasiva. Neste estudo de caso, dois dispositivos UPnP foram utilizados como prova de conceito do funcionamento da infra-estrutura. Os atributos, serviços e ações [2] destes dispositivos são descritos a seguir.

4.6.1 Interruptor

O dispositivo Interruptor (*Switch*) define um dispositivo UPnP com duas ações que podem ser executadas. Similar a um interruptor comum, este dispositivo apenas mantém o estado ligado ou desligado, podendo alternar entre os estados através da ação `SetPower`. A ação `GetPower`, verifica o estado atual do interruptor. Este dispositivo foi implementado com a biblioteca CyberLink e seus principais atributos, serviços e ações são apresentados na Tabela 4.1.

Nome	**Valor**
`UDN`	uuid:switch1
`Tipo`	urn:schemas-upnp-org:device:switch:1
`Nome`	Switch
`Serviço 1`	urn:schemas-upnp-org:service:power:1
`Ações do Serviço 1`	`SetPower` e `GetPower`

Tabela 4.1: Características do dispositivo Interruptor

A utilização destes interruptores em dispositivos de hardware [4] permite que aparelhos que não implementam dispositivos UPnP possam participar de um ambiente de computação pervasiva sendo ligados e desligados por um ponto de controle comum. Assim, televi-

[2] A especificação UPnP define que as funcionalidades que são executadas pelos pontos de controle são as ações e não os serviços. Estes, por sua vez agregam um conjunto de ações num determinado dispostivo. Para um melhor entendimento do leitor, nesta seção as referências para serviços e ações seguem a especificação de UPnP.

sores, condicionadores de ar e lâmpadas podem facilmente participar de um ambiente de computação pervasiva com suporte à auto-configuração.

4.6.2 *Media Renderer*

O dispositivo *Media Renderer* implementa um dispositivo UPnP capaz de reproduzir conteúdos de áudio e vídeo através da infra-estrutura de rede. Ele define um conjunto de ações que podem ser executadas pelos pontos de controle, indicando como o conteúdo de áudio ou vídeo deve ser reproduzido. Diferentemente do dispositivo Interruptor, o *Media Renderer* foi implementado através da linguagem Python utilizando a biblioteca BRisa [21]. A utilização de linguagens diferentes na implementação dos dispositivos usados no estudo de caso, evidencia a interoperabilidade na utilização do padrão UPnP e, consequentemente, da infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Os principais atributos, serviços e ações do dispositivo Media Renderer são apresentados na Tabela 4.2.

Nome	**Valor**
`UDN`	uuid:mediarenderer1
`Tipo`	urn:schemas-upnp-org:device:MediaRenderer:1
`Nome`	BRisa Media Renderer
`Serviço 1`	urn:schemas-upnp-org:service:RenderingControl:1
`Ações do Serviço 1`	`ListPresets`, `SelectPreset`, `GetVolume` e `SetVolume`
`Serviço 2`	urn:schemas-upnp-org:service:AVTransport:1
`Ações do Serviço 2`	`Stop`, `Play`, `Pause`, `Seek`, `Next` e `Previous`

Tabela 4.2: Características do dispositivo *Media Renderer*

As ações `Play` e `Pause` implementam funcionalidades para a reprodução de um arquivo de áudio e para a parada desta reprodução respectivamente. Elas são utilizadas neste estudo de caso.

4.7 Interface de Administração

Para facilitar as tarefas de administração do *Adapt!* foi criada uma aplicação web onde um administrador humano pode interagir diretamente com a infra-estrutura implementada. A página principal da aplicação de administração é ilustrada na Figura 4.2 e suas funcionalidades são descritas a seguir.

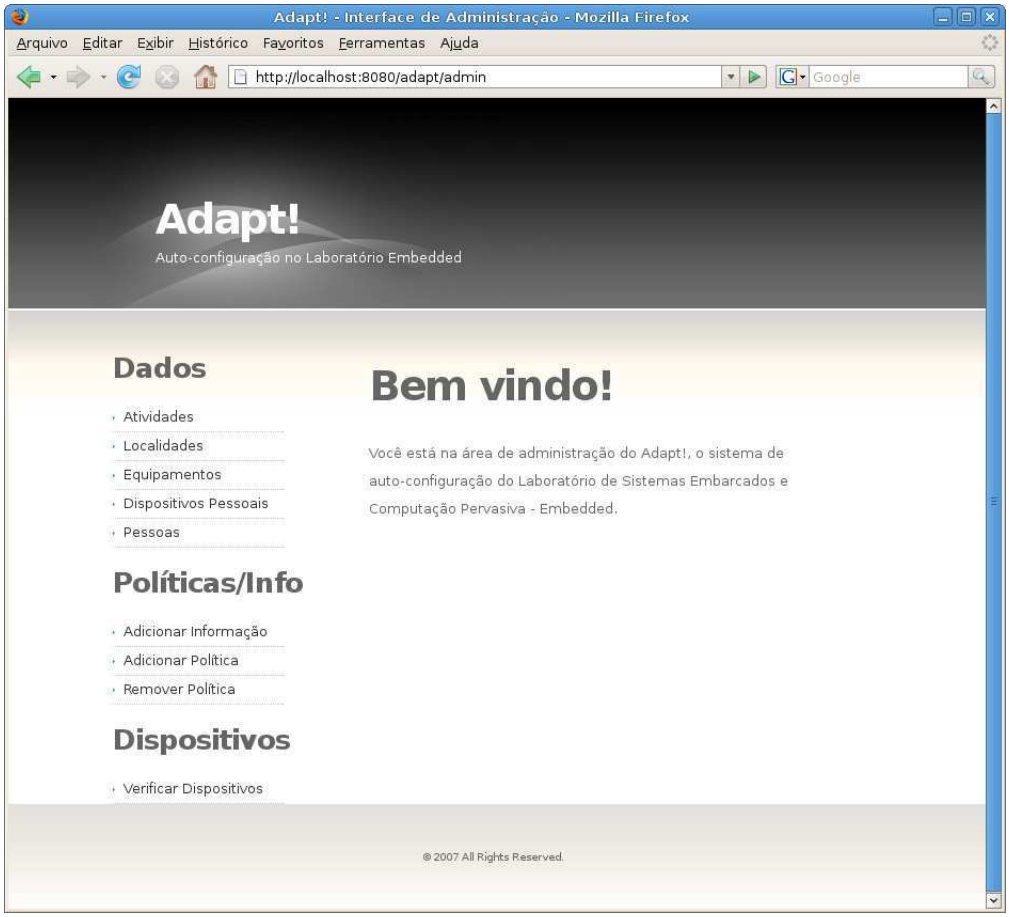

Figura 4.2: Página principal do *Adapt!*

4.7.1 Gerência do Banco de Dados que Representa o Ambiente

Localidades, dispositivos pessoais, equipamentos, atividades e pessoas podem ser cadastradas no banco de dados através da aplicação de administração. Isto permitirá que um administrador humano tenha uma visão mais precisa do estado do ambiente. Em vez de apenas visualizar identificadores complexos, o administrador pode saber qual o modelo do dispositivo que entrou numa determinada sala, quem está de posse deste dispositivo e, possivelmente,

inferir que esta pessoa também entrou na sala com o dispositivo verificado. Prover precisão na representação do ambiente permite que os administradores humanos possam identificar situações onde a auto-configuração é necessária e, assim, criar novas políticas para melhorar o funcionamento do *Adapt!*. Na Figura 4.3, ilustra-se como um dispositivo pessoal é adicionado ao banco de dados utilizando a aplicação de administração.

Figura 4.3: Adicionando um novo dispositivo pessoal ao banco de dados

4.7.2 Gerência de Políticas e Informações

A aplicação de administração permite a gerência de políticas e a adição de informações a partir de uma conexão direta com o módulo de fachada do *Adapt!*. Dessa forma o administrador pode verificar, adicionar e remover regras que estão sendo utilizadas pelos engenhos de política provendo a auto-configuração. Além disso, é possível adicionar informações simulando o comportamento de um provedor de informação disperso no ambiente. Na Fi-

gura 4.4, ilustra-se a inserção da informação que indica que um determinado dispositivo está entrando na "Sala-3" através do predicado `enter_location`.

Figura 4.4: Adicionando uma nova informação ao *Adapt!*

4.7.3 Verificação de Dispositivos UPnP

Nesta seção, o administrador poderá verificar quais são os dispositivos UPnP ativos no ambiente real, apresentando identificadores, serviços, ações e parâmetros necessários para execução. Esta verificação é necessária para que a criação de políticas indique precisamente qual ação deve ser executada para prover a auto-configuração no ambiente de computação pervasiva. Na Figura 4.5, ilustra-se a verificação dos dispositivos *Switch* e *Media Renderer* pela aplicação de administração do *Adapt!*.

Figura 4.5: Dispositivos dispersos no ambiente

4.8 Cenários de Execução

Apresentados os componentes definidos pelo *Adapt!* é possível verificar o funcionamento da infra-estrutura de auto-configuração em ambientes de computação pervasiva, agindo sobre os dispositivos participantes da rede UPnP. Para este estudo de caso serão apresentados dois cenários de execução onde a auto-configuração é evidenciada, adicionando aspectos de computação autônoma em ambientes de computação pervasiva.

A implementação dos cenários de execução utilizaram alguns recursos do Laboratório *Embedded*, como:

- 1 terminal de trabalho com processador Intel Core 2 Duo de 2.13GHz e 1GB de memória RAM. Este terminal foi utilizado para a execução do *Adapt!*.
- 1 Nokia Internet *Tablet* N800 utilizado para executar o dispositivo Interruptor.

- 1 Nokia Internet *Tablet* N800 utilizado para executar o dispositivo *Media Renderer*.

Estes recursos foram utilizados de acordo com os cenários de execução descritos a seguir.

4.8.1 Ligando e Desligando a Lâmpada de uma Sala

Neste cenário de execução a auto-configuração do ambiente deve ser realizada para ligar ou desligar uma lâmpada quando uma pessoa entrar ou sair de uma determinada sala identificada "Sala-3". Na implementação deste cenário, o dispositivo Interruptor foi utilizado para representar uma Lâmpada que deve ser ligada ou desligada de acordo com as ações de "Maria". Para isso algumas políticas de alto nível foram adicionadas ao *Adapt!* a partir da aplicação de administração.

Considerando o provedor de informação implementado com o *Embedded Location*, toda vez que "Maria" entrar na "Sala-3" o provedor de informação irá repassar uma tripla $< Maria, enter\_location, Sala - 3 >$ para o módulo de fachada do *Adapt!* onde esta informação vai ser traduzida, atualizando o banco de dados que representa o ambiente. Assim, a partir de uma consulta SQL a este banco de dados identificando que "Maria" está na "Sala-3", é possível ligar a lâmpada da sala utilizando o engenho de políticas baseado em consultas SQL (*QueryPolicyEngine*). Como o acendimento da lâmpada é realizado através execução da ação `SetPower`, esta política apresentaria parâmetros descritos na Tabela 4.3. O cadastramento desta política pode ser realizado através da aplicação de administração, como ilustrado na Figura 4.6.

`Política (Consulta SQL)`	SELECT * FROM PersonalDevice pd WHERE pd.location = "Sala-3" AND pd.person = "Maria"
`UDN`	uuid:switch1
`Serviço`	urn:schemas-upnp-org:service:power:1
`Ação`	SetPower
`Parâmetros`	Power=1

Tabela 4.3: Política para ligar a lâmpada

Após a criação da política para acender a lâmpada quando "Maria" entrar na "Sala-3", faz-se interessante a criação de uma política que mude o estado da lâmpada para o estado

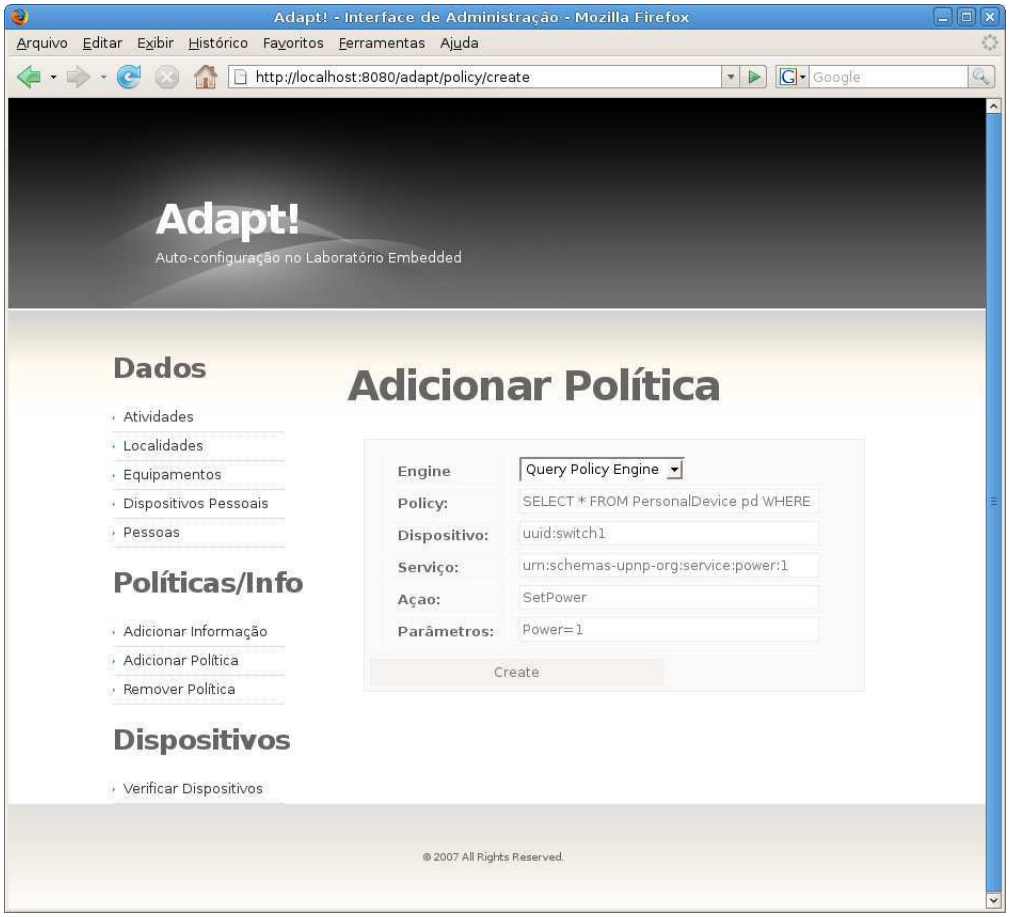

Figura 4.6: Adicionando uma política ao *Adapt!*

de “desligada” quando “Maria” sair da mesma sala. Neste estudo de caso, foi criada uma política para trabalhar com o segundo engenho de política provido pelo `Adapt!`, o engenho baseado em informações *InfoPolicyEngine*. Neste caso, a verificação das políticas acontece de acordo com o pareamento das informações enviadas pelo Provedor de Informação com as políticas cadastradas, como apresentado na Seção 4.3.2. Os dados necessários para a criação desta política são descritos na Tabela 4.4.

É importante notar que esta política se baseia na informação que contém o predicado `exit_location` e que a verificação da mesma executa a ação `SetPower` passando o parâmetro `Power=0`, diferentemente da política anterior para acender a lâmpada.

Após o cadastramento das políticas necessárias para acender a lâmpada quando “Maria” entrar na “Sala-3” quanto para desligar quando ela sair, os testes de execução puderam ser realizados. O *Adapt!* se comportou como esperado, executando a ação da lâmpada sempre que “Maria” entrava ou saia da “Sala-3”, de acordo com as informações enviadas pelo

Política (Informação)	Maria # exit_location # Sala-3
UDN	uuid:switch1
Serviço	urn:schemas-upnp-org:service:power:1
Ação	SetPower
Parâmetros	Power=0

Tabela 4.4: Política para desligar a lâmpada

provedor de informação `Embedded Location`. Durante os testes, uma nova política foi atribuída ao engenho *InfoPolicyEngine* para realizar o mesmo trabalho da política atribuída ao engenho *QueryPolicyEngine* no acendimento da lâmpada. A adição desta nova política foi realizada para verificar o comportamento do Gerente de Planejamento na situação em que "Maria" entrasse na "Sala-3". Analisando o *log* do sistema, verificou-se que nesta situação o Módulo de Planejamento se comportou como esperado, fazendo com que ação `SetPower` fosse executada apenas uma vez, evitando a redundância na execução das ações indicadas pelos engenhos.

4.8.2 Iniciando e Parando a Reprodução de Áudio

Além da criação de políticas para prover a auto-configuração sobre o dispositivo Interruptor, implementado como uma lâmpada, o estudo de caso definido neste trabalho realizou testes utilizando um dispositivo UPnP *Media Renderer*. Estes testes foram necessários para comprovar o funcionamento do *Adapt!* junto a padrões amplamente utilizados no mercado de aparelhos eletrônicos. Afinal, atualmente o dispositivo *Media Renderer* já pode ser encontrado embarcado em aparelhos eletrônicos vendidos em larga escala.

A utilização do dispositivo *Media Renderer* neste cenário de execução define uma política similar à apresentada anteriormente para acender a lâmpada com o dispositivo Interruptor. Neste novo cenário, a auto-configuração é utilizada para reproduzir um arquivo de áudio num determinado dispositivo *Media Renderer* quando qualquer pessoa entrar no corredor do Laboratório *Embedded*. Utilizando novamente as informações providas pelo *Embedded Location* esta nova política poderia ser atribuída ao engenho *InfoPolicyManager* de acordo com os atributos descritos na Tabela 4.5.

Política (Informação)	* # enter_location # Corredor
UDN	uuid:mediarenderer1
Serviço	urn:schemas-upnp-org:service:AVTransport:1
Ação	Play
Parâmetros	URI=<*URI do arquivo de áudio*>

Tabela 4.5: Política para reproduzir o arquivo de áudio

Nesta nova política foi utilizado o símbolo (*) para indicar que o sujeito da informação pode assumir qualquer valor na verificação da mesma. A utilização deste símbolo, juntamente com o predicado `enter_location` e o valor "Corredor" indicam que a ação `Play` encontrada no serviço `urn:schemas-upnp-org:service:AVTransport:1`, do dispositivo de UDN `uuid:mediarenderer1`, deve ser executada sempre que qualquer pessoa entrar no corredor.

Após a inserção da nova política ao `Adapt!` os testes deste cenário de execução foram realizados. Novamente, o sistema de auto-configuração apresentou o comportamento esperado, fazendo com que o dispositivo *Media Renderer* reproduzisse o arquivo de áudio definido como parâmetro da política.

Capítulo 5

Conclusões e Trabalhos Futuros

A presença de tecnologias de rede sem fio em aparelhos como televisores, aparelhos de som, lâmpadas, etc., juntamente com o avanço e popularização dos dispositivos móveis tem motivado a criação dos primeiros ambientes de computação pervasiva. Uma importante característica do paradigma de computação pervasiva é a necessidade de adaptação do ambiente às pessoas que estão inseridas no mesmo de forma transparente, através da automatização de tarefas de acordo com informações contextuais. Esta automatização pode ser promovida utilizando abordagens de computação autônoma, um tópico recente nas discussões da comunidade científica.

Neste trabalho é apresentada uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Como foi detalhado, na concepção desta infra-estrutura foram tratados aspectos de dinamicidade, heterogeneidade, ciência de contexto e limitação dos dispositivos presentes nos ambientes de computação pervasiva. A abordagem adotada na concepção da arquitetura desta infra-estrutura permite a utilização de diversos engenhos de política que podem raciocinar paralelamente sobre as informações contextuais do ambiente.

Como forma de validação, a infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis foi implementada seguindo uma arquitetura orientada a serviços, utilizando a tecnologia UPnP. Esta implementação, denominada *Adapt!*, foi utilizada com sucesso na busca da auto-configuração no Laboratório de Sistemas Embarcados e Computação Pervasiva (*Embedded*) da Universidade Federal de Campina Grande.

Como trabalhos futuros, espera-se estender o *Adapt!* para utilizar um modelo de ontologias para a representação do ambiente, como apresentado em [13; 20]. Ao mesmo tempo

se faz necessária a implementação de novos engenhos de política que utilizam esta ontologia para realizar o raciocínio sobre as políticas de auto-configuração, possivelmente com aprendizado de novas políticas.

Em paralelo, pretende-se investigar a utilização de uma infra-estrutura semelhante na busca dos outros aspectos de computação autônoma, como auto-tratamento, auto-otimização e auto-proteção. Abordagens que contemplam soluções para prover aspectos de segurança e auto-configuração baseada no perfil do usuário também estão entre os trabalhos futuros.

Bibliografia

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *SOSP '99: Proceedings of the seventeenth ACM symposium on operating systems principles* (Charleston, South Carolina, United States, 1999), ACM, pp. 186–201.

[2] AHMED, S., AHAMED, S. I., SHARMIN, M., AND HAQUE, M. M. Self-healing for autonomic pervasive computing. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing* (Seoul, Korea, 2007), ACM Press, pp. 110–111.

[3] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Self-tuning wireless network power management. *Wireless Networks 11*, 4 (2005), 451–469.

[4] AUTOMATED OUTLET. Automated Outlet - Home Automation, Lighting Control Super Store! http://www.automatedoutlet.com/product.php?productid=407, 2008. Acessado em maio de 2008.

[5] AVVENUTI, M., AND VECCHIO, A. Mobilermi: upgrading java remote method invocation towards mobility: Research articles. *Software - Practice & Experience 35*, 10 (2005), 939–975.

[6] BRAY, J., AND STURMAN, C. *Bluetooth: Connect Without Cables*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[7] BUBLITZ, F. M., FILHO, E. C. L., DE ALMEIDA, H. O., DE BARROS COSTA, E., AND PERKUSICH, A. Context-Awareness in Pervasive Environments: An Overview. In *Encyclopedia of Networked and Virtual Organizations*, G. D. Putnik and M. M. Cunha, Eds. Idea Group Publishing, Hershey, PA, EUA, 2007.

[8] CHEN, H., HARIRI, S., KIM, B., ZHANG, Y., AND YOUSIF, M. Self-Deployment and Self-Configuration of Pervasive Network Services. In *ICPS '04: Proceedings of the The IEEE/ACS International Conference on Pervasive Services (ICPS'04)* (Novi Sad, Sérvia e Montenegro, 2004), IEEE Computer Society, pp. 242–242.

[9] CHEN, T., LIKE, Y., AND BIN, X. An implementation of power-aware storage architecture. In *IWCMC '06: Proceedings of the 2006 international conference on Wireless communications and mobile computing* (Vancouver, British Columbia, Canada, 2006), ACM, pp. 1117–1122.

[10] CYBERLINK. CyberLink for Java. http://www.cybergarage.org/net/upnp/java/, 2008. Acessado em maio de 2008.

[11] DANTAS, A., BRENNAND, C., HOZANO, M., AND HERBSTER, R. Computação Autônoma, Outubro 2006. Projeto da disciplina de Inteligência Artificial do curso de Mestrado em Ciência da Computação do DSC/UFCG.

[12] DONG, X., HARIRI, S., XUE, L., CHEN, H., ZHANG, M., PAVULURI, S., AND RAO, S. Autonomia: an autonomic computing environment. In *Proceedings of IEEE Int. Conference on Performance, Computing, and Communications (IPCC)* (2003), pp. 61–68.

[13] EJIGU, D., SCUTURICI, M., AND BRUNIE, L. An ontology-based approach to context modeling and reasoning in pervasive computing. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops* (White Plains, NY, USA, 2007), IEEE Computer Society, pp. 14–19.

[14] ERICKSON, T. Some problems with the notion of context-aware computing. *Communications of the ACM 45*, 2 (2002), 102–104.

[15] ESLER, M., HIGHTOWER, J., ANDERSON, T., AND BORRIELLO, G. Next century challenges: data-centric networking for invisible computing: the portolano project at the university of washington. In *MobiCom '99: Proceedings of the 5th annual ACM / IEEE international conference on Mobile computing and networking* (Seattle, Washington, United States, 1999), ACM, pp. 256–262.

[16] FILHO, E. C. L., BUBLITZ, F. M., OLIVEIRA, L., BARBOSA, N. M., PERKUSICH, A., DE ALMEIDA, H. O., AND DE MELO FERREIRA, G. V. V. Service Provision for Pervasive Computing Environments. In *Encyclopedia of Mobile Computing and Commerce*, D. Taniar, Ed., vol. 2. Information Science Reference, Hershey, PA, EUA, 2007, pp. 877–884.

[17] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.

[18] GARLAN, D., SIEWIOREK, D., SMAILAGIC, A., AND STEENKISTE, P. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing 1*, 2 (2002), 22–31.

[19] GOLAND, Y. Y., CAI, T., LEACH, P., GU, Y., AND ALBRIGHT, S. Simple service discovery protocol/1.0 operating without an arbiter, Outubro 1999.

[20] GU, T., PUNG, H. K., AND ZHANG, D. Q. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications 28*, 1 (January 2005), 1–18.

[21] GUEDES, A., SANTOS, D., DO NASCIMENTO, J., SALES, L., PERKUSICH, A., AND ALMEIDA, H. Set Your Multimedia Application Free with BRisa Framework: An Open Source UPnP Implementation for Resource Limited Devices. *5th IEEE Consumer Communications and Networking Conference* (2008), 1257–1258.

[22] GUTTMAN, E. Autoconfiguration for ip networking: Enabling local communication. *IEEE Internet Computing 5*, 3 (2001), 81–86.

[23] HEDBERG, S. R. News: Beyond Desktop Computing: MIT's Oxygen Project. *IEEE Distributed Systems Online 1*, 1 (2000).

[24] HODES, T. D., CZERWINSKI, S. E., ZHAO, B. Y., JOSEPH, A. D., AND KATZ, R. H. An architecture for secure wide-area service discovery. *Wireless Networks 8*, 2/3 (2002), 213–230.

[25] HORN, P. *Autonomic Computing — IBM's Perspective on the State of Information Technology*. IBM Corporation, Outubro 2001.

[26] HSQLDB. HSQLDB. http://hsqldb.org, 2008. Acessado em maio de 2008.

[27] HUHNS, M. N., AND SINGH, M. P. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing 9*, 1 (2005), 75–81.

[28] KEPHART, J. O., AND CHESS, D. M. The Vision of Autonomic Computing. *Computer 36*, 1 (2003), 41–50.

[29] KEPHART, J. O., AND DAS, R. Achieving Self-Management via Utility Functions. *IEEE Internet Computing 11*, 1 (2007), 40–48.

[30] KLYNE, G., AND CARROLL, J. Resource description framework (RDF): Concepts and abstract syntax. W3C Recomendation. http://www.w3.org/TR/rdf-concepts/. Acessado em maio de 2008.

[31] LAURENT, S. S., DUMBILL, E., AND JOHNSTON, J. *Programming Web Services with XML-RPC*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[32] LEE, J. L., AND SUNWOO, M. H. Implementation of a Wireless Multimedia DSP Chip for Mobile Applications. *Journal of VLSI Signal Processing Systems 40*, 3 (2005), 281–287.

[33] LIN, P., MACARTHUR, A., AND LEANEY, J. Defining Autonomic Computing: A Software Engineering Perspective. In *ASWEC '05: Proceedings of the 2005 Australian conference on Software Engineering* (Brisbane, Australia, 2005), IEEE Computer Society, pp. 88–97.

[34] LOUREIRO, E., BUBLITZ, F., BARBOSA, N., PERKUSICH, A., ALMEIDA, H., AND FERREIRA, G. A Flexible Middleware for Service Provision Over Heterogeneous Pervasive Networks. In *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks* (Niagara-Falls, Buffalo-NY, 2006), IEEE Computer Society, pp. 609–614.

[35] LOUREIRO, E. C. Um Middleware Extensível para Disponibilização de Serviços em Ambientes Pervasivos. Dissertação de Mestrado, Universidade Federal de Campina Grande, julho 2006.

[36] MAFFEIS, S., AND SCHMIDT, D. Constructing reliable distributed communications systems with CORBA. *IEEE Communications Magazine 35*, 2 (1997), 56–61.

[37] MELCHER, B., AND MITCHELL, B. Towards an Autonomic Framework: Self-Configuring Network Services and Developing Autonomic Applications. *Intel Technology Journal 08*, 4 (2004), 279–290.

[38] MENASCE, D. A., AND KEPHART, J. O. Guest Editors' Introduction: Autonomic Computing. *IEEE Internet Computing 11*, 1 (2007), 18–21.

[39] MONTANI, S., AND ANGLANO, C. Achieving self-healing in service delivery software systems by means of case-based reasoning. *Applied Intelligence 28*, 2 (2008), 139–152.

[40] O'DONNELL, T., LEWIS, D., AND WADE, V. Intuitive human governance of autonomic pervasive computing environments. In *WOWMOM '05: Proceedings of the First International IEEE WoWMoM Workshop on Autonomic Communications and Computing (ACC'05)* (Messina, Italy, 2005), IEEE Computer Society, pp. 532–536.

[41] OHRTMAN, F. D., AND ROEDER, K. *Wi-Fi Handbook: Building 802.11b Wireless Networks*. McGraw-Hill, Inc., New York, NY, USA, 2003.

[42] PAPAZOGLOU, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering* (Rome, Italy, 2003), IEEE Computer Society, p. 3.

[43] PAPAZOGLOU, M. P., AND GEORGAKOPOULOS, D. Service-Oriented Computing. *Communications of the ACM 46*, 10 (2003), 24–28.

[44] PARK, J., YOO, G., AND LEE, E. Proactive self-healing system based on multi-agent technologies. In *SERA '05: Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications* (Mt. Pleasant, MI, USA, 2005), IEEE Computer Society, pp. 256–263.

[45] PATOUNI, E., AND ALONISTIOTI, N. A Framework for the Deployment of Self-Managing and Self-Configuring Components in Autonomic Environments. In *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless,*

Mobile and Multimedia Networks (Niagara-Falls, Buffalo-NY, 2006), IEEE Computer Society, pp. 480–484.

[46] PERKINS, C. E., AND MYLES, A. Mobile IP. *Proceedings of International Telecommunications Symposium* (1994), 415–419.

[47] RANGANATHAN, A., AND CAMPBELL, R. H. Autonomic Pervasive Computing Based on Planning. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)* (New York, NY, USA, 2004), IEEE Computer Society, pp. 80–87.

[48] RANGANATHAN, A., AND CAMPBELL, R. H. Self-Optimization of Task Execution in Pervasive Computing Environments. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing* (Seattle, Washington, USA, 2005), IEEE Computer Society, pp. 333–334.

[49] RULNICK, J. M., AND BAMBOS, N. Mobile power management for wireless communication networks. *Wireless Networks 3*, 1 (1997), 3–14.

[50] SAHA, D., AND MUKHERJEE, A. Pervasive Computing: A Paradigm for the 21st Century. *Computer 36*, 3 (2003), 25–31.

[51] SATYANARAYANAN, M. Pervasive computing: vision and challenges. *IEEE Personal Communications 8*, 4 (2001), 10–17.

[52] SHIREHJINI, A. A. N. A generic upnp architecture for ambient intelligence meeting rooms and a control point allowing for integrated 2d and 3d interaction. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence* (Grenoble, France, 2005), ACM, pp. 207–212.

[53] STERRITT, R., PARASHAR, M., TIANFIELD, H., AND UNLAND, R. A concise introduction to autonomic computing. *Advanced Engineering Informatics 19*, 3 (2005), 181–187.

[54] TESAURO, G., AND KEPHART, J. O. Utility functions in autonomic systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)* (New York, NY, USA, 2004), IEEE Computer Society, pp. 70–77.

[55] UPNP FORUM. UPnP Device Architecture. http://www.upnp.org, 2008. Acessado em maio de 2008.

[56] UPNP FORUM. UPnP Forum. http://www.upnp.org, 2008. Acessado em maio de 2008.

[57] WALDO, J. The jini architecture for network-centric computing. *Communications of the ACM 42*, 7 (1999), 76–82.

[58] WANG, Q. Towards a rule model for self-adaptive software. *ACM SIGSOFT Software Engineering Notes 30*, 1 (2005), 8.

[59] WEISER, M. The computer for the twenty-first century. *Scientific American 265*, 3 (1991), 94–104.

[60] WHITE, S., HANSON, J., WHALLEY, I., CHESS, D., AND KEPHART, J. An architectural approach to autonomic computing. *Autonomic Computing, 2004. Proceedings. International Conference on* (2004), 2–9.

[61] ZHU, F., MUTKA, M. W., AND NI, L. M. Service discovery in pervasive computing environments. *IEEE Pervasive Computing 4*, 4 (2005), 81–90.