Universidade Federal Fluminense

DAVID BARRETO FERREIRA

Uma Interface para Prototipagem e Gerenciamento de Aplicações Pervasivas

Niterói

2013

Universidade Federal Fluminense

DAVID BARRETO FERREIRA

Uma Interface para Prototipagem e Gerenciamento de Aplicações Pervasivas

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Orientador:
Orlando Gomes Loques Filho, Ph.D.

Niterói

2013

DAVID BARRETO FERREIRA

Uma Interface para Prototipagem e Gerenciamento de Aplicações Pervasivas

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Aprovada em Março de 2013.

BANCA EXAMINADORA

Prof. Orlando Gomes Loques Filho, Ph.D. IC/UFF – Orientador

Prof. Debora Christina Muchaluat Saade, IC/UFF

Prof. Alexandre Sztajnberg, DICC-UERJ

Niterói

2013

À minha família.

Agradecimentos

Agradeço primeiramente à Deus, que tem me dado forças para superar os desafios encontrados durante a minha jornada no mestrado, e na vida. Agradeço à minha família, que me apoiou, que me criticou, que pressionou e que me fez crescer e ser uma pessoal melhor. Sobretudo, aos meus pais Joême e Maria e irmãos Sandra e Júnior, pelos conselhos e apoio. À minha esposa Meiryelle que tem o seu toque em várias partes deste trabalho, e que por várias vezes abdicou do seu tempo comigo para que este trabalho fosse realizado.

Ao Professor e Orientador Orlando Loques, pelos sábios conselhos e larga experiência divididos comigo. Agradeço pelas broncas e pelos ensinamentos que certamente levarei para o resto da vida. Ao membros da banca, Alexandre Sztajnberg, que há muito tem me ajudado e aconselhado, e Debora Christina Muchaluat Saade por terem aceitado o convite, mesmo em cima da hora.

Aos colegas do Laboratório Tempo, principalmente ao Douglas e Matheus, grandes contribuidores deste trabalho. Certamente poderiam ser considerados co-autores desse trabalho. Ao Sérgio, que foi praticamente um co-orientador, sempre auxiliando com muitos bons conselhos. À Edhelmira, Tácio e André pela força e amizade. Ao Guilio, Gustavo e Breno que mesmo sendo de outros projetos sempre estiveram dispostos à me ajudar.

Agradeço também à equipe da Prill Tecnologia, na pessoa do Eduardo Prillwitz, por me apoiar e me liberar sempre que precisasse estar na UFF. Também aos colegas de trabalho, que me apoiaram e me ajudaram a superar muitos desafios.

Por fim, agradeço à CAPES, pelo incentivo à pesquisa através da bolsa de Mestrado, e ao programa de Pós-Graduação do IC/UFF pelo auxílio concedido para a apresentação do meu trabalho no SBRC em Brasília.

Resumo

O processo de desenvolvimento de aplicações pervasivas traz consigo diversos desafios, como a dificuldade de integração de sensores e atuadores e a criação e manipulação de cenários de teste para validar as aplicações. Tendo em vista estes desafios, foi concebida a Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP), que tem como objetivo fornecer a seus usuários uma plataforma de suporte à construção, teste e execução de aplicações para ambientes inteligentes (smart environments).

Para prover essas funcionalidades, a ferramenta proposta facilita a simulação de sensores e atuadores bem como meios para visualizar a interação da aplicação com componentes reais presentes no ambiente. Os dispositivos do ambiente inteligente são representados em um mapa esquemático, onde o desenvolvedor pode executar operações que modificam o seu estado e observar seu comportamento mediante a estas alterações. Isso possibilita a construção de cenários de teste personalizados. Assim, o desenvolvedor pode construir suas aplicações sem a necessidade de possuir a infraestrutura completa de um ambiente inteligente, assim como comprar sensores e outros dispositivos.

Adicionalmente, os usuários finais (e.g., um morador da residência) podem utilizar a IPGAP para gerenciamento de seus ambientes inteligentes, podendo controlar os dispositivos remotamente. Além disso, pode-se compor regras de contexto em alto nível, que executam no ambiente autonomomante. Isso pode ser feito selecionando graficamente as variáveis de contexto de dispositivos, conectando-as através de conectivos lógicos (e.g., e, ou), formando expressões lógicas. Então, as regras de contexto são implantadas no ambiente e iniciam a avaliação do contexto e a execução das ações.

Palavras-chave: Computação Ubíqua, Computação Pervasiva, Ambientes Inteligentes, Simulador, Prototipagem, Aplicações Pervasivas.

Abstract

The development process of pervasive applications brings many challenges, such as the difficulty in achieve sensors and actuators integration and the creation and manipulation of test scenarios in order to validate these applications. Considering these challenges we designed the Pervasive Applications Prototyping and Management Interface (IPGAP), which aims at providing to its users a platform to support the construction, testing and execution of applications for smart environments.

In order to provide these features, our tool facilitates the simulation of sensors and actuators as well as ways to visualize the interaction of the application and real components within the environment. The smart environment devices are represented in a schematic map where the developer can observe their behavior and perform operations that modify their state. This enables the construction of customized test scenarios. Thus, the developers can prototype their applications without the need of a complete smart environment infrastructure as well as the need of buying sensors and other devices.

Additionally, the end-users (e.g., residents of the smart home) can use the IPGAP for managing their smart environments and for controlling their devices remotely. Besides, they are able to compose high-level context rules which run in the environment autonomously. It can be done by graphically selecting the context variables from devices and connecting them through logical connectives (e.g., and, or), creating logical expressions. Next the context rules are deployed in the environment, and start to evaluate the context and to perform actions.

Keywords: Ubiquitous Computing, Pervasive Computing, Ambient Intelligence, Simulator, Prototyping, Pervasive Applications.

Lista de Listagens

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	:	Application Programming Interface;
AR	:	Agente de Recurso;
GUI	:	Graphic User Interface;
GPS	:	Global Positioning System;
IHC	:	Interação Humano-Computador;
IC	:	Interpretador de Contexto;
IPGAP	:	Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas;
P2P	:	Peer-to-Peer;
RFID	:	Radio-Frequency Identification;
SDK	:	Software Development Kit;
SDR	:	Serviço de Descoberta de Recursos;
SLR	:	Serviço de Localização de Recursos;
SRR	:	Serviço de Registro de Recursos;
Ubicomp	:	Ubiquitous Computing;
VC	:	Variável de Contexto;
XML	:	Extensible Markup Language;

Sumário

Capítulo 1

Introdução

Neste capítulo discutiremos os principais tópicos dos temas que permeiam este trabalho, além da motivação e os objetivos e contribuições do projeto. Apresentaremos na Seção 1.1 o paradigma da Computação Ubíqua/Pervasiva e descreveremos a Computação Sensível ao contexto e sua inserção no cenário atual da computação, incluindo um cenário de utilização dessas ideias. Na Seção 1.2 serão explicitadas as motivações deste trabalho e os desafios a serem resolvidos, bem como a proposta e os objetivos do trabalho. Por fim, na Seção 1.3 a organização desta dissertação será apresentada.

1.1 Contextualização

1.1.1 Computação Ubíqua, Pervasiva e Sensível ao Contexto

A história da computação é marcada por diversas mudanças no campo da Interação Humano-Computador (IHC). Os modelos de interação de maior visibilidade nos últimos 60 anos são descritos em [90] como as *ondas da computação*, denominadas *a era do mainframe* e *a era do computador pessoal* (do inglês, PC – *personal computer*).

A era do *mainframe* nos remete a um cenário onde um único computador é compartilhado por diversos usuários, e o acesso a essas máquinas era limitado, na maioria das vezes, a especialistas. Nessa ocasião era comum que os recursos computacionais fossem centralizados e acessados através de terminais. Assim, essa onda da computação diz respeito ao uso compartilhado de um recurso computacional escasso entre diversos usuários, seja por conta de seu alto valor comercial ou mesmo pela dificuldade técnica em utilizá-lo.

A segunda onda da computação se encaixa em um contexto onde o computador se

torna um item de utilização individual em sua essência, armazenando informações privativas de cada usuário e utilizando recursos próprios, como memória e processamento – o computador se tornou um item pessoal. Logo o número de utilizadores do PC ultrapassou o de computadores compartilhados, principalmente devido à popularização de sistemas com interfaces gráficas do usuário (do inglês GUI – *Graphic User Interface*), como Windows e Machintosh entre as décadas de 1980 e 1990. Em [90] o PC é descrito como um dispositivo que requer a atenção total do usuário enquanto uma tarefa está sendo realizada.

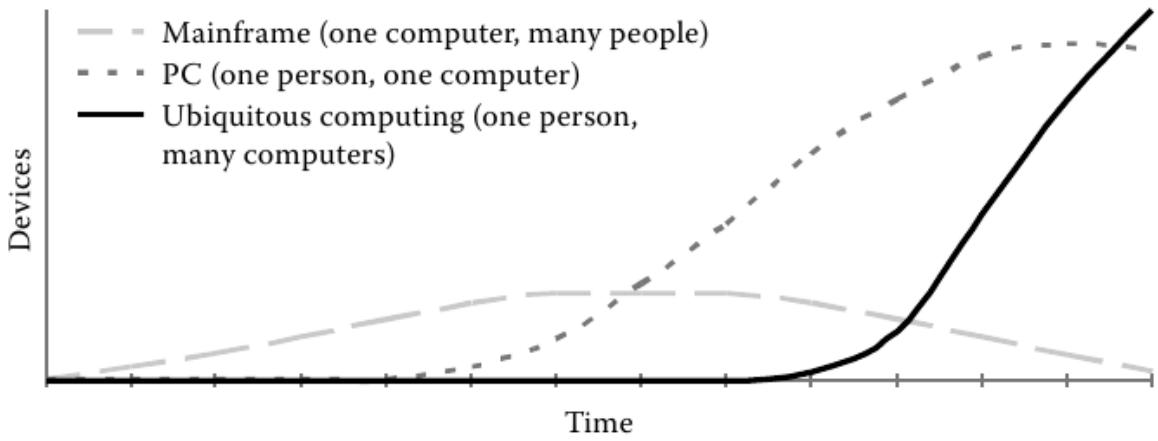


Figura 1.1: As três ondas da Computação

Em suas pesquisas ao longo da década de 1990 [87–89], Mark Weiser[1] descreveu uma série de princípios que culminariam em grandes mudanças nos modelos de interação existentes, prevendo o início de uma terceira onda da computação: a *Computação Ubíqua* (*Ubiquitous Computing* – Ubicomp), também conhecida como *Computação Pervasiva* [4][2] (*Pervasive Computing*). Se nos modelos anteriores haviam aproximadamente um computador para muitos usuários e um computador para um usuário, nesta nova onda haveria diversos computadores compartilhando um único usuário. Na clássica Figura 1.1 [85] podemos ver a evolução das eras da computação descritas por Weiser.

Entretanto, a alteração da cardinalidade no relacionamento entre computadores e pessoas não é a única, tampouco a mais marcante característica da Computação Ubíqua. Um mesmo usuário pode possuir diversos computadores pessoais: um *desktop* na sala de sua casa, outro no trabalho, vários computadores portáteis como *notebooks*, *netbooks*, *palmtops*, *tablets* e *smartphones*, sem que isso implique necessariamente em estar utilizando

[1] `http://www.ubiq.com/weiser/`

[2] Os termos *Computação Ubíqua* e *Computação Pervasiva* serão utilizados indistintamente ao longo deste trabalho.

Computação Ubíqua. Todos esses dispositivos utilizados da forma tradicional prendem a atenção do usuário de tal forma que não possam realizar outras tarefas.

A palavra *ubíquo* significa *onipresente*, sugerindo-nos que nesta era os computadores estão presentes em toda a parte – e é esta a ideia exata por traz deste conceito: dispositivos computacionais estarem presentes nos objetos do dia a dia, tais como utensílios domésticos, roupas, paredes, móveis, entre outros, tornando-se invisíveis no ambiente. Isto significa que a interação entre o usuário e estes dispositivos deve ser o mais transparente possível, de modo que se torne natural o suficiente para que não se tome consciência de se estar utilizando um computador.

Weiser afirma esse conceito ao iniciar seu artigo seminal sobre Computação Ubíqua [87] com a seguinte sentença:

> *"As tecnologias mais profundas e duradouras são aquelas que desaparecem. Elas dissipam-se nos artefatos do cotidiano até tornarem-se indistinguíveis."*
> (tradução nossa)

Previu-se então que o futuro da computação estaria na utilização de serviços providos por tecnologias que estivessem tão entranhadas no ambiente que o usuário pudesse utilizá-los inconscientemente. Ainda em [87], Weiser afirma que a Ubicomp tornaria o uso de dispositivos computacionais, ao contrário dos dias de hoje, menos frustrantes, e "tão refrescantes como um passeio no bosque". A denominação computação calma (*calm computing*) advém justamente desta possibilidade de interação com os dispositivos computacionais de maneira intuitiva e natural. Em [90] a Computação Ubíqua é comparada à eletricidade, pois não se tem a consciência de que está sendo utilizada para realizar as diversas tarefas do dia a dia.

Essas ideias são opostas ao paradigma do *desktop*, que tenta inserir o mundo real dentro do computador, com representações dos objetos do cotidiano em forma de ícones e imagens. A Ubicomp propõe o paradigma inverso: inserir o computador no mundo real [89]. Por exemplo, em vez de se ler um jornal na tela de um PC, em uma interface gráfica que imita o documento impresso, no contexto da Ubicomp a leitura seria realizada em um dispositivo dedicado, feito de um material maleável como o papel [23, 74] com um processador embutido. Dessa forma o usuário leria o documento justamente como se estivesse lendo um livro impresso comum, porém com funcionalidades como notícias atualizadas através da internet, exibição das notícias de sua preferência, entre outras [85].

Poucos anos mais tarde, as ideias da Computação Ubíqua/Pervasiva evoluíram a um

novo patamar. Anind Dey[3] & Gregory Abowd[4] propuseram que os sistemas tivessem a capacidade de responder à estímulos provocados por alterações no *contexto* do ambiente. Uma definição de contexto foi apresentada por eles em [2], como a seguir:

> [Contexto é] *"qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar, ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo eles próprios."*
> (tradução nossa)

Essa definição tornou mais clara a ideia de transparência de interação, pois através do conhecimento do contexto, aplicações são capazes de prover serviços coerentes com estado atual do ambiente sem que o usuário tenha que explicitamente fornecer uma entrada para o sistema [13]. Por exemplo, em uma conferência os dados de contexto dos participantes podem ser utilizados para sugerir automaticamente trilhas e apresentações, baseado em seus tópicos de interesse [29]. Em uma residência, a temperatura do cômodo pode ser ajustada baseado na sensação de frio ou calor dos usuários, obtida através de sensores corporais. Esses conceitos deram origem à **Computação Sensível ao Contexto** (do inglês, *Context Aware Computing*[5]) e as aplicações que se utilizam dela são frequentemente chamadas de *aplicações sensíveis ao contexto*. Outros importantes trabalhos sobre computação sensível ao contexto podem ser encontrados em [1, 3, 30].

1.1.2 Conjuntura Atual

Nas últimas décadas, tem-se observado um grande avanço nas tecnologias da área da computação como um todo, o que já implica em mudanças no modo de interação com os dispositivos computacionais. Com o advento da internet, informações passaram a ser mais facilmente distribuídas entre diversos dispositivos interconectados, o que tem angariado mais força nos últimos anos com a popularização da computação em nuvens (do inglês, *cloud computing*) [81].

Dentre outros serviços, a computação em nuvens permite a execução de aplicações em plataformas de *hardware* e/ou *software* elásticas disponibilizadas através da internet [6], além do compartilhamento de arquivos entre qualquer dispositivo computacional

[3] `http://www.cs.cmu.edu/~anind/`
[4] `http://www.gregoryabowd.com/`
[5] Alguns autores traduzem o conceito como Computação Ciente do Contexto

do mundo, independentemente de seu Sistema Operacional. Atualmente estão disponíveis diversos serviços que atuam no segmento de computação em nuvens como Windows Azure[6], Amazon Web Services[7], Dropbox[8], Google App Engine[9] e Apple iCloud[10].

Os avanços tecnológicos das redes de comunicação e a popularização dos serviços providos através da internet se tornaram um dos principais responsáveis pela recente explosão de dispositivos móveis no mercado. *Smartphones* e *tablets* estão presentes nas casas de uma grande parcela da população, equipados com dispositivos como GPS (*Global Positioning System*), acelerômetro, câmeras digitais e alguns sensores, que proporcionam aos seus usuários diversos serviços. Além disso, sensores diminutos podem ser encontrados a preços razoavelmente baixos mais facilmente, e a produção de eletrodomésticos equipados com um processador e acesso a internet já é uma realidade, tornando cada vez mais factível a criação de aplicações ubíquas.

Ademais, aplicações para plataformas móveis tem caminhado cada vez mais em direção à Ubicomp, com milhões de aplicações desenvolvidas e distribuídas nas lojas virtuais nos últimos anos, cada vez mais inseridas no cotidiano dos usuários. Esse sucesso se deve, sobretudo, ao surgimento de sistemas operacionais mais adequados para os dispositivos móveis, como Google Android, Apple iOS e Microsoft Windows Phone. Incluem-se neste número diversas aplicações interessantes como, por exemplo, um aplicativo para identificação de estresse no usuário através da captação de sua voz pelo microfone do aparelho [54], e um aplicativo que adquire a frequência cardíaca através do LED da câmera de um *smartphone* [43].

Além disso, outros conceitos relacionados à Ubicomp têm surgido, estimulados pelos avanços já mencionados. Um deles é o conceito de **Ambientes Inteligentes** proposto em [8], que descreve um ambiente repleto de sensores e atuadores provendo um sistema sensível ao contexto, visando minimizar a interação explícita dos usuários com os dispositivos computacionais. Um outro exemplo é o paradigma da **Internet das Coisas** (do inglês, *Internet of Things*) cujo principal objetivo é a identificação dos dispositivos (as "coisas") do dia a dia utilizando-se de um esquema de endereçamento único, de modo a facilitar seu acesso e interação de qualquer parte do mundo. Para este fim, tecnologias como RFIDs (*Radio-Frequency Identification*) e o IPv6 são frequentemente discutidas como soluções a serem utilizadas [7, 79].

[6] `http://www.windowsazure.com`
[7] `http://aws.amazon.com/`
[8] `http://www.dropbox.com`
[9] `http://appengine.google.com/`
[10] `https://www.icloud.com/`

1.2 Definição do Problema

1.2.1 Motivação: Ambientes Inteligentes

Considere um ambiente onde sensores e atuadores, praticamente imperceptíveis, interconectados através de uma rede de comunicação estão espalhados por suas dependências, tendo a capacidade de adquirir e alterar seu contexto. Utensílios do cotidiano como eletrodomésticos, móveis, portas, janelas, torneiras, lâmpadas, roupas, objetos de escritórios, entre diversos outros, contém pequenos processadores e também estão conectados à rede, tornando o sistema *ciente* de todas as informações do ambiente. Este cenário nos estimula a pensar em diversos tipos de aplicações. Abaixo uma listagem com algumas delas:

- **Saúde:** Sensores capturam dados fisiológicos do paciente (e.g. pressão arterial e frequência cardíaca), e sua atividade é inferida por acelerômetros. Um módulo inteligente infere o estado de saúde do paciente através de técnicas de lógica *fuzzy* e pode emitir alertas acionando automaticamente uma ambulância, ou contactando um profissional de saúde cadastrado [21, 56];

- **Segurança:** Através de dados de contexto coletados de sensores e preferências do usuário, a aplicação tranca portas e janelas da casa automaticamente quando identifica por exemplo, que todos os moradores já foram dormir. Quando detectada a presença de pessoas não autorizadas no entorno da residência certa hora da noite, gera um alarme ou contacta as autoridades competentes. Ao detectar sinais de incêndio, chama o serviço do corpo de bombeiros já informando a localização da residência;

- **Economia de recursos:** O consumo de energia no ambiente é constantemente monitorado, e quando um dispositivo qualquer atinge um limite previamente especificado, é desligado ou um alarme é gerado. A utilização de aparelhos que consomem grande quantidade de energia é automaticamente escalonada para horários de baixa tarifação, quando possível. Os dispositivos que não estão sendo utilizados são desligados ou postos em modo de hibernação. Vazamentos na tubulação de água são detectados, ocasionando a geração de alarmes. Torneiras esquecidas abertas são automaticamente fechadas.

- **Conforto:** A cafeteira e a torradeira são ligadas assim que o usuário acorda. A temperatura do ambiente é ajustada automaticamente, de acordo com a sensação e

preferências dos usuários. A TV sugere a programação de acordo com o interesse do telespectador.

1.2.2 Problema

É notável que aplicações e dispositivos inovadores vêm tomando espaço na mídia e no mercado. Todos os anos feiras e exibições tecnológicas, como o CES[11] (*Consumer Electronics Show*) são realizadas em todo mundo mostrando as novas tendências. Claramente muitas das aplicações e dispositivos destacados nas feiras tem um caráter que nos remete à Ubicomp.

Entretanto, de modo geral essas novas aplicações e tecnologias são auto-contidas, ou seja, não compartilham as informações geradas, nem expõem seus serviços no ambiente a fim de cooperar com outros aplicativos e provisionar serviços diferenciados para o usuário. O grande desafio da computação Ubíqua/Pervasiva é exatamente utilizar essas aplicações integradas a um ambiente inteligente, fornecendo seus serviços e informações a outras entidades.

Podemos citar como causas desse efeito a dificuldade em integrar os dispositivos que fazem parte do ambiente inteligente [34], a falta de ferramentas adequadas para a criação e integração dessas aplicações, e a dificuldade em depurá-las [86]. Além disso, um ambiente de testes contendo todos os dispositivos e a infraestrutura necessária para realizá-los pode ser inviável financeiramente, ao passo que um ambiente construído em pequena escala pode não ser suficiente para testar os diversos cenários possíveis em um ambiente inteligente [26]. Por fim, as dificuldades de desenvolvimento mencionadas podem refletir para o usuário final, resultando em produtos de alto custo, inflexíveis e difíceis de gerenciar [19].

1.2.3 Proposta de Solução e Objetivos

Para resolver esses problemas, é proposta uma ferramenta de suporte à construção de protótipos de aplicações pervasivas chamada IPGAP (Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas), que através de técnicas e Prototipagem Rápida (*Rapid Prototyping*) fornece ao desenvolvedor um ambiente de testes para suas aplicações de forma a facilitar o desenvolvimento [1] e diminuir o custo do projeto. O desenvolvimento rápido de um protótipo permite que os desenvolvedores tenham um artefato funcional nas fases iniciais do desenvolvimento o que consequentemente permite a correção de erros o

[11] `http://www.cesweb.org/`

mais cedo possível [44].

Utilizando-se de simulação na prototipagem de aplicações pervasivas é possível que os testes sejam realizados sem a necessidade de ter uma infraestrutura real completa. Dessa forma, permite-se que, por exemplo, seja testada a interação de sensores que não existem no mercado ou protocolos e algoritmos novos facilmente, testando a eficiência ou viabilidade de novas tecnologias [72].

Juntamente com um conjunto de serviços básicos para gerenciamento dos recursos do ambiente (como descoberta e registro), APIs para invocação remota de operações e comunicação por eventos, e um suporte para interpretação de contexto, a IPGAP auxilia o desenvolvedor na construção e evolução de suas aplicações mais facilmente.

1.3 Organização da Dissertação

Os capítulos a seguir estão estruturados da seguinte forma: No Capítulo 2, apresentaremos uma visão geral dos principais conceitos utilizados como base para o desenvolvimento da IPGAP. Descreveremos no Capítulo 3 o projeto SmartAndroid, sua motivação e as hipóteses que utiliza . No Capítulo 4, veremos mais detalhes sobre o funcionamento da ferramenta, seus conceitos, características e exemplos de utilização. Mostraremos na Seção 5 uma avaliação da IPGAP através de aplicações desenvolvidas. Os trabalhos relacionados serão apresentados e discutidos no Capítulo 6, e as conclusões e trabalhos futuros, no Capítulo 7.

Capítulo 2

Conceitos Envolvidos

Neste Capítulo serão descritos os principais conceitos nos quais esta proposta se baseia. Entre os assuntos abordados constam as abstrações de dados adotadas, os procedimentos de registro e descoberta de recursos e os aspectos de comunicação entre eles, bem como os mecanismos utilizados para adquirir o contexto do ambiente. Os tópicos serão discutidos em sua maioria de forma conceitual, com foco principal na arquitetura das propostas. Oportunamente, possíveis implementações serão discutidas ao longo do texto.

2.1 Agentes de Recurso

Definimos como *recurso* qualquer entidade de *hardware* ou *software* que exponha serviços (ou *operações*[1]) que possam ser utilizados por aplicações, ou outras entidades presentes no ambiente. Assim, podemos dizer que sensores, atuadores e dispositivos domésticos (e.g. fogão, geladeira, ar-condicionado) se inserem nessa definição. Por exemplo, um sensor de temperatura pode expor um serviço que obtém a temperatura corrente do cômodo onde se encontra. Um fogão pode expor, através de serviços, a informação de que seu forno está ou não ligado, quais bocas estão acesas, entre outras funcionalidades, para que entidades externas obtenham ou alterem essas informações. Um módulo de *software* que agrega dados de diversos dispositivos para expor informações de consumo de energia do ambiente também é um exemplo de recurso.

O acesso aos serviços providos pelos recursos pode variar muito, tornando a sua integração com outros componentes do sistema um processo custoso. Esse problema pode ser contornado através da uniformização da interface dos recursos por uma entidade que encapsule suas particularidades, ajudando a diminuir a complexidade de integração do sis-

[1]Nesta seção utilizaremos os temos *serviço* e *operação* de forma intercambiável.

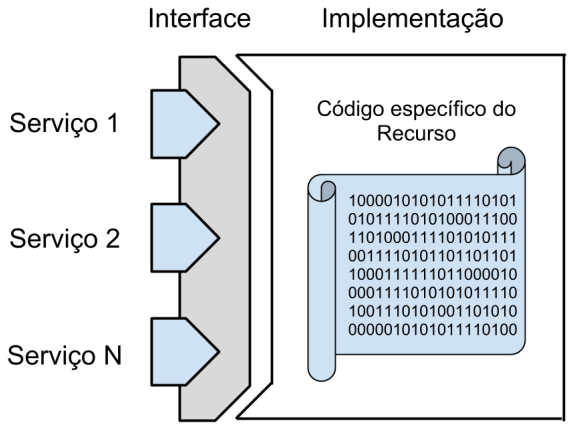


Figura 2.1: Um Agente de Recurso

tema. Nesse sentido, desenvolveu-se o conceito de **Agentes de Recurso (AR)** [20,73,76], que encapsulam os detalhes de implementação e comunicação dos recursos e expõem a interface de seus serviços de forma padronizada.

Por exemplo, o AR de um sensor de umidade será responsável por conhecer o protocolo específico utilizado para adquirir as informações do aparelho. Para que uma aplicação invoque as operações do sensor, este deve conhecer apenas a interface fornecida pelo AR correspondente, sem ter que necessariamente utilizar o protocolo original. Esse conceito é utilizado da mesma forma na Orientação à Objetos, onde é conhecido como *encapsulamento*. Na Figura 2.1 podemos ver a representação básica de um AR.

Os ARs fornecem operações básicas, necessárias para que o recurso funcione adequadamente no ambiente inteligente. A implementação genérica inclui primitivas de comunicação síncrona e assíncrona (Seção 2.3), técnicas de Programação Orientada a Aspectos (POA) para a notificação de mudanças no contexto do recruso (Seção 2.4), bem como as diretivas necessárias para utilizar os serviços de gerenciamento (Seção 2.2) para que o recurso seja visível e acessível no ambiente inteligente. O desenvolvedor deve, através de um estilo de programação bem definido, estender este AR básico para incluir o comportamento e as funcionalidades adequados ao seu recurso.

2.2 Serviços de Gerenciamento

Aplicações para Ubicomp lidam constantemente com os recursos do ambiente, adquirindo seu contexto, invocando seus serviços e gerenciando seu ciclo de vida. Para que se tenha acesso a esses recursos é de grande valia o uso de **Serviços de Gerenciamento** (SG) que permitam o registro de novos recursos e a sua descoberta no ambiente, além de gerir a sua

localização. O *framework* conceitual proposto em [57] prevê a utilização de componentes que implementam o **Serviço de Registro**, **Serviço de Descoberta** e o **Serviço de Localização**, utilizando a abstração de ARs para gerenciar os recursos do ambiente.

2.2.1 Serviço de Registro

Este serviço é responsável por adicionar/remover referências de ARs em um *repositório de recursos*. Neste repositório se encontram todos os recursos ativos no sistema, ou seja, os recursos que estão *visíveis* no ambiente e podem fornecer seus serviços para outras entidades e aplicações. Sempre que um serviço novo surgir no sistema, este deve invocar o **Serviço de Registro de Recursos (SRR)**, para que sua referência seja incluída no repositório de recursos e consequentemente fique ativo (visível) para o sistema como um todo.

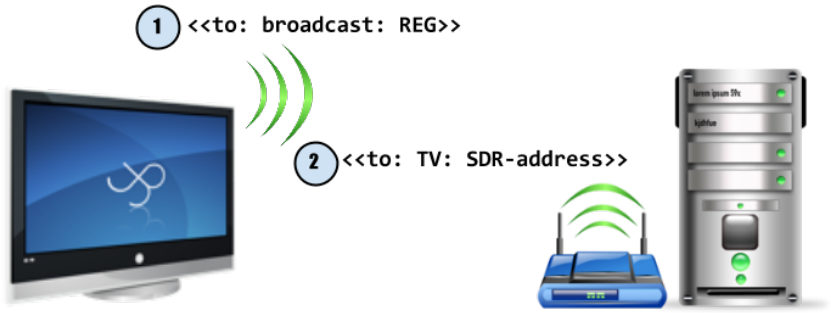


Figura 2.2: TV se registrando no Serviço de Registro de Recursos

Para efetuar o registro, o dispositivo deve enviar uma mensagem de *broadcast*, contendo suas informações básicas, além de informações de controle criptografadas informando que deseja se registrar no ambiente. O SRR ao receber a mensagem, verifica as informações do recurso e o registra no repositório. Na resposta à mensagem, é enviada a referência do Serviço de Descoberta (Seção 2.2.2) para que o solicitante do registro possa utilizá-la na interação com o restante do ambiente.

Na Figura 2.2 podemos ver esse procedimento sendo realizado por uma TV. Esta envia uma mensagem de *broadcast*, que é recebida por todos os dispositivos ao alcance do sinal. O SRR, presente em um servidor no ambiente também recebe a mensagem, e responde à requisição da TV. Os outros dispositivos que receberam a mensagem de *broadcast* descartam a mensagem recebida. É importante lembrar a importância da utilização de mensagens criptografadas, para que dados sensíveis do ambiente não sejam expostos de maneira inadequada (ver Seção 3.4).

O SRR é utilizado na IPGAP para registrar os recursos simulados criados diretamente pela ferramenta, de forma que estes fiquem visíveis também para outros dispositivos presentes no ambiente, mesmo que fora da IPGAP. Além disso, a IPGAP fornece uma interface administrativa que permite ao usuário remover qualquer recurso do repositório, também através de chamadas ao SRR.

2.2.2 Serviço de Descoberta

Para que um recurso possa ser acessado e tenha seus serviços invocados é necessário que sua referência seja conhecida. Sobretudo em um ambiente que contém diversos recursos que podem ser registrados e ter seu registro desfeito autonomamente no sistema, o uso de um serviço que os descubra é benéfico.

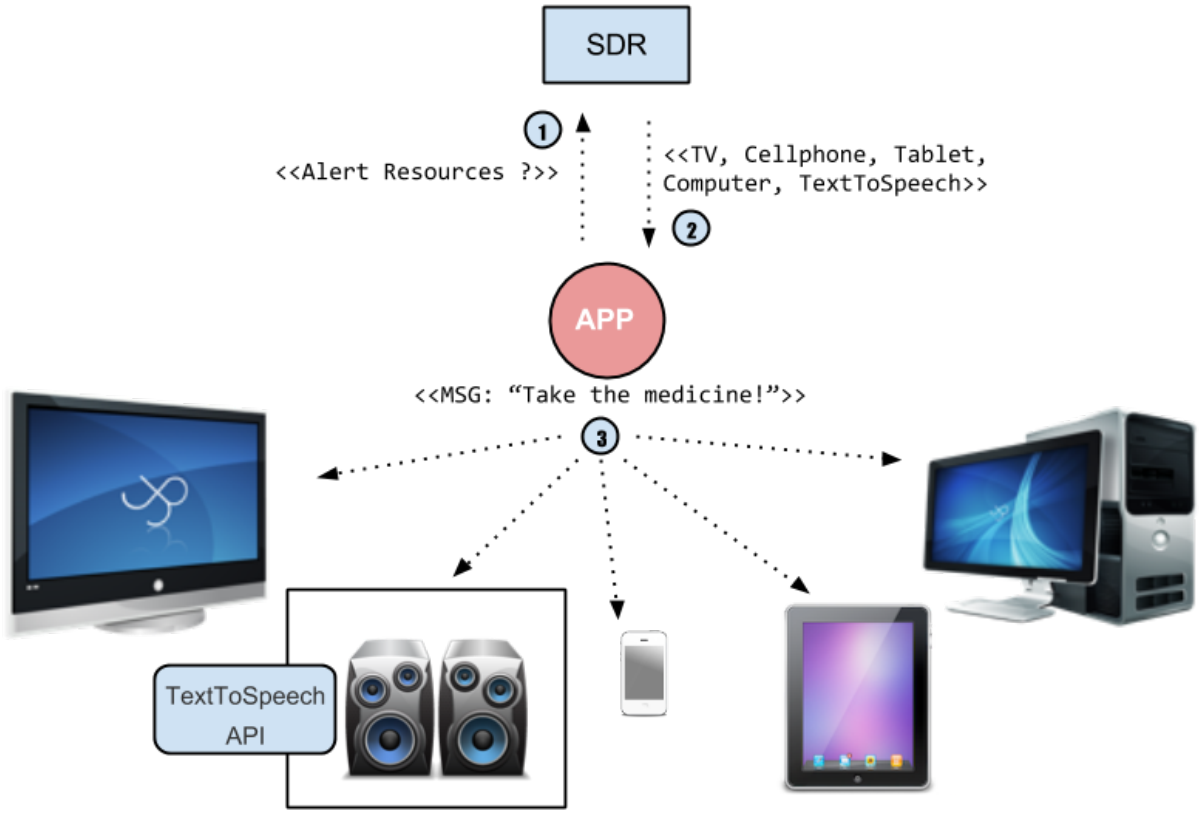


Figura 2.3: Aplicação consultando recursos através do SDR

Como exemplo, considere uma aplicação que em um dado fluxo necessite enviar um alerta para o usuário. Para esta aplicação não é importante qual o recurso em específico receberá o alerta, o importante é que seja transmitido e que o usuário o visualize. Uma solução possível seria esta aplicação manter uma lista com todos os dispositivos da casa que pudessem tratar essa mensagem, obrigando que a aplicação os conheça a priore.

Essa solução impõe uma responsabilidade desnecessária à aplicação: a de conhecer os dispositivos do ambiente que possam realizar o serviço desejado, ferindo o princípio de separação de interesses.

Para atender a estas e outras situações, foi concebido o **Serviço de Descoberta de Recursos (SDR)**, através do qual pode-se obter a referência dos recursos desejados, através de consultas. Na Figura 2.3 vemos a interação da aplicação acima descrita com o SDR e os recursos do ambiente. No passo **1** é realizada a consulta ao SDR, solicitando recursos que possam emitir alertas. O SDR processa a requisição e busca no repositório de recursos os ARs que condizem com o que foi pedido. Alguns recursos que poderiam ser selecionados nesta busca são: TV, Computador, Tablet, Smartphone, ou um módulo que utilize a API **TextToSpeech**[2] para transmitir a mensagem em forma de som. No passo **2**, o SDR responde com as referências – representadas pelos ARs dos recursos – que atendem a consulta. Finalmente no passo **3**, a aplicação já possui os ARs dos recursos disponíveis, e pode invocar o serviço de mensagem de alerta.

A descoberta pode ser realizada através de vários tipos de consulta, que retornam como resultado referências para os ARs que satisfazem os critérios da busca. Algumas consultas envolvem o tipo dos recursos, que são caracterizados através da definição de uma ontologia mínima (veja [14] para um detalhamento). Na Figura 2.4 vemos a assinatura dos principais métodos de busca do SDR, e na Listagem 2.1 um exemplo de uso de cada uma das consultas.

A IPGAP utiliza a API do SDR para adquirir a referência dos recursos que o usuário deseja. É a apresentada uma lista com os recursos que atendem a busca do usuário, que podem ser adicionados ao mapa do ambiente, representado na IPGAP.

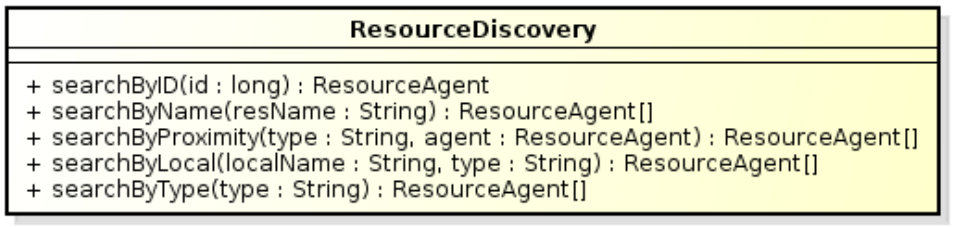


Figura 2.4: Métodos de Busca do SDR

[2] http://developer.android.com/reference/android/speech/tts/TextToSpeech.html

Listagem 2.1: Consultas do SDR

```
1 //Todos os recursos do tipo Lampada registrados no sistema
2 ResourceAgent[] lamps = discovery.searchByType("Lamp");
3 //Busca o recurso pela sua ID
4 ResourceAgent ag = discovery.searchByID(14234419L);
5 //Todos os recursos que contenham "sensor" em seu nome
6 ResourceAgent[] res = discovery.searchByName("sensor");
```

2.2.3 Serviço de Localização

Os aspectos relacionados à localização dos recursos são de responsabilidade do **Serviço de Localização de Recursos (SLR)**. Este serviço realiza buscas em uma base na qual os ARs estão indexados pelo *lugar* onde se encontram. A lista de lugares disponíveis é conhecida pelo SLR no momento em que é carregada a estrutura de dados que representa o mapa do ambiente, contendo a posição de cada cômodo. É realizado um cruzamento dos dados de posicionamento dos recursos e dos cômodos, relacionando assim cada recurso com os lugares onde estão. A estrutura de dados que contém o mapa do ambiente é configurável e pode ser carregada diretamente pela IPGAP.

Listagem 2.2: Consultas de Localização do SLR

```
1 //Os recursos do tipo "Smartphone" mais próximos do recurso 'ag'
2 ResourceAgent[] tvsNear = localization.searchByProximity("Smartphone", ag);
3 //Os recursos do tipo "PresenceSensor" presentes no ambiente "Sala"
4 ResourceAgent[] presenceSensors =
5   localization.searchByLocal("Sala", "PresenceSensor");
6 //Sobrecarga do método acima, utilizando um objeto da classe 'Place'
7 //referenciado por 'myPlace', para representar o lugar
8 ResourceAgent[] presenceSensors =
9   localization.searchByLocal(myPlace, "PresenceSensor");
10 //O local onde se encontra o recurso 'ag'
11 Place place = localization.getPlaceOf(ag);
12 //Retorna a lista de lugares (cômodos) disponíveis no ambiente
13 Place[] places = localization.getPlaces();
```

Através desse conhecimento é possível buscar um recurso presente em um cômodo específico, ou mesmo encontrar o recurso mais próximo de uma dada posição. Ao combinar as buscas de localização com as buscas por tipo do SDR, podemos realizar buscas como: *O smartphone que está mais próximo da sala.* A Listagem 2.2 mostra códigos de exemplo para o uso das consultas do SLR.

2.3 Paradigmas de Comunicação

2.3.1 Comunicação Síncrona

Conforme já discutido, em uma aplicação pervasiva frequentemente é necessário que se obtenha o contexto de um recurso, ou enviar comandos para ele (e.g. liga/desliga). Esse procedimento geralmente é realizado de maneira síncrona, ou seja, enviando mensagens e aguardando suas respostas (protocolo *request/reply*). Na maioria dos *frameworks* atuais, isso se dá em mais alto nível, através abstrações de objetos distribuídos / chamadas remotas como às utilizadas no RPC (*Remote Procedure Call*) [15], RMI (*Remote Method Invocation*) [68], CORBA [65], Serviços Web (*Web Services*) [84], entre outros.

Para prover essa funcionalidade, as interfaces dos recursos devem estar bem definidas, e logicamente deve ser conhecidas pelos chamadores. Além disso, existirão duas implementações para cada interface: a implementação que de fato executa as regras de negócio envolvidas, e uma implementação que apenas encaminha as chamadas para a entidade adequada, agindo como um *proxy*. A implementação deste *proxy* quase sempre é gerada automaticamente a partir da interface em questão. Dessa forma, a chamada remota se torna transparente para o solicitante, que se comporta como se estivesse realizando uma chamada local.

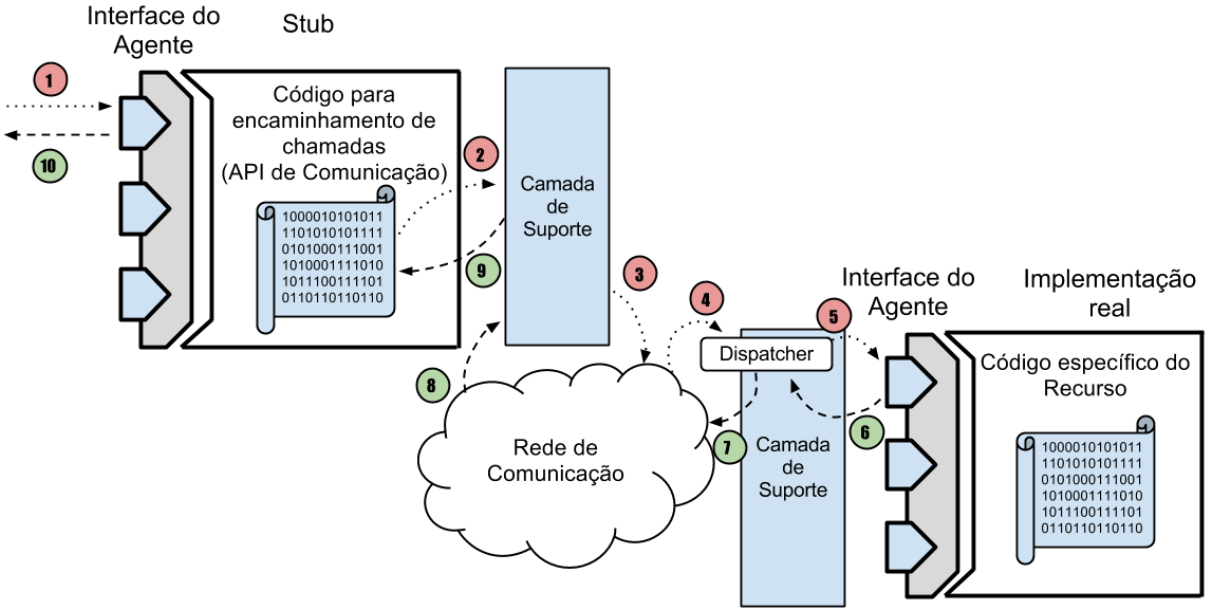


Figura 2.5: Chamada Remota

A entidade que faz o encaminhamento acima descrito, conhecida como *stub*, serializa cada chamada juntamente com seus parâmetros (*marshalling*) em um formato como

XML[3], JSON[4] ou um formato arbitrário. A mensagem serializada é repassada para uma camada de suporte para que seja enviada através da rede de comunicação. Ao chegar ao *host* de destino, a mensagem é desserializada pelo *middleware* de suporte, e a operação do AR adequada é chamada, passando-se os parâmetros presentes na mensagem. Após a execução da operação, o *middleare* de suporte desserializa o valor de retorno da chamada (*unmarshalling*), e a envia de volta ao *stub*, que por fim devolve o retorno ao chamador.

Na Figura 2.5 podemos ver uma chamada remota feita através de um *stub*, e encaminhada até o AR que implementa o serviço requerido. No passo **1**, um dos serviços presentes na interface do AR é invocado. Já no passo **2**, o código do *stub* faz o *marshalling* dos dados necessários, e no passo **3**, os envia através da rede. A rede de comunicação entrega o pacote à camada de suporte do *host* de destino no passo **4**, onde é processado por um componente chamado ***Dispatcher***. No passo **5**, o Dispatcher seleciona qual serviço foi invocado pelo cliente, e realiza uma chamada local para o AR requerido, invocando o serviço selecionado. O AR processa a chamada, e devolve a resposta para a camada de suporte no passo **6**. O Dispatcher faz o *marshalling* do retorno recém recebido, e envia o pacote de volta à rede de comunicação no passo **7**. No passo **8**, a rede entrega o pacote para a camada de suporte do *host* do emissor, que realiza o *unmarshalling* dos dados de retorno e no passo **9**, devolve o retorno ao stub. Finalmente, no passo **10**, os dados são retornados ao chamador.

A IPGAP utiliza largamente este paradigma de comunicação, para manipular os recursos do ambiente, como acender ou apagar uma lâmpada ou mudar o canal da TV.

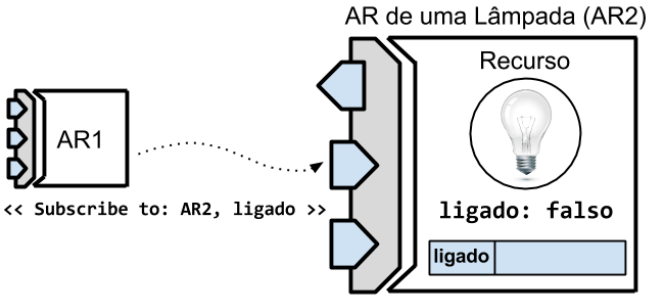
2.3.2 Comunicação Assíncrona

Na Seção anterior falamos sobre a comunicação síncrona, onde a entidade requisitante espera pela resposta da sua chamada. Embora seja uma técnica útil em muitos casos, nem sempre esse comportamento é adequado pois a entidade permanece bloqueada até que receba a resposta da sua requisição, ou que ocorra um *timeout*. Em situações em que não há garantias de que a resposta será enviada de imediato, caracteriza-se o uso de comunicação assíncrona (comunicação por eventos), onde o requisitante faz a chamada, e continua sua execução.

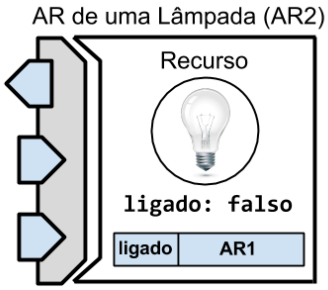
[3] `http://www.w3.org/standards/xml/core`
[4] `http://www.json.org/`

2.3.2.1 *Publish-Subscribe*

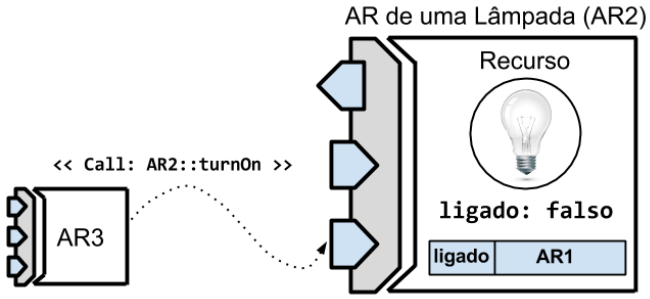
A comunicação assíncrona em um sistema distribuído é frequentemente modelada pelo padrão publica-subscreve (*publish-subsribe*) [38], onde temos dois tipos de entidades: as que publicam eventos, e as que são interessadas em receber esse evento, para tratá-lo da forma adequada. Os interessados devem se subscrever ao publicador de eventos, informando-o que a partir de agora desejam receber mensagens (eventos) de um determinado tipo. Assim que um evento ocorrer, o publicador notificará cada interessado do ocorrido. Os interessados devem estar preparados para receber a mensagem do publicador em uma *thread* separada – implementando um método de *callback* – pois não se tem garantias de quando o evento irá ocorrer.



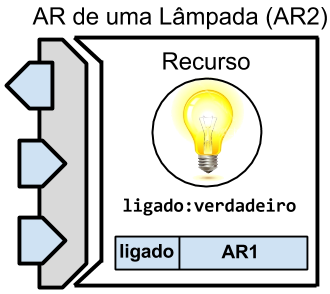
(a) Subscrição do AR1 ao AR2



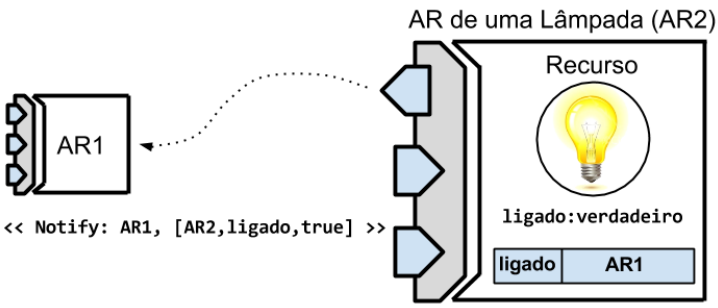
(b) AR2 armazena subscrição do AR1



(c) AR3 invoca operação do AR2



(d) Estado do AR2 modificado



(e) Notificação do AR1

Figura 2.6: Processo de subscrição e notificação de ARs

Listagem 2.3: Tratamento de Eventos

```
1 @Override
2 public void notificationHandler(ResourceAgent entity,
3   String event, Object value) {
4   // Do something...
5 }
```

Na Figura 4.7, podemos ver o processo de subscrição/notificação de eventos utilizada pelo padrão publica-subscreve. Na Figura 2.6(a), vemos um AR qualquer (AR1) se subscrevendo à um AR que representa uma lâmpada (AR2), que está apagada. Na invocação do serviço *Subscribe* do AR2, o AR1 informa que está interessado em receber notificações de modificação da variável *ligado*. O AR2, então, relaciona a referência do AR1 com a variável *ligado* em uma estrutura de dados interna (Figura 2.6(b)).

Em um momento posterior no tempo, um terceiro AR (AR3) invoca o serviço *turnOn* do AR2 (Figura 2.6(c)), onde a variável *ligado* é modificada para o valor *verdadeiro* e a lâmpada é acendida (Figura 2.6(d)). A chamada é interceptada através de técnicas de Programação Orientada a Aspectos [35], onde nesse momento os interessados na variável *ligado* (que foi modificada) são notificados (Figura 2.6(e)).

A mensagem de notificação de modificações deve conter as identificações do evento gerado e da entidade que o gerou, bem como um parâmetro contendo a alteração ocasionada no estado dessa entidade. Um exemplo de implementação do tratamento de eventos pode ser visto na Listagem 2.3. Neste exemplo, o primeiro parâmetro é um objeto da classe **ResourceAgent**, que modela um AR, onde **entity** representa a entidade que gerou o evento. No segundo parâmetro temos uma **String**, que representa o evento gerado, e por último, um objeto genérico que representa o valor alterado.

Utilizando o código da Listagem 2.3 no exemplo dado na Figura 4.7, podemos perceber que a mensagem de notificação conterá como primeiro parâmetro uma referência para o AR2, no segundo parâmetro uma **String** com o valor *"ligado"* (variável modificada no evento), e no terceiro parâmetro, um objeto **Boolean** com o valor *true* (valor para qual a variável *ligado* foi modificada).

A utilização deste estilo de comunicação no sistema, se dá sobretudo na notificação de mudança no estado dos ARs, onde todos os interessados em um estado específico são notificados quando este é alterado. Semelhantemente, no suporte à Regras de Contexto (Seção 2.4) a comunicação por eventos é utilizada na notificação de que um regra foi satisfeita.

2.3.2.2 Rede Endereçada por Interesses

O modelo de comunicação via pilha de protocolos TCP/IP utiliza o princípio de *onde* está o serviço que se quer acessar, em outras palavras, *qual o endereço da máquina que provê o serviço*. Vimos na Seção 2.3.2 a utilização do padrão *publish-subscribe* para notificar as entidades interessadas em mudanças no estado de um determinado AR. Neste caso, a utilização de comunicação via TCP/IP está adequada, pois a entidade que fornece o serviço é conhecida, a saber, o AR no qual estamos interessados nas mudanças de estado.

Entretanto, a implementação de um interesse como por exemplo *a mudança de estado de todos os sensores de presença do ambiente* via TCP/IP não seria tão trivial, pois não se conhece a priori o endereço da instância do AR que fornece o serviço, necessitando que uma entidade centralizada se encarregue do encaminhamento do interesse. A situação ideal seria a utilização de uma rede par-a-par (do inglês, *peer-to-peer* – P2P) através da qual fosse possível publicar o interesse na rede, e as entidades que possuíssem interesses correspondentes responderiam.

Dessa forma, o protocolo de comunicação ad hoc REPI (Rede Endereçada Por Interesses) [28,33] encaminha as mensagens de acordo com termos que representam os interesses das entidades, sem a necessidade de utilização do endereçamento origem-destino. Ademais, a criação da rede é totalmente colaborativa e o conhecimento da estrutura da rede não é necessário, tornando o algoritmo totalmente distribuído. Em [24] é proposta uma implementação do protocolo REPI voltada para aplicações pervasivas.

Na implementação realizada, pode-se configurar qual a abordagem utilizar para a comunicação assíncrona: REPI ou *Publish-Subscribe*, através de uma interface administrativa na IPGAP.

2.4 Interpretação de Contexto

Conforme apresentado no Capítulo 1 de maneira introdutória, aplicações sensíveis ao contexto são aplicações que reagem a estímulos provocados pelo ambiente físico, obtidos através de sensores. Essa é uma caracterização fundamental para tornar os sistemas ubíquos/pervasivos mais transparentes e menos intrusivos. Por exemplo, em uma ambiente inteligente o contexto poderia ser: a temperatura de um ar-condicionado, quantos indivíduos se encontram na cozinha, quem está assistindo à TV da sala, a umidade detectada no quarto, entre outros.

Aplicações pervasivas podem utilizar expressões lógicas, conhecidas como **regras de contexto** (RC), para implementar a sensibilidade ao contexto em um ambiente inteligente. As RCs são compostas por condições obtidas a partir do estado de ARs e, opcionalmente, de um temporizador. Dessa forma, a análise de uma RC é realizada constantemente por um **Interpretador de Contexto** (IC), que caso a avalie como verdadeira, notifica as entidades interessadas esse fato.

A criação de uma RC inicia-se na seleção das **variáveis de contexto** (VC) que participarão da regra. As VCs correspondem às informações de contexto de um AR, ou seja, seu estado (e.g. o canal em que se encontra uma TV). A partir das VCs são construídas condições, utilizando-se operadores de comparação como $<$, $>$, $\leq$, $\geq$, $=$, $\neq$. Assim, para interligar as condições, são utilizados conectivos lógicos como **e**, **ou** e **não**.

No momento em que uma RC é composta, é criada uma instância de um IC para individualmente avaliá-la e lidar com a notificação de seus interessados. A utilização da relação de um-para-um entre RCs e ICs deve-se à organização distribuída dos nós que é possibilitada a partir dessa abordagem. Nesse sentido o processamento das regras torna-se mais eficiente, além de proporcionar a diminuição dos pontos de falha do sistema.

Além disso, o *binding* das condições de uma RC com o contexto dos ARs envolvidos é realizado automaticamente. O IC responsável pela regra se encarrega de subscrever aos ARs participantes, para receber notificações sobre modificações no contexto requerido.

Após a construção das condições, pode-se definir diversas ações executadas por entidades chamadas de **atuadores**. Os atuadores são ARs que se subscrevem a um IC para receber notificações sobre a RC desejada. Quando uma RC é avaliada como verdadeira, todos os atuadores subscritos no IC correspondente são notificados. Então, estes atuadores podem executar ações, geralmente invocando serviços de um ou mais ARs presentes no ambiente inteligente.

Ressaltamos que os ICs assim como as fontes dos contextos envolvidos em uma dada regra e os atuadores são especializações do AR básico, e por isso herdam seu comportamento. Assim, tanto a notificação de mudança no contexto dos ARs envolvidos como a notificação dos ARs interessados é realizada por meio do padrão *publish-subscribe*, da mesma forma como discutido na Seção 2.3.2. Na Figura 2.7 são apresentadas as interações entre as entidades que compõem a interpretação de contexto.

As regras de contexto são criadas na IPGAP através de uma GUI, que permite a seleção das condições da regra intuitivamente. Ao acionar-se o ícone do dispositivo desejado

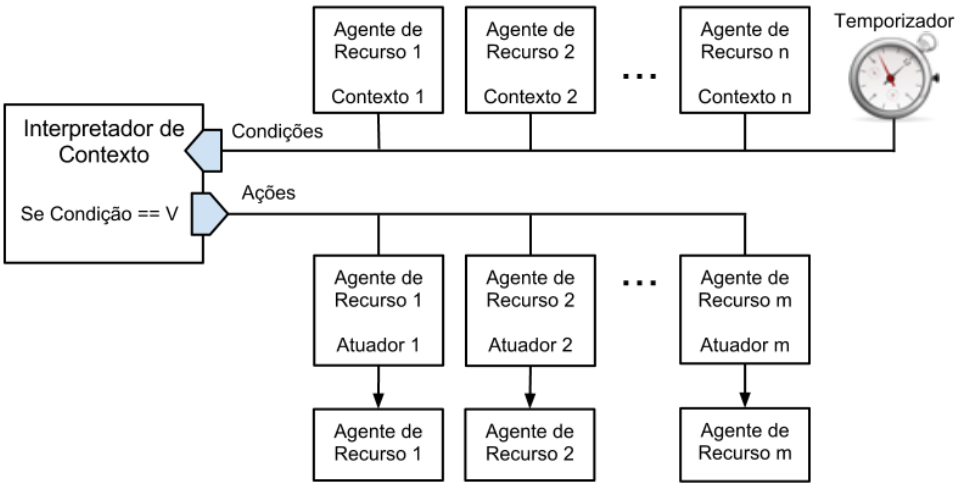


Figura 2.7: Interpretação de uma Regra de Contexto

são exibidas suas informações de contexto, que podem então ser selecionadas e incluídas na regra. Uma regra pode envolver o contexto de diversos dispositivos, combinados por conectivos lógicos. A partir da descrição da regra, as referências dos ARs envolvidos são obtidas automaticamente, e a regra é então executada. Assim, podem ser criadas regras simples, como desligar aparelhos que não estão sendo utilizados por um certo tempo, ou complexas, como acionar uma ambulância caso um morador tenha sua situação de saúde identificada como crítica, através da monitoração de seus dados fisiológicos [25]. Um estudo mais profundo sobre interpretação de contexto pode ser encontrado em [36, 37].

Capítulo 3

SmartAndroid

Neste capítulo será apresentado o projeto **SmartAndroid**, com suas principais características e funcionalidades, bem como sua relação com a IPGAP e o *framework* de suporte. Além disso, discutiremos brevemente as hipóteses assumidas em relação a comunicação e segurança. Dessa forma, objetiva-se clarificar o papel de cada conceito mencionado nessa dissertação em aplicações práticas. É importante salientar que questões como a escolha do ambiente, tecnologias e implementações relativas à assuntos como comunicação e segurança são ortogonais à proposta original, que é flexível o suficiente para que seja implementada através de diversas plataformas, tecnologias e algoritmos.

3.1 Introdução

O SmartAndroid é um projeto desenvolvido no Laboratório Tempo[1] do Instituto de Computação da Universidade Federal Fluminense, como uma implementação de referência para comprovação de conceitos propostos em [37, 39, 58]. O SmartAndroid tem por objetivo prover uma infraestrutura uniforme para construção de aplicações ubíquas/pervasivas, com foco em aplicações para ambientes inteligentes.

3.2 Android: Motivação

O SmartAndroid foi desenvolvido com base na plataforma Android do Google, que além de ser um Sistema Operacional popular e gratuito, conta com um SDK completo para desenvolvimento de aplicações. Segue abaixo um detalhamento da motivação para adoção do Android como plataforma base em nossas implementações:

[1] Laboratório de Sistemas de Tempo Real e Embarcados - `http://www.tempo.uff.br`

- **O fato de ser um projeto *open-source***[2]; Isso aumenta as chances de que versões do sistema estejam disponíveis para as mais diversas plataformas. Este requisito é fundamental para o projeto, pois deseja-se disponibilizar a maior quantidade de dispositivos compatíveis possível. De fato, temos versões do Android para TVs, relógios e uma série de outros dispositivos, inclusive placas de desenvolvimento como o Beaglebone[3], além de *smartphones* e *tablets*. Outra informação que contribui para a compatibilidade do Android, é que este executa sobre o Sistema Operacional Linux, já presente em milhares de dispositivos embarcados.

- **A facilidade de desenvolvimento para essa plataforma**; Há uma grande quantidade de material disponível na internet, incluindo uma página[4] dedicada aos desenvolvedores contendo APIs, tutoriais, e toda a documentação necessária. Além disso, a adoção da linguagem Java como linguagem de programação contribui para a diminuição da curva de aprendizagem.

- **A grande quantidade de aparelhos contendo o Sistema Operacional Android de fábrica**; Uma recente pesquisa[5] da IDC (*International Data Corporation*) mostrou que mais de 60% dos aparelhos no mercado de *smartphones* no início de 2013 saíram de fábrica com o sistema instalado (Figura 3.1);

3.3 Organização Típica

Os elementos que compõem o ambiente do SmartAndroid estão organizados de forma essencialmente distribuída, onde se comunicam diretamente uns com os outros. Os serviços básicos do sistema – como o SDR, SRR, SLR discutidos no Capítulo 2 – executam em máquinas mais robustas, assegurando assim requisitos de qualidade de serviço e tolerância a falhas.

A principal forma de comunicação entre as entidades do sistema é através de redes sem fio. Essa abordagem permite maior mobilidade dos dispositivos e a flexibilidade da rede, aspectos que auxiliam a tornar invisível a interação do usuário com o ambiente, questão de suma importância em sistemas ubíquos. A implementação de referência utiliza comunicação via Wi-Fi, por conta da grande gama de dispositivos que atualmente possui

[2] `http://source.android.com`
[3] `http://beagleboard.org/`
[4] `http://developer.android.com`
[5] `http://www.idc.com/getdoc.jsp?containerId=prUS24108913`

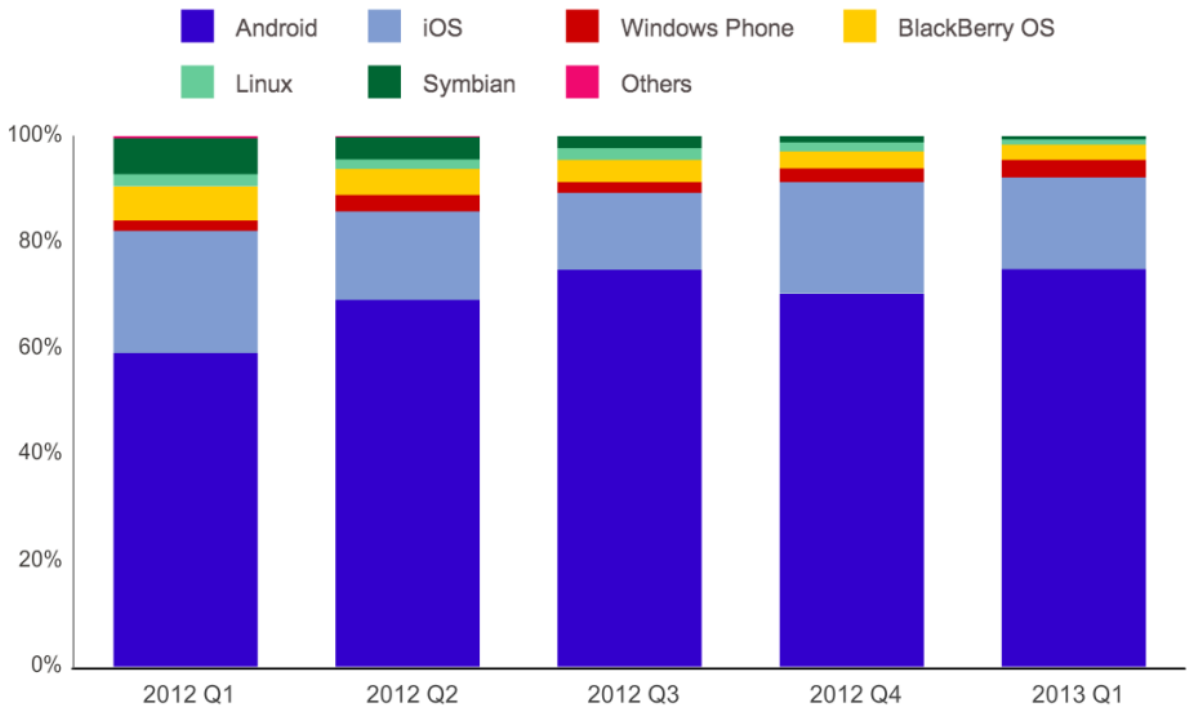


Figura 3.1: Mercado de *smartphones* do 1º trimestre de 2012 ao 1º trimestre de 2013 em relação aos seus Sistemas Operacionais

esta funcionalidade, e da segurança que esta tecnologia oferece, porém não há restrições para que se utilize tecnologias como **Bluetooth** e **Zigbee**. Podemos ver na Figura 3.2 o esquema de organização das principais entidades do sistema.

3.4 Privacidade e Segurança

A utilização de recursos computacionais integrados ao dia a dia das pessoas traz à tona a questão da privacidade e segurança dos usuários e de suas informações pessoais. Para prover essa funcionalidade, o SmartAndroid utiliza técnicas simples e estabelecidas na área, de forma que dispositivos e aplicações se comuniquem apropriadamente.

Como primeira medida, os recursos do ambiente somente podem utilizar o sistema quando autenticados através de um mecanismo de *log in*, para que sejam identificados. Assim, pode-se impedir que entidades não autorizadas se comuniquem com certos recursos e aplicações do ambiente.

A definição de perfis de utilizadores e grupos é de fundamental importância para definir e controlar o que cada usuário pode ou não fazer na residência, onde perfis diferentes possuem permissões diferentes. Perfis como *dono da casa*, *criança*, *adolescente*, *visitante*,

Figura 3.2: Organização das entidades do ambiente inteligente

administrador, *parente*, *amigo próximo* podem ser definidos, cada qual com a possibilidade de realizar tarefas distintas no ambiente. Por exemplo, é recomendável que crianças não tenham acesso à aplicações e recursos que possam oferecer risco, como o fogão ou aplicações que lidam com trancamento de portas e janelas, mesmo através de aplicações e regras de contexto iniciadas por elas. Em contra partida, essa restrição não deve impedir a utilização de outros recursos do ambiente, como recursos multimídia.

Uma outra preocupação é a possibilidade de interceptação das informações pessoais do usuário, por exemplo, por meio de programas analisadores de pacotes. Neste caso a utilização de mensagens criptografadas é uma solução viável. Mais detalhes sobre segurança e privacidade podem ser encontrados em [57].

3.5 Arquitetura em Camadas

O SmartAndroid conta com um *framework* de suporte, que provê para as aplicações pervasivas os serviços de gerenciamento básicos além de uma API contendo as instruções necessárias para a manipulação dos recursos contidos no ambiente e uma implementação de primitivas de comunicação. Estas funcionalidades utilizam a abstração de ARs para a representação dos recursos do ambiente (ver Capítulo 2).

Assim, o *framework* age como um intermediador entre as aplicações e o ambiente

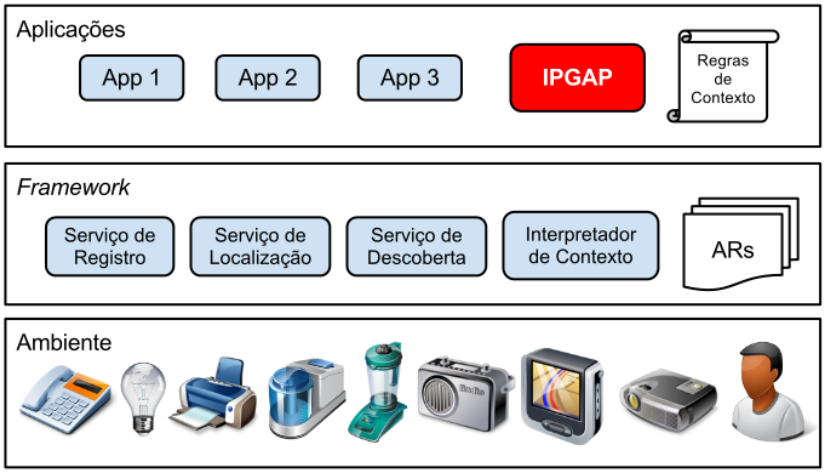


Figura 3.3: Arquitetura do SmartAndroid em Camadas

fornecendo toda a infraestrutura básica necessária. A IPGAP é, do ponto de vista do *framework*, uma aplicação pois esta se utiliza da infraestrutura provida para realizar suas atividades e se comunicar com o ambiente (Capítulo 4). Na Figura 3.3 o SmartAndroid é apresentado numa visão de camadas, contendo as aplicações pervasivas, o *framework* e o ambiente.

3.6 Simulação dos dispositivos

Os recursos utilizados nas aplicações desenvolvidas através do SmartAndroid, como sensores e atuadores, foram simulados por meio de dispositivos Android. Assim, através de aparelhos de fácil aquisição, pôde-se criar diversos cenários no projeto, à baixo custo.

A simulação é realizada com auxílio de um aplicativo que exibe a imagem do recurso desejado, permitindo que se altere seu estado por meio de toques na tela do aparelho, seguindo a técnica *Wizard of Oz* [32]. O dispositivo se registra no sistema e expõe as informações de contexto e serviços do recurso, tal qual faria um recurso real. Na Figura 3.6 podemos ver exemplos de aplicativos executando nos aparelhos para simular os recursos do ambiente. Por possuir uma tela maior, um *tablet* foi utilizado para visualização da ferramenta de prototipagem (Figura 3.4(c)).

A utilização desses aparelhos diminui grande parte do custo de desenvolvimento (financeiro e temporal) do projeto, pois não exige a aquisição de dispositivos reais, como sensores e atuadores, o que é útil nas fases iniciais de um projeto. Além disso, a integração de dispositivos reais no ambiente é suportada, provendo um ambiente completo para

(a) Lâmpada (b) Sensor de temperatura (c) IPGAP

Figura 3.4: Dispositivos Android simulando recursos

testes. O fato de um recurso ser simulado ou real é totalmente transparente para as aplicações, permitindo que os dispositivos reais e simulados sejam facilmente intercambiáveis. Isso seria um desafio se fosse preciso desenvolver toda infraestrutura a partir do zero.

3.7 Localização dos Recursos

As informações de posicionamento de um dispositivo podem ser adquiridas de diversas formas, como por exemplo, utilizando medições de sinal de radio-frequência tais como redes GSM [69,82], *Bluetooth* [50], WiFi [52,53] e *tags* RFID [40,70,91], ou mesmo, todos eles [77]. Em [51] são utilizados os sensores inerciais de um *smartphone*, que cumprem seu propósito sem a necessidade de uma infraestrutura de apoio. Outros trabalhos propõem o uso de um *smart floor* [16], capaz de localizar e rastrear pessoas ao pisarem no dispositivo, distinguindo-as inclusive de impactos realizados pelo mobiliário.

O SmartAndroid não implementa nenhuma técnica em específico para localização dos recursos, tratando e propagando essa informação em alto nível através de seus serviços. A implementação da aquisição dos dados é transparente, ficando a cargo do projetista do AR a definição de *como* a localização será obtida. Esse aspecto é importante para manter o sistema flexível a qualquer proposta, permitindo inclusive que diferentes métodos sejam utilizados, deixando o fabricante do recurso livre para escolher a técnica que melhor

funciona com seu produto.

Dessa forma, o tratamento dos aspectos referentes à localização dos recursos por parte do SLR conforme discutido no Capítulo 2, também não está limitado a uma implementação específica.

Capítulo 4

A IPGAP

Neste Capítulo apresentaremos a IPGAP, seus objetivos e os benefícios trazidos com a sua utilização. As principais funcionalidades também serão abordadas, e as possibilidades que a ferramenta proporciona serão discutidas.

Na Seção 4.1 apresentaremos uma visão geral da ferramenta. Os perfis de utilização da IPGAP serão analisados na Seção 4.2, onde trataremos das possibilidades que a ferramenta proporciona segundo os requisitos oriundos de cada tipo de usuário. Na Seção 4.3, discutiremos como o ambiente físico e seus recursos são representados na IPGAP, e como essa representação é persistida. A organização da ferramenta em aplicativos é discutida na Seção 4.4, juntamente com uma discussão sobre ligação com recursos externos à IPGAP. Finalmente, na Seção 4.5 apresentaremos funcionalidades específicas da IPGAP como emulação de movimento e composição de regras de contexto.

4.1 Introdução

A **Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas – IPGAP** tem como principal objetivo prover um ambiente controlado, em que os recursos simulados e reais interajam entre si transparentemente e sejam facilmente manipulados utilizando-se a API contida no *framework* proposto em [57]. Dessa forma aplicações pervasivas podem ser testadas no ambiente da IPGAP, permitindo que seus desenvolvedores a avaliem e, consequentemente, a aperfeiçoem.

Os recursos registrados no sistema através do SRR podem ser gerenciados diretamente pela IPGAP, por meio de seus ARs. Isso inclui a invocação dos serviços expostos por cada AR, e a visualização do seu estado interno, juntamente com o mapa do ambiente.

4.2 Perfis de utilização

Inicialmente a ferramenta proposta teve como seu principal alvo o desenvolvedor de aplicações pervasivas, visando facilitar os testes e evolução de suas aplicações. No decorrer do desenvolvimento do projeto foi observado que a IPGAP seria conveniente também para um usuário final[1], o qual teria condições de gerenciar os dispositivos da sua casa utilizando a IPGAP como um *controle remoto*. Assim, conceitualmente haveriam duas interfaces: uma atendendo os requisitos e necessidades do desenvolvedor de aplicações pervasivas, e outra para atender o usuário final. A seguir, uma descrição destes dois perfis.

4.2.1 A IPGAP para o Desenvolvedor

Conforme discutido na Seção 1.2.2, o desenvolvimento de aplicações pervasivas não é trivial, sobretudo por conta da escassez de ferramentas de desenvolvimento, ambientes de testes e dispositivos de fácil integração/manipulação. Para executar testes nesse tipo aplicação, é necessário que se tenha disponível uma infraestrutura incluindo os recursos (sensores, atuadores e outros dispositivos inteligentes), além do próprio espaço físico.

Além disso, é necessário fornecer à aplicação o *input* necessário, o que nem sempre é uma tarefa fácil. Por exemplo, para testar uma aplicação que lide com localização de pessoas através de sensores presença e *tags* de RFID, deve-se de alguma forma alimentar esses dispositivos com uma entrada, para que o comportamento da aplicação possa ser posto à prova.

Tendo em vista estes desafios, a IPGAP foi idealizada para que o desenvolvedor possa visualizar e manipular o ambiente de testes e seus recursos de forma simples, através de uma representação gráfica (Seção 4.3). O desenvolvedor pode então adicionar recursos em seu ambiente sem a necessidade de tê-los fisicamente, podendo assim executar e testar suas aplicações. Os estímulos requeridos para os testes também podem ser realizados por meio da IPGAP, através de aplicativos (Seção 4.4) que representam o recurso, além de um avatar adicionado ao mapa que dispara eventos gerados a partir da sua movimentação (Seção 4.5.2).

Outro requisito importante para o desenvolvedor é o acompanhamento da aplicação em execução. A IPGAP registra os eventos gerados pelos recursos, com o objetivo de identificar se uma aplicação está funcionando adequadamente. Conforme descrito na Seção 2.3.2, todas as vezes que uma variável de contexto de um AR é alterada, o próprio

[1] Um utilizador do ambiente inteligente, potencialmente leigo na área da computação

AR publica um evento para as entidades interessadas informando qual variável foi alterada e o valor para qual foi alterada. Quando o evento é disparado, a IPGAP exibe na tela um *pop-up* contendo a variável de contexto que foi alterada, bem como o valor que foi consolidado nesta variável após o evento (Seção 4.5.1). O evento também é gravado em um arquivo de *log*, para posterior visualização. Essa funcionalidade se dá através do registro de interesse por parte da IPGAP no contexto de todos os recursos. Note que esse registro de interesse pode ser desabilitado total ou parcialmente, para evitar o envio de mensagens desnecessariamente.

4.2.2 A IPGAP para o Usuário Final

As funcionalidades da IPGAP não estão restritas ao desenvolvedor: uma outra importante aplicação da ferramenta de prototipagem está na sua utilização pelo usuário final. Este pode utilizar a IPGAP para gerenciar o ambiente inteligente através da visualização do contexto de cada recurso e da atuação proporcionada por meio de invocações remotas aos serviços providos pelos recursos disponíveis. Isso pode ser feito através da interação com os dispositivos representados na tela da IPGAP. O usuário pode visualizar graficamente o estado de cada um dos dispositivos, e também acionar as operações disponíveis também na tela da ferramenta.

Ademais, regras de contexto (Seção 2.4) podem ser compostas de forma intuitiva através de uma GUI (Seção 4.5.3), atribuindo ao usuário a possibilidade de programar seu ambiente de acordo com suas preferências, explicitadas em forma de expressões lógicas simples [36, 37]. Essa abordagem é conhecida na literatura como *end-user programming* pois o usuário final, utilizando-se de ferramentas adequadas, é capaz de construir artefatos que em outras situações só poderiam ser desenvolvidos por um programador.

Assim, por meio das interações com a IPGAP, o usuário final pode manipular os dispositivos e criar regras de contexto que atuarão em seu ambiente e configurá-lo de acordo com suas preferências, através de um dispositivo móvel como um *smartphone* ou *tablet*, via rede sem fio.

A situação ideal é que este usuário possa controlar toda sua casa remotamente, de qualquer parte do mundo através da internet. Naturalmente isso envolve uma série de questões sobre segurança de rede, níveis de permissões, entre outras. Estes aspectos são abordados com detalhes em [57].

4.3 Representação do Ambiente

O ambiente inteligente é representado na IPGAP através de imagens que simplificam a sua visualização. Incluem-se nessas representações o mapa do ambiente, os dispositivos e as pessoas. A representação de pessoas, objetos e lugares é importante pois segundo Dey & Abowd [2] são as entidades básicas de um sistema sensível ao contexto. Assim, por meio das representações dessas entidades, desenvolvedores e usuários finais podem manipular o ambiente inteligente.

4.3.1 Mapa do Ambiente

Os lugares, ou neste caso, os cômodos do ambiente são representados na IPGAP como um mapa esquemático 2D, visto de cima. Cada cômodo tem sua área pré-definida por intermédio de um editor de mapas (Apêndice A), que define os limites de cada cômodo, as paredes e também a aparência da casa representada.

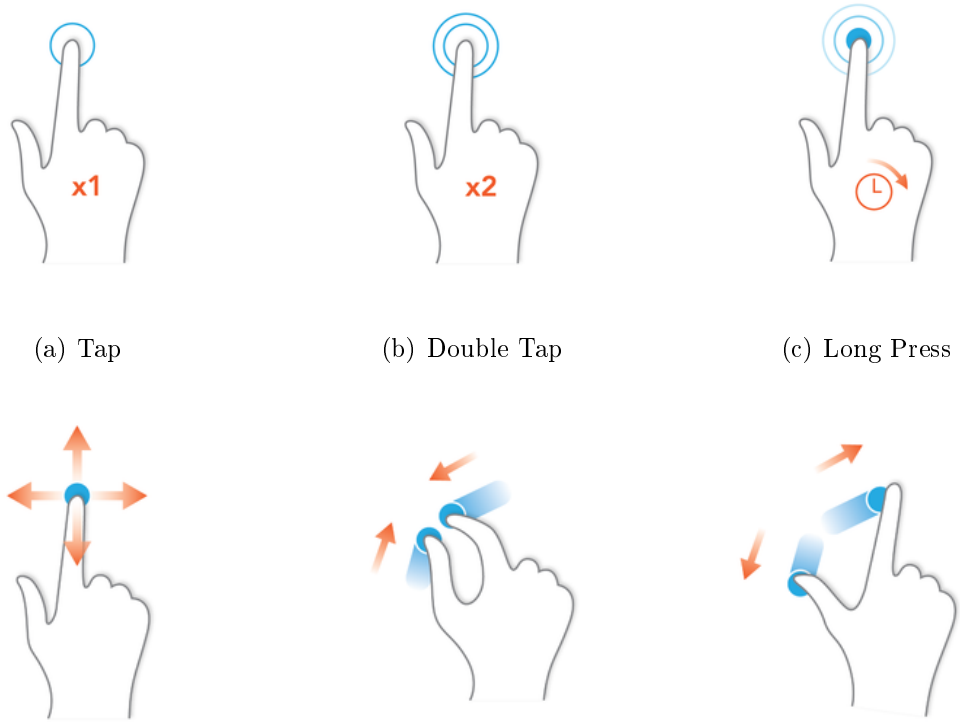
A escala utilizada é definida em tempo de construção do mapa, onde no momento de sua implantação na IPGAP pode-se configurar a relação entre *pixels* e metros. A configuração da escala é importante para que a IPGAP possa renderizar os dispositivos e gerenciar a movimentação das entidades corretamente. Os SG do sistema (Seção 2.2) utilizam como unidade o **metro**, que deve ser convertida para *pixels* no momento em que a medida for utilizada na IPGAP.

A IPGAP dá suporte ao reconhecimento de alguns gestos básicos, que são utilizados para manipular o mapa. Na Figura 4.1[2] vemos os tipos de gestos reconhecidos na IPGAP. Por exemplo, pode-se utilizar os gestos apresentados nas Figuras 4.1(e) e 4.1(f) para, respectivamente, diminuir e aumentar o *zoom* do mapa, ao passo que o gesto da Figura 4.1(d) faz com que o mapa seja movido.

4.3.2 Recursos

Os recursos do ambiente são visualizados na interface de prototipagem através ícones, para que o desenvolvedor ou usuário final possa manipulá-los. Os ícones dos recursos são posicionados na tela de acordo com um mapeamento entre sua posição real e o sistema de coordenadas do mapa esquemático, de forma que sejam apresentados nos mesmos cômodos, e aproximadamente na mesma posição em relação ao mapa, onde se encontrem

[2]Imagens retiradas de `http://en.wikipedia.org/wiki/Multi-touch`



(a) Tap (b) Double Tap (c) Long Press

(d) Pan (e) Pinch Close (f) Pinch Open

Figura 4.1: Reconhecimento de Gestos na IPGAP

os recursos fisicamente.

Pode-se mover o recurso no mapa da IPGAP através da funcionalidade de *arrasta-e-solta*. Após o ícone ser posicionado em um local, a IPGAP altera a posição do recurso no sistema, através do AR correspondente. Assim, essa informação fica disponível de forma global, para que outras entidades acessem a informação.

É importante mencionar que os ícones são um "espelho" dos recursos presentes no ambiente inteligente. Estes estão registrados no sistema através do SRR, e podem ser descobertos e utilizados por aplicações.

Aplicando o gesto apresentado na Figura 4.1(a) nos ícones dos recursos, é possível executar *aplicativos* (Seção 4.4). Além disso, ações como *desregistrar recurso*, *listar variáveis de contexto* e *remover ícone* também podem ser executadas. Note que a ação de remoção do ícone não implica necessariamente no *desregistro* do respectivo recurso. Se o recurso for um recurso externo à IPGAP, ele não será *desregistrado*.

4.3.3 Pessoas

A entidade *pessoa* é tratada pelo *framework* em geral, como um tipo especial de recurso. A exemplo dos recursos genéricos, as pessoas também possuem ARs que as representam, porém podem agregar outros ARs que representariam sensores corporais, *tags* de RF-ID, entre outros.

Na IPGAP, a pessoa é um avatar que pode ser adicionado no mapa, assim como os outros recursos, onde sua movimentção realizada pela **Interface de Emulação de Movimento – IEM** (Seção 4.5.2).

4.3.4 Persistência

Uma questão fundamental para uma ferramenta de prototipagem é a possibilidade de que seu estado seja persistido, a fim de que se possa salvar o trabalho e continuar mais tarde, mesmo após o desligamento do dispositivo.

A IPGAP cumpre esse papel salvando informações dos dispositivos que foram adicionados no mapa juntamente com suas respectivas posições. Além disso, no caso de agentes criados dentro da própria IPGAP, também é salvo seu estado interno. Dessa forma, os usuários podem recuperar qualquer cenário criado previamente, a partir de qualquer dispositivo.

Em nossa implementação, o estado é salvo por meio da *serialização* dos objetos correspondentes aos recursos presentes no mapa, e gravação dos *bytes* resultantes em arquivo.

O salvamento/carregamento/eliminação do estado do ambiente pode ser executado através do menu da ferramenta. Ressaltamos que, para que o carregamento proceda com sucesso, deve ser feita uma breve análise dos dados de modo a evitar inconsistências, como a adição no mapa de agentes que não estão mais registrados no sistema.

4.4 Aplicativos

Nossa ferramenta utiliza o conceito de aplicativos, atualmente atribuído ao universo dos *smartphones* e *tablets*, embora exista na grande maioria dos Sistemas Operacionais de propósito geral há bastante tempo (com uma semântica ligeiramente diferente). No contexto das aplicações móveis, existem lojas virtuais (como Google Play e Apple Store) que disponibilizam uma base com milhares de aplicativos que podem ser baixados pelos usuá-

rios. Dessa forma, pode-se facilmente customizar o ambiente com a instalação de novos *softwares*.

No contexto da IPGAP, um aplicativo possui uma GUI que representa a interface de um determinado recurso, além de poder invocar operações do AR do mesmo (potencialmente em outro *host*), o que nos permite manipular e visualizar seu estado interno de forma intuitiva. Essa funcionalidade se dá através de primitivas de comunicação síncrona, semelhante às utilizadas no RPC (*Remote Procedure Call*) e RMI (*Remote Method Invocation*). Assim como nas abordagens do RPC e RMI, existe um *stub* – um componente que possui a mesma interface do recurso-alvo, porém sua implementação contém chamadas remotas para este recurso, agindo como um *proxy*.

Imagine um aplicativo capaz de exibir a interface de um fogão (Figura 4.2), onde o usuário pode através dela, acender uma boca ou o forno do aparelho. Ao acionar-se alguma funcionalidade do aplicativo do fogão, essa operação deve invocar a sua correspondente no AR do fogão real (que pode ser embutido no *hardware* do fogão real pelo seu fabricante. Ver Seção 4.4.3), conforme mostramos no esquema da Figura 4.3. Note que para isso, o aplicativo do fogão deve possuir um objeto *stub* que encaminha a chamada para o fogão real. Um aplicativo pode ser acionado através da IPGAP por meio de sua representação no mapa, acionando seu ícone (ver Seção 4.3), passando a exibir a interface do dispositivo que representa, como na Figura 4.2.

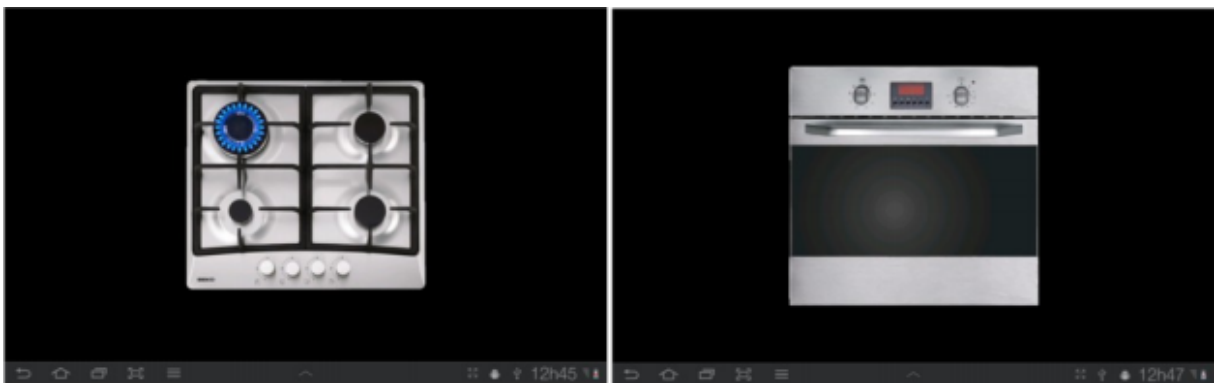


Figura 4.2: Aplicativo do fogão em um *tablet*. Visão das bocas e do forno

O caminho inverso também deverá ocorrer, ou seja, o que acontecer no fogão real também deve ser refletido no aplicativo do fogão. Isso é fruto de uma comunicação assíncrona estabelecida entre o fogão real e o aplicativo, provida por uma implementação do padrão *publish-subscribe*. Assim, dizemos que o aplicativo em questão é **interessado** nas mudanças que ocorrerem no fogão real, ou seja, sempre que o fogão real alterar seu estado, o aplicativo será notificado através de um evento.

4.4.1 Simuladores

É comum que o desenvolvedor necessite de diferentes entidades presentes em um ambiente inteligente para testar suas aplicações. Isso pode se tornar inviável conforme o número de entidades cresce, pois envolve os custos com os equipamentos e principalmente com a integração destes. Por esse motivo a IPGAP inclui um conjunto de aplicações (que pode ser ampliado) que simulam os principais dispositivos presentes em um ambiente inteligente, como lâmpada, fogão, TV, ar-condicionado assim como sensores de localização, temperatura, umidade, entre outros. Dessa forma, o desenvolvedor pode criar suas aplicações utilizando esses componentes – chamados de simuladores – como se fossem os equipamentos reais.

Ressaltamos que os simuladores expõem seus serviços e o seu contexto no ambiente através de ARs, o que torna uma possível troca de um simulador por dispositivo real transparente para o desenvolvedor. Os simuladores fornecidos com a IPGAP são também considerados aplicativos. Um aplicativo simulador deve conter um código para gerar valores (aleatoriamente ou segundo diretivas de simulação) além de conter o AR do referido recurso simulado. Já um aplicativo comum, contém um *stub* que encaminha os parâmetros passados nas operações chamadas para o AR do recurso alvo, através de invocações remotas.

Figura 4.3: Aplicativo do fogão atuando no fogão real

Um desenvolvedor pode, seguindo um estilo bem definido de programação, criar seus próprios aplicativos. Um exemplo desse caso seria um centro de pesquisas que está desenvolvendo um novo tipo de sensor. O primeiro passo seria utilizar um simulador para

este sensor e testá-lo no ambiente da IPGAP juntamente com os demais recursos. Dessa forma pode-se testar o sensor antes mesmo de se ter um protótipo físico completo.

4.4.2 Instalação de Aplicativos

Como citado anteriormente, nossa ferramenta permite a instalação de aplicativos no ambiente de prototipagem, de maneira similar ao que ocorre nos sistemas operacionais para plataformas móveis. Dessa forma o desenvolvedor tem a possibilidade de estender esse ambiente segundo as suas necessidades. Idealmente, os usuários da IPGAP podem baixar os aplicativos e/ou simuladores em uma loja aos moldes da Google Play e Apple Store, onde pode-se fazer buscas por aplicativos, visualizar sua descrição, fazer o download e assim, instalá-los na IPGAP. Esse processo se dá de forma automatizada, o que torna essa opção útil para o usuário final, podendo este, acrescentar os aplicativos que quiser na interface de visualização do ambiente.

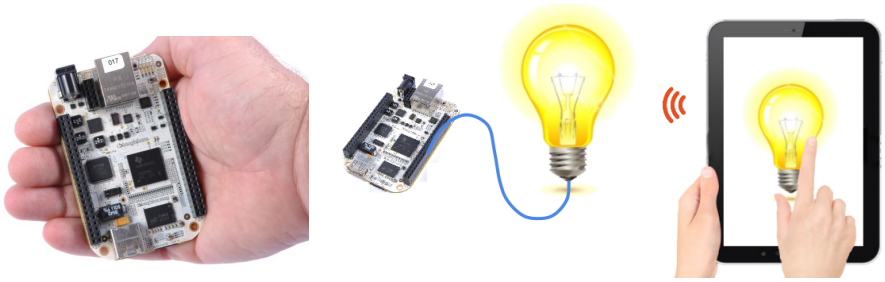
4.4.3 Recursos externos à IPGAP

Recursos reais como eletrodomésticos e sensores podem fazer parte do sistema, enriquecendo assim os testes realizados na aplicação a ser desenvolvida, além de possibilitar o uso da IPGAP pelos usuários finais para controlar dispositivos reais de ambientes inteligentes. Para isso os dispositivos devem possuir as APIs padronizadas do *framework* embutidas, de forma a suportar as suas funcionalidades básicas.

Uma alternativa seria desenvolver um componente (*wrapper*) que implemente ambas as APIs (a API de nossa proposta e a API nativa do recurso desejado). Quando esse componente recebe uma chamada através de nossa API, ele encaminha a chamada para o recurso-alvo, utilizando a API de seu fabricante. Ou seja, o componente faz a conversão de uma chamada do nosso sistema para uma chamada nativa da API do aparelho em questão. Essa técnica é mais viável nas etapas de desenvolvimento, pois não depende de que se tenha um recurso que utiliza o padrão do sistema nativamente.

Nosso grupo está realizando testes com o Beaglebone[3] com o objetivo de construir protótipos de diversos dispositivos. O Beaglebone (Figura 4.4(a)) é uma placa de desenvolvimento de tamanho reduzido e baixo custo, que contém pinos para entrada e saída e um processador ARM AM335x, que suporta a execução de sistemas como Ubuntu e Android. Através desse equipamento podemos, por exemplo, conectar uma lâmpada

[3] `http://beagleboard.org/`



(a) Beaglebone (b) Esquema com Beaglebone e Lâmpada

Figura 4.4: Experimentos com Beaglebone

comum e controlá-la através de um relé ligado aos pinos de I/O do Beaglebone. A Figura 4.4(b) ilustra essa possibilidade.

4.5 Modos Auxiliares

4.5.1 Modo de Depuração

Um dos principais desafios do desenvolvedor de aplicações pervasivas é depuração. De acordo com [86], o fluxo de controle desse tipo de aplicação é muito difícil de ser seguido utilizando ferramentas clássicas de depuração, pois estas estão geralmente associadas a um único processo, executando em um único dispositivo. Isso se dá pois uma aplicação pervasiva é essencialmente um sistema distribuído. Em [42] são elucidados vários outros aspectos da depuração em sistemas distribuídos em geral. Algumas propostas, como [45, 48], tentam resolver esse problema utilizando versões centralizadas da aplicação, ou utilizando virtualização.

A IPGAP, no entanto, utiliza uma abordagem baseada em eventos para auxiliar o desenvolvedor na depuração da aplicação. A partir do mecanismo de subscrição de eventos provido pelo *framework*, é possível exibir na tela da IPGAP as informações dos recursos. A IPGAP possui um componente chamado **Resource Manager – RM**, que é responsável pela exibição dos eventos. Toda vez que um recurso é adicionado ao mapa, o RM se registra como interessado em todas as suas VCs. Assim, quando ocorre um evento, ou seja, quando alguma VC é alterada, o RM é notificado. Nesse momento é exibido um *pop-up* na IPGAP, junto ao ícone do recurso correspondente.

O uso desta técnica permite, além da depuração da aplicação, o registro das atividades realizadas no ambiente em arquivos de *log*. Os dados registrados podem posteriormente ser utilizados para fins de auditoria, ou mesmo como entrada para sistemas de mineração

de dados para identificar preferências dos usuários. Na Figura 4.5 podemos ver a IPGAP exibindo os eventos de alguns recursos.

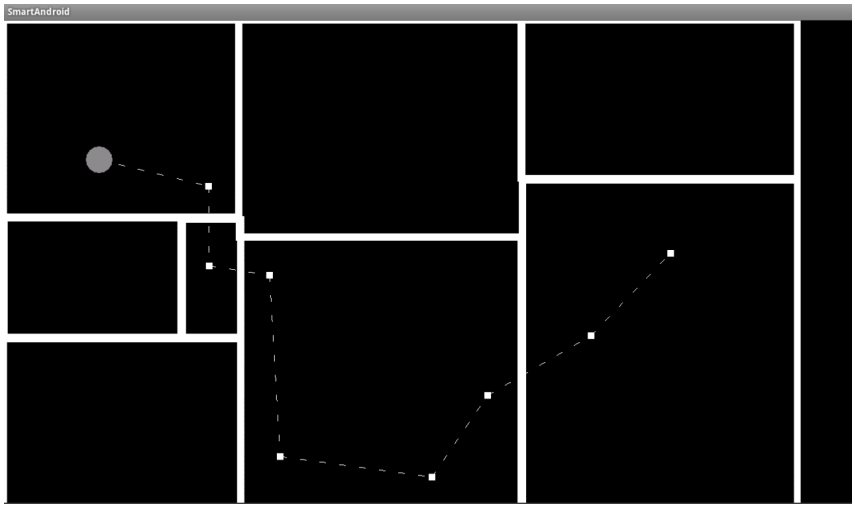
Figura 4.5: Eventos gerados pelos recursos exibidos na IPGAP

4.5.2 Interface de Emulação de Movimento

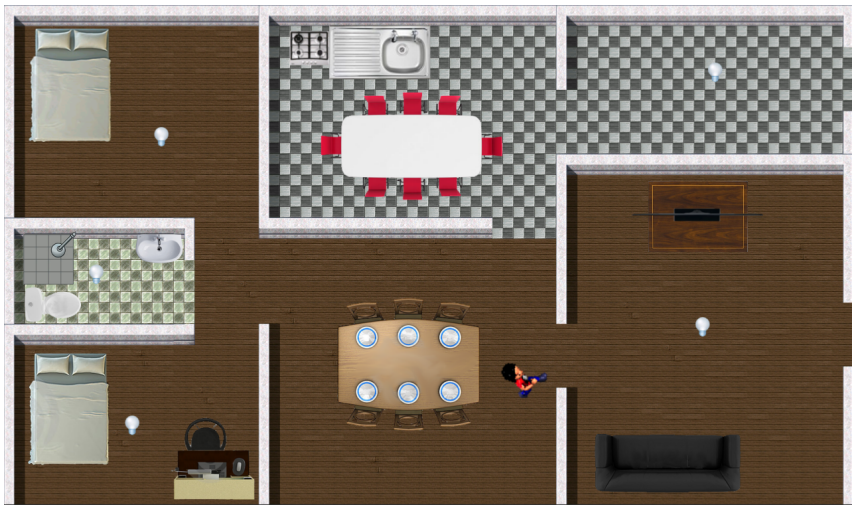
Em testes de aplicações pervasivas que lidam com localização de dispositivos e pessoas, é importante que se tenha uma ferramenta que simule o movimento dessas entidades. A IPGAP inclui uma ferramenta que tem esse propósito chamada **Interface de Emulação de Movimento – IEM**.

Por meio da IEM, o desenvolvedor pode controlar um dispositivo ou avatar no mapa do ambiente provido pela IPGAP, simulando sua movimentação. A IEM obtém do SLR as informações sobre cada cômodo do ambiente, para renderizar a área ocupada por eles na tela. Formas geométricas representam o recurso, que pode ser movimentado manualmente, arrastando a entidade pelo mapa, ou através da definição de um trajeto pré-definido.

Podemos ver na Figura 4.6 um exemplo de uso da IEM. Na Figura 4.6(a) vemos a definição de um trajeto para o recurso. Esse trajeto pode ser executado quantas vezes for necessário para os testes, e pode ser executado passo a passo, ou seja, parando em cada ponto definido (os pequenos retângulos na Figura 4.6(a)) ou sem parar até o final. Além



(a) IEM



(b) IPGAP

Figura 4.6: Definição do trajeto do usuário sendo aplicado na IPGAP

disso, pode-se definir trajetos para vários recursos ao mesmo tempo, possibilitando testes mais sofisticados na aplicação.

A criação de casos de testes automatizados para testes em batelada poderia ser feita através de um *script* simples, que contivesse diretivas para representar o movimento, posições, velocidade do movimento, pausas, entre outros aspectos. Essas possibilidades são discutidas na Seção 7.2.1.4.

4.5.3 Interface de Composição de Regras de Contexto

Através da IPGAP é possível a criação/edição de regras de contexto que envolvam os dados dos recursos presentes no ambiente inteligente. Por meio de uma GUI, o desevolvedor ou

mesmo o usuário final pode compor suas regras e registrá-las no sistema, para que sejam interpretadas por um Interpretador de Regras de Contexto (Seção 2.4).

(a) O fogão e suas variáveis de contexto (b) Operadores lógicos

Figura 4.7: Componentes da Interface de Composição de Regras

A criação se dá mediante a seleção das informações de contexto de um recurso, juntamente com conectivos lógicos e comparadores. Ao concluir a regra de contexto, pode-se associá-la a um ou mais atuadores. Os atuadores subscritos no IC serão notificados no momento em que a regra for avaliada como verdadeira e executarão alguma ação a partir dessa informação.

Na IPGAP uma regra é criada através do menu de composição de regras de contexto. A partir desse passo, a IPGAP passa a estar no modo *composição de regras*, onde o usuário pode visualizar as variáveis de contexto de um recurso qualquer e utilizá-las nas condições e ações de uma regra 4.7(a).

No modo *composição de regras* também é exibida uma barra lateral, que mostra a forma textual da regra em composição, de forma que erros na lógica sejam detectados pelo criador da regra o quanto antes. Os operadores lógicos e outros itens que compõem um regra (como parênteses) são selecionados pelo usuário através de uma caixa de ferramentas 4.7(b).

O usuário é guiado entre as etapas necessárias à composição da regra. São exibidas mensagens informando quando este deve selecionar uma variável de contexto, ou escolher um operador lógico. Ao finalizar a construção da regra, pode-se cadastrar um ou mais atuadores que serão automaticamente subscritos à regra recém criada. Então o usuário é guiado ao modo *atuação*, onde pode selecionar um AR no mapa e escolher uma ou mais operações, que serão executadas quando a regra é satisfeita. Por exemplo, o ar-

condicionado poderia ser selecionado, escolhendo-se a operação *desliga*. Assim, quando a regra de contexto fosse satisfeita, o ar-condicionado seria desligado.

Capítulo 5

Aplicações

Neste capítulo serão apresentadas aplicações que foram desenvolvidas com o intuito de avaliar o ambiente da IPGAP. As aplicações serão descritas detalhadamente, incluindo uma discussão sobre as hipóteses assumidas, a arquitetura da solução e implementações. Em seguida serão apresentados os passos para a configuração do ambiente na IPGAP, a execução da aplicação e os resultados obtidos.

5.1 CarePlanReminder

Nos últimos anos o número de aplicações que viabilizam a Assistência Domiciliar à Saúde (ADS) (*Home Health Care*) vem aumentando, sobretudo por conta do crescimento do número de pacientes idosos e com doenças crônicas. A utilização da ADS traz diversos benefícios, como a diminuição de custos com internação nos hospitais e a possibilidade de maior contato do paciente com seus familiares e entes queridos.

Em um ambiente típico da ADS, um profissional de saúde pode prescrever ao paciente um conjunto de tarefas com horários e periodicidades pré-definidos, como por exemplo, tomar uma medicação ou fazer um dado exercício. Essas prescrições devem ser cumpridas pelo paciente, que é acompanhado pelo profissional de saúde remotamente, além de cuidadores e familiares. Esse conjunto de prescrições é conhecido como *plano de cuidados* (*care plan* – CP).

Uma das funcionalidades de uma aplicação de ADS, é a utilização de um mecanismo para alertar o paciente no momento em que deve executar alguma tarefa presente no CP, com objetivo de aumentar sua adesão ao tratamento.

O **CarePlanReminder** é uma aplicação que gerencia prescrições médicas, possibi-

litando além de sua exibição/edição, a geração de um alerta quando é chegado o momento de executar uma tarefa prescrita ao paciente. Atualmente, diversas aplicações com esse mesmo intuito estão disponíveis nas lojas virtuais de aplicativos para *smartphones* e *tablets*. O que difere o CarePlanReminder das outras é a *sensibilidade ao contexto* (Capítulos 1, 2) utilizada no momento do envio do alerta.

Em um aplicativo comum, a mensagem de alerta é mostrada no próprio aparelho, ou enviada por meio de mensagens texto ou e-mail. Essa abordagem não é eficaz, pois nem sempre o paciente está próximo ao aparelho ou com acesso à internet ou telefone celular, e obrigá-lo a utilizar esses serviços todo o tempo se torna uma opção intrusiva. O CarePlanReminder envia a mensagem de alerta ao dispositivo mais próximo do paciente, permitindo-o receber o alerta em qualquer cômodo que esteja, mesmo sem estar junto a um *smartphone*.

Dessa forma, o alerta pode ser enviado a diversos tipos de equipamentos, como *smartphones*, *tablets*, TVs, equipamentos de som e *notebooks*, desde que estejam preparados para utilizar o padrão proposto pelo **SmartAndroid**. Opcionalmente, o alerta pode ser enviado também através dos meios tradicionais (e.g. mensagens texto para telefones celulares ou e-mail), inclusive para os familiares ou cuidadores.

5.1.1 Hipóteses

O cenário necessário para o funcionamento CarePlanReminder está baseado nas seguintes premissas:

(i) Os ARs referentes aos dispositivos mencionados acima devem implementar uma interface padronizada, de forma que eles possam ser referenciados como instâncias de um mesmo tipo. Essa hipótese é importante para que a busca por ARs ocorra adequadamente. O SDR e o SLR utilizam polimorfismo nas operações que envolvem buscas por tipo, e necessitam que hierarquia de tipos esteja bem definida.

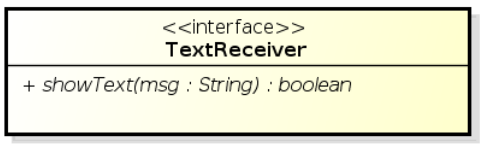


Figura 5.1: Interface TextReceiver

(ii) Como consequência do item (i), os ARs mencionados devem possuir uma operação padronizada, que receba uma *String* como parâmetro de entrada e a exiba segundo

as especificações do dispositivo. Por exemplo, a TV recebe a *String* e a exibe na tela e o aparelho de som utiliza a API *TextToSpeech* para transformá-la em som e tocá-la.

Os requisitos impostos pelas hipóteses (i) e (ii) são oferecidos pelo SmartAndroid em sua hierarquia de tipos padrão. O desenvolvedor pode implementar a interface *TextReceiver* para que seu AR seja encontrado em buscas por essa interface, e possa oferecer esse serviço (Figura 5.1). Isso não impede que sejam criadas outras interfaces, customizando a hierarquia padrão. No caso da utilização de uma interface customizada, a consulta ao serviços de busca poderá ter que sofrer alterações.

5.1.2 Arquitetura da Solução

O núcleo da solução, conforme pode ser visto na Figura 5.2, é composto pelos componentes **Task Manager**, **Notification Service** e **Alarm Manager**. O componente Task Manager é responsável pelo gerenciamento das tarefas prescritas, obtidas diretamente do CP. Assim que uma tarefa é inserida no CP do paciente, o Task Manager realiza seu agendamento através do Alarm Manager, levando em conta os horários de início e fim, além da periodicidade configurados para a prescrição. Assim, o Task Manager é notificado através de um evento gerado pelo Alarm Manager, no momento em que a prescrição deve ser executada. Nesta implementação, o componente Alarm Manager foi construído em cima da API de agendamento de tarefas presente no SDK do Android (AlarmManager[1]).

Quando a notificação por parte do Alarm Manager ocorre, o Task Manager dispara um evento para o componente Notification Service. Este evento ocorre localmente (em relação à aplicação), e para sua implementação foi utilizado o padrão de projeto *Observer*. Ao receber este evento, o componente Notification Service realiza uma consulta ao SLR para obter uma lista de ARs, ordenada por proximidade. Finalmente, a mensagem é enviada para o dispositivo mais próximo.

Na Listagem 5.1 vemos o código referente ao método de *callback* que trata o evento disparado pelo componente Task Manager. A assinatura do método, na linha **2**, mostra os parâmetros enviados na chamada, representando o nome e descrição da prescrição. Na linha **10**, é realizada uma consulta ao SLR, buscando-se a lista de todos os ARs do tipo **TextReceiver**, ordenados por proximidade (ordem crescente) em relação ao AR **user**.

Ainda em relação à linha **10**, destacamos dois pontos: O primeiro ponto é sobre a

[1] `http://developer.android.com/reference/android/app/AlarmManager.html`

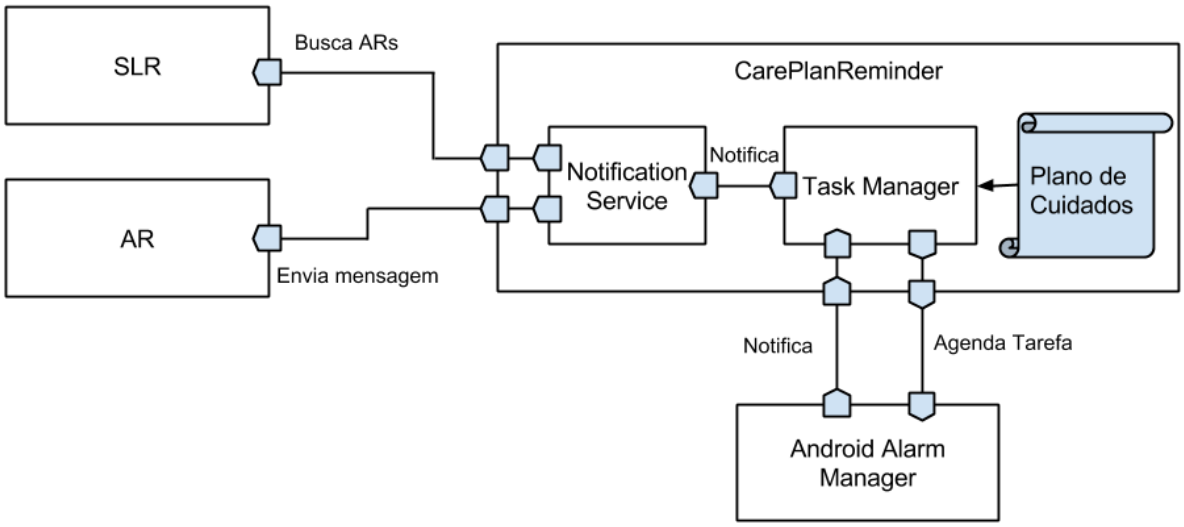


Figura 5.2: Arquitetura do CarePlanReminder

utilização da interface **TextReceiver** que, conforme discutido anteriormente, é definida pela sistema. Poderia ser utilizada em seu lugar uma interface definida pelo desenvolvedor, desde que os ARs que deseja-se buscar implementem essa interface. O segundo ponto, é sobre a variável **user**, passada por parâmetro no método **searchByProximity**. Essa variável referencia um AR especializado para representar uma pessoa, e é definida em uma fase de configuração da aplicação, onde o paciente é escolhido. A Figura 5.4(a) mostra um botão com rótulo *"choose the patient..."*, onde esse procedimento pode ser realizado. Uma busca no SDR então retorna a referência do AR correspondente à identificação do paciente escolhido, que é armazenada na variável **user**.

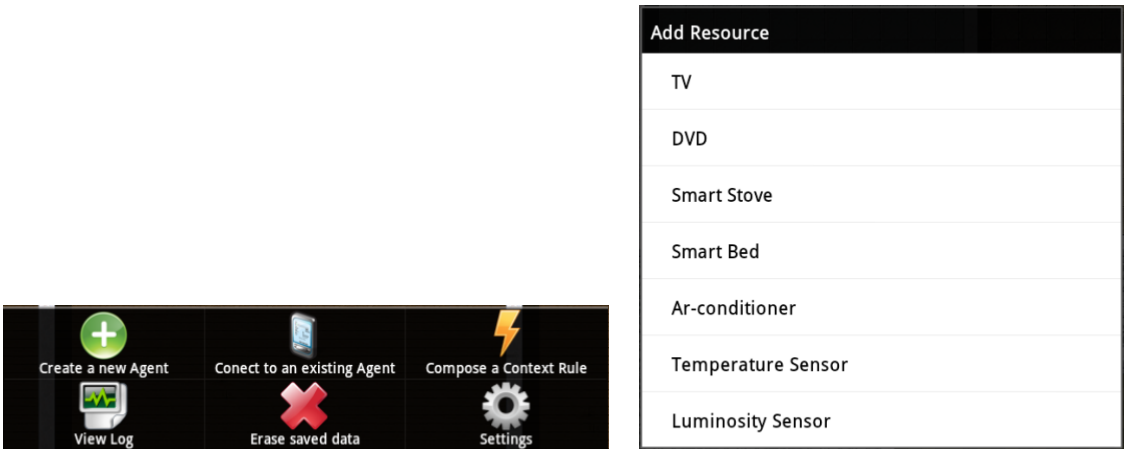
Listagem 5.1: Tratamento do Evento de Prescrição no CarePlanReminder

```
1  @Override
2  public boolean onPrescriptionFired(String name, String desc) {
3
4    boolean ret = false;
5
6    String msg = "Está na hora da prescrição " + name + ".";
7
8    if (desc != null) msg += " Descrição: " + desc;
9
10   ResourceAgent[] agents = slr.searchByProximity("TextReceiver", user);
11
12   if (view != null && view.length > 0) {
13     TextReceiver rec = (TextReceiver) agents[0];
14     ret = rec.showMessage(msg);
15   }
16   return ret;
17 }
```

Finalmente, na linha **13** selecionamos o agente mais próximo (primeira posição do vetor), e na linha **14** a mensagem é enviada através do método **showMessage**, definido na interface **TextReceiver**.

5.1.3 Visualização na IPGAP

A IPGAP permite a visualização do ambiente executando, incluindo os eventos disparados e a manipulação dos recursos. O primeiro passo para a configuração do ambiente na IPGAP é a população do mapa com os recursos desejados. Nesta aplicação, os recursos principais são os dispositivos que possam exibir uma mensagem (*tablets*, TVs, monitores, *smartphones*) além de um avatar representando o paciente.



(a) Menu principal da IPGAP

(b) Menu para escolha de Recursos na IPGAP

Figura 5.3: Menu para escolha de Recursos na IPGAP

Através do menus apresentados na Figura 5.3 pode-se adicionar esses dispositivos no mapa do ambiente. Ao escolhermos a opção *"Create a new Agent"* do menu principal (Figura 5.3(a)), é exibida a uma lista de recursos simulados que podem ser inseridos no mapa (Figura 5.3(b)). Cada recurso pode ser posto no local desejado, por meio da aplicação de arrasta-e-solta em seus respectivos ícones. Ressaltamos que ao adicionarmos um recurso simulado na IPGAP, este se registra no SRR e fica disponível no sistema para que qualquer outro recurso possa interagir com ele. A inserção do avatar do paciente se dá de forma semelhante, porém conforme descrito na Seção 4.5.2, deve-se utilizar a IEM para movimentá-lo pela casa.

Ao término da configuração do ambiente executamos o CarePlanReminder em um *smartphone* (Figura 5.4(a)), e cadastramos uma prescrição para o paciente. Deve-se informar o início e fim do tratamento ou atividade, e a periodicidade em que deve ser

(a) GUI (b) Lembrete de uma Prescrição

Figura 5.4: O CarePlanReminder em Execução

executada expressa em horas, dias e semanas. Opcionalmente, pode-se cadastrar a descrição desta prescrição. Um passo importante é escolha do paciente – o usuário relativo ao avatar recém criado no mapa da IPGAP. A aplicação necessita do AR que representa o usuário, para poder obter sua posição.

Quando é chegada a hora de executar a prescrição, são realizadas consultas aos serviços do sistema para adquirir o dispositivo mais próximo do usuário, que seja capaz de exibir a mensagem de alguma forma. Uma mensagem é exibida neste dispositivo contendo um lembrete sobre o que o usuário deve fazer. Isso fica visível na IPGAP por meio de *pop-ups* que aparecem próximos ao dispositivo, informando o que foi alterado em seu estado – neste caso, exibindo o texto da mensagem recebida.

Neste exemplo, no momento da execução da prescrição o paciente se encontrava na sala e o dispositivo mais próximo foi a TV deste mesmo cômodo, a qual exibiu a mensagem de lembrete. Na Figura 5.4(b) vemos este cenário representado na IPGAP, com a TV exibindo o lembrete.

5.2 MediaFollowMe

Considere o seguinte cenário: um usuário está assistindo a um filme através do *Blu-ray* na TV da sala de sua casa. Caso seja necessário se deslocar para outros cômodos da

residência, normalmente o usuário necessita pausar o filme para não perder a sequência da trama. Porém, dependendo de quanto tempo seja gasto, pode-se perder a dinâmica do filme, e muitas vezes o interesse por ele se vai completamente.

Uma alterativa seria, quando factível, levar o filme para o outro cômodo, e assisti-lo em outros aparelhos de TV e *Blu-ray*. Isso demandaria um pequeno trabalho extra, por exemplo, pôr o filme no mesmo trecho em que havia parado. E ainda, isso não seria possível em outras ocasiões, como estar assistindo a um filme por um canal de TV por assinatura, em vez de pelo *Blu-ray*. As TVs por assinatura com recursos HD permitem que se pause um filme, porém geralmente a migração do filme no para aparelhos em outro cômodo não é trivial.

Tendo em vista esses fatos, o aplicativo **MediaFollowMe** permite que um *stream* de mídia (músicas, filmes, etc) seja migrado para o dispositivo mais próximo do usuário, desde que este dispositivo seja capaz de executá-lo. O usuário, que porta uma *tag* de RF-ID, pode ser identificado por sensores de presença, os quais disparam eventos quando ele entra ou sai de um cômodo. Estes eventos são utilizados pela aplicação em dois momentos: primeiro, para identificar quando um usuário deixou o cômodo, e que então a execução da mídia deve ser pausada; depois para identificar em qual cômodo ele entrou, onde deve buscar um outro dispositivo para tocar a mídia.

Através de consultas aos serviços básicos do sistema (Seção 2.2), a aplicação realiza consultas para obter uma lista de dispositivos ordenados por proximidade, de forma análoga ao procedimento adotado na aplicação apresentada na Seção 5.1.

5.2.1 Hipóteses

Para que este exemplo seja factível, são assumidas algumas premissas em relação às funcionalidades dos dispositivos utilizados. São elas:

(i) Os sensores de presença referidos são equipados com algum mecanismo para identificar o usuário que entra no cômodo (não apenas identificar que há alguém no cômodo), como por exemplo, um leitor de RF-ID [91].

(ii) Os dispositivos emissores da mídia possuem a funcionalidade de transmitir o *stream* para dispositivos visualizadores (que obviamente, devem possuir a funcionalidade de receber e executar o *stream*). Uma das formas de implementar efetivamente essa abordagem seria através de transmissores/receptores sem fio HD. Fabricantes

como HP[2], IOGear[3] e ActionTech[4], entre outros, distribuem esse tipo de aparelho no mercado.

(iii) As hipóteses sobre obtenção da localização dos dispositivos e do usuário são resolvidas pelo *framework* proposto em [57] (ver Seção 3.7).

(iv) Semelhantemente aos itens (i) e (ii) das hipóteses apresentadas na Seção 5.1.1, deve-se ter uma interface que seja implementada por todos os dispositivos que possam executar a mídia. Isso permite a busca do tipo definido por esta interface no repositório de recursos.

5.2.2 Arquitetura da Solução

No desenho da solução do MediaFollowMe apresentado na Figura 5.5, vemos a interação da aplicação com os serviços **SDR** e **SLR**, além dos ARs dos sensores de presença e do emissor de mídia. Note que pessoas também são representadas no sistema através de ARs. O AR que representa uma pessoa, na verdade, agrega diversos outros ARs, como os de sensores corporais, emissores de RF-ID, entre outros dispositivos.

O SDR é utilizado para se obter as referências dos sensores de presença da casa, a fim de receber notificações no momento em que o usuário entrar ou sair de um cômodo. Essa etapa é realizada em tempo de configuração da aplicação, juntamente com a escolha do emissor da mídia e o usuário a ser "seguido" pela mídia.

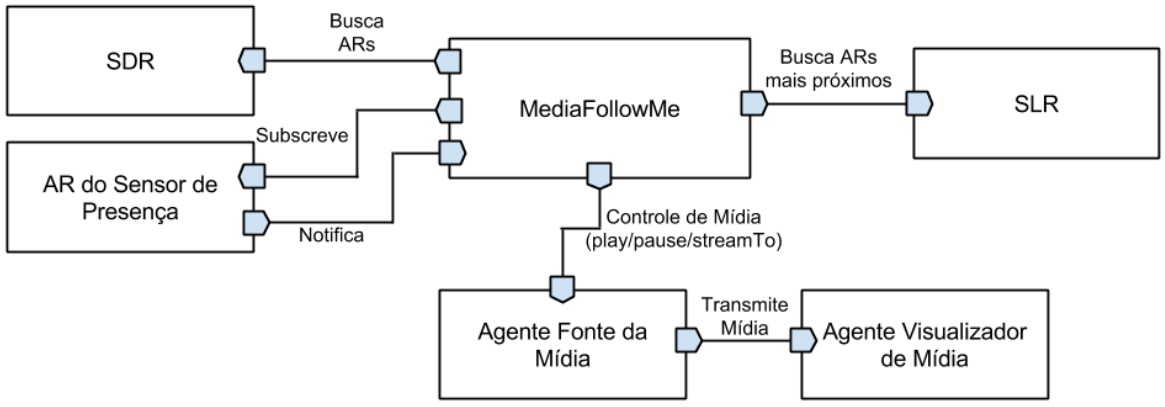


Figura 5.5: Arquitetura do MediaFollowMe

[2] http://www.shopping.hp.com/en_US/home-office/-/products/Accessories/Specialty-Products/QE389AA
[3] http://www.iogear.com/product/GW3DHDKIT/
[4] http://www.actiontec.com/218.html

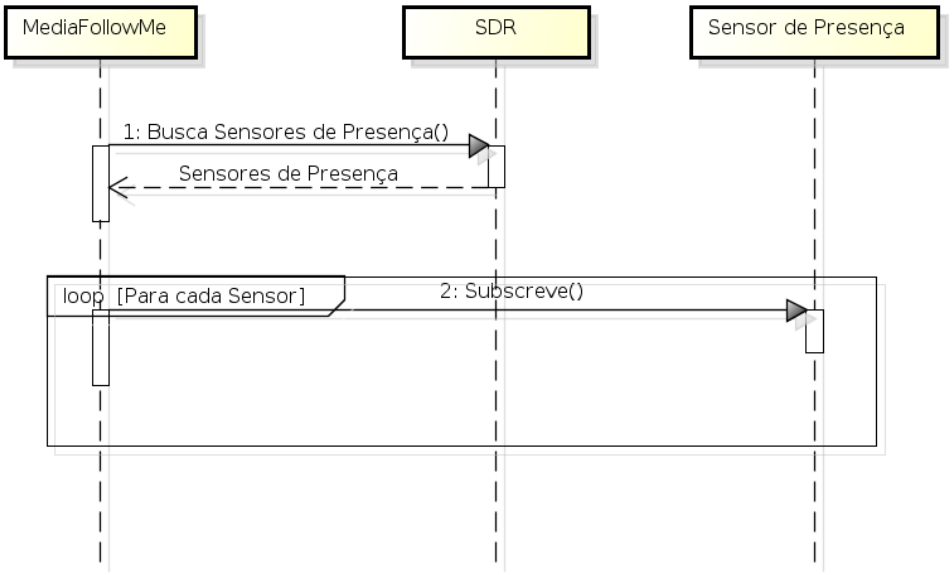


Figura 5.6: MediaFollowMe: Busca e Subscrição aos Sensores de Presença

Listagem 5.2: Obtendo as referências e registrando interesse nos eventos

```
1 ResourceAgent[] pSensors = sdr.searchByType("PresenceSensor");
2
3 for (sensor : pSensors) {
4   sensor.registerStakeholder("IN", this);
5   sensor.registerStakeholder("OUT", this);
6 }
```

Após obter todas as referências que precisa, o MediaFollowMe se registra como interessado em receber eventos dos ARs dos sensores de presença do ambiente. Ilustramos esse processo da Figura 5.6 e do trecho de código da Listagem 5.2. Na linha **1** é feita uma busca por todos os sensores de presença do ambiente através do SDR. Para cada sensor obtido, o MediaFollowMe se registra como interessado em eventos de entrada (linha **4**) e saída (linha **5**) de usuários nos cômodos onde se encontram os sensores.

No momento em que o usuário deixar um cômodo, o sensor de presença correspondente detectará este fato e notificará através de um evento todos os interessados nessa informação, incluindo o MediaFollowMe. Os eventos devem informar quem é o recurso que o disparou (neste caso, o sensor de presença do cômodo onde o usuário estava), qual o contexto alterado (entrada ou saída do cômodo) e qual o valor alterado (qual usuário se locomoveu). Assim, a aplicação receberá a referência do AR do usuário que saiu do cômodo em questão, e pausará a mídia proveniente do *Blu-ray* da sala, devido à ausência do indivíduo.

Semelhantemente à saída de um cômodo, um evento também é disparado na entrada

do usuário. Quando o usuário adentra um outro cômodo, o sensor de presença deste cômodo detecta a sua chegada e notifica à aplicação via eventos. A aplicação encontra o dispositivo mais próximo do usuário e realiza uma chamada remota para o *Blu-ray* (fonte dos dados) solicitando que ele mude o destino do *stream* para o novo dispositivo, e retoma a reprodução do filme (que estava em pausa).

Listagem 5.3: Tratamento dos eventos e consulta por dispositivo mais próximo

```
1  @Override
2  public void notificationHandler(ResourceAgent res, String context, Object
       obj) {
3    ...
4    Person p = (Person) obj;
5
6    if (p.getName().equals(USER_NAME) {
7      if (context.equals("OUT") {
8        bluray.pause();
9      } else if(context.equals("IN") {
10       Place current = slr.getPlaceOf(res);
11       ResourceAgent[] views = slr.searchByLocal(current,
12         "MediaExecutor");
13
14       if (views != null && views.lenght > 0) {
15         bluray.streamTo(views[0]);
16         bluray.play();
17       } else {
18         throw new MediaVisualizerNotFoundException(
19           "No media visualizer in " + current.getName());
20       }
21     }
22   }
23 }
```

Na Listagem 5.3 apresentamos o tratamento dos eventos recebidos pelo MediaFollowMe. O método de *callback* **notificationHandler** é definido na API do *framework* e deve ser implementado pelos objetos que desejam receber notificações de eventos. Este método passa como parâmetros o AR que gerou o evento, o contexto que foi alterado e o valor que foi alterado. No MediaFollowMe esses parâmetros são o AR do sensor de presença que gerou o evento, se o evento é de entrada (IN) ou saída (OUT), e qual usuário entrou ou saiu do cômodo.

A classe **Person** (linha **4**) é uma especialização de **ResourceAgent** (implementação de um AR), portanto, através de polimorfismo, pode ser passada como parâmetro na consulta ao SLR (linha **11**), em conformidade ao que foi apresentado na Seção 2.2.3. Esta consulta retorna a lista dos recursos do tipo **MediaExecutor** presentes no cômodo corrente.

Na linha **6** a aplicação testa se o usuário que entrou/saiu do cômodo em questão é o usuário requerido. Por fim, na linha **15** é feita uma chamada para o AR do *Blu-ray* solicitando que o *stream* da mídia seja repassado para o dispositivo de visualização mais próximo do usuário dentro do cômodo. Na linha **16**, é realizada uma chamada remota ao *Blu-ray* para reproduzir a mídia.

5.2.3 Visualização na IPGAP

Os primeiros passos para visualizar a aplicação em execução na IPGAP são semelhantes aos apresentados na Seção 5.1.3: deve-se adicionar os recursos simulados necessários à aplicação na IPGAP, através de seu menu principal. Será necessário adicionar sensores de presença em todos os cômodos, além de dispositivos que lidem com mídia para que a aplicação funcione apropriadamente. Caso não existam recursos de mídia disponíveis em outros cômodos, o aplicativo mostrará uma mensagem informando que não foi possível encontrar dispositivos. Para simular o cenário descrito, adicionamos na sala uma TV e um *Blu-ray* e no quarto um computador, além de sensores de presença em todos os cômodos.

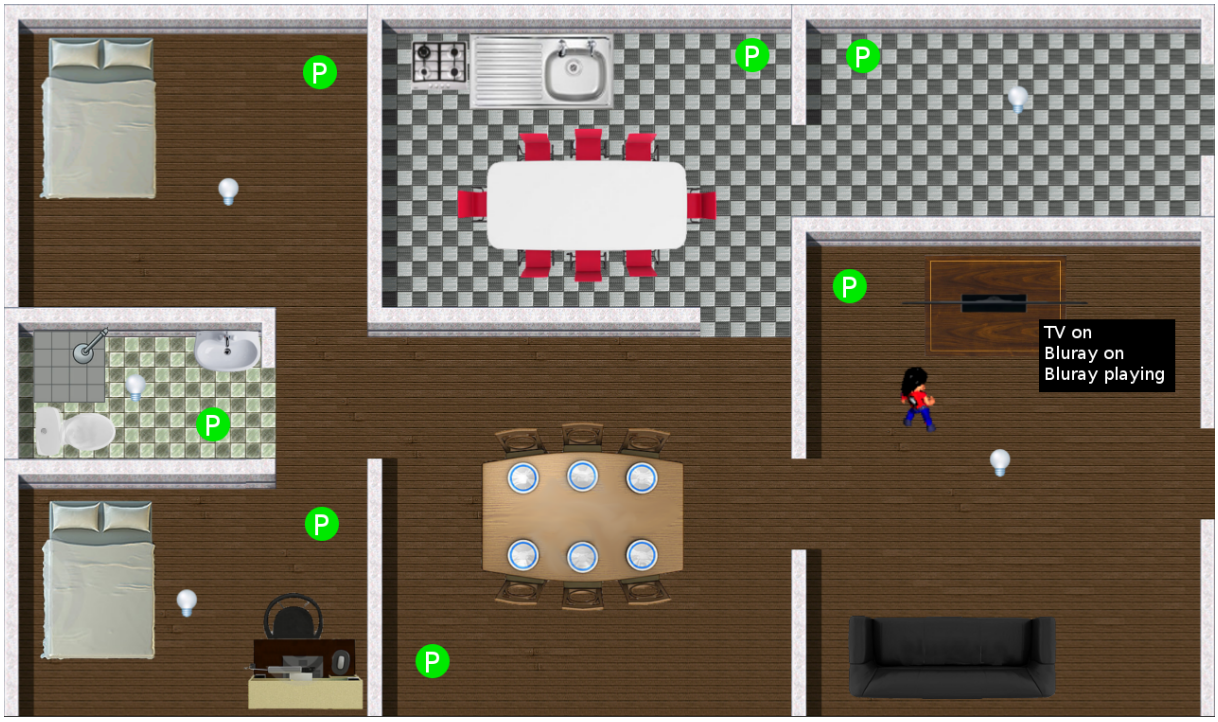
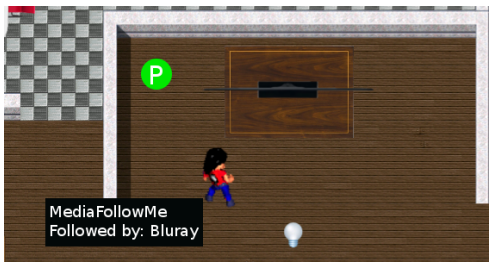
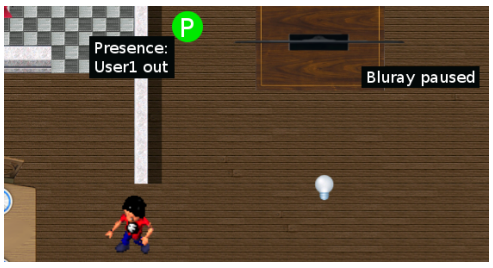


Figura 5.7: Mídia sendo executada no *Blu-ray* e TV da sala

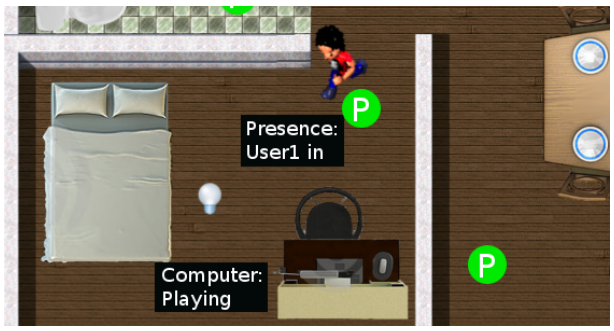
No início da simulação, movemos o avatar para a sala, onde se encontram a TV e o *Blu-ray*. Através dos aplicativos da TV e do *Blue-ray* na IPGAP (ver Seção 4.4), ligamos estes dispositivos. O computador do quarto é ligado de forma similar. Ainda através do aplicativo do *Blu-ray*, configuramos a saída do *Blu-ray* para ser encaminha para a entrada



(a) Seleção da fonte emissora da mídia



(b) Usuário saindo sala: Mídia é pausada



(c) Usuário entrando no quarto: Mídia é executada no computador

Figura 5.8: O MediaFollowMe em execução

da TV e executamos a Mídia. Na Figura 5.7 vemos a notificação da IPGAP sobre a mídia em execução.

Então, a aplicação MediaFollowMe é executada em um *smartphone* ou *tablet*, onde através de sua interface selecionamos o usuário e a fonte emissora da mídia. Essa seleção, conforme discutido anteriormente, é realizada através de consultas ao SDR que retornam as referências dos ARs correspondentes. Esse procedimento é ilustrado na Figura 5.8(a).

Ao movermos o avatar para fora da sala (Figura 5.8(b) através da IEM (Seção 4.5.2), um evento é disparado pelo sensor de presença da sala e o MediaFollowMe pausa a execução da mídia. Enquanto o avatar se desloca para a sala de jantar, os sensores de presença igualmente reconhecem a chegada do usuário no cômodo e lançam um evento, porém não existem dispositivos disponíveis nesse cômodo. Finalmente, conforme ilustrado na Figura 5.8(c), movemos o usuário para o quarto. Novamente o sensor de presença dispara um evento, e o MediaFollowMe desta vez encontra um dispositivo neste cômodo: o computador, para onde a mídia é migrada. A mídia então é executada novamente.

Capítulo 6

Trabalhos Relacionados

Podemos encontrar na literatura da área de Computação Ubíqua e Pervasiva outras propostas de ferramentas que visam a prototipagem de aplicações pervasivas (veja [78] para uma *survey*). Neste capítulo foram reunidos os artigos mais importantes do segmento, discutindo e comparando-os com a ferramenta proposta neste trabalho.

6.1 eHome Simulator

Em [5], é apresentado o **eHome Simulator**, criado com base na plataforma OSGi (*Open Service Gateway Initiative*)[1]. A proposta conta com uma visualização 2D do ambiente, onde seu plano de fundo é modelado através da ferramenta SketchUp[2]. Por meio de uma ferramenta própria de edição, construída em Java, é possível mapear o ambiente, definindo as partes bloqueadas por paredes, por móveis, etc. Todo o sistema segue o padrão descrito em [63], denominado *SCD-process* (*Specification Configuration and Deployment*) que permite o reuso de serviços baseados em componentes, construídos para o domínio de *smart homes*. Esse processo é implementado a partir de um módulo chamado **eHomeConfigurator**.

O trabalho traz à tona uma discussão sobre reutilização de serviços – terminologia que seria equivalente às *aplicações*, descritas em nossa proposta. Entretanto, não são fornecidos maiores detalhes sobre como é realizada a implementação desses serviços ou que padrão devem seguir. Apesar da utilização do OSGi para configuração, gerenciamento e reuso do ciclo de vida dos serviços, o trabalho não apresenta flexibilidade na criação de cenários de teste, com adição de dispositivos novos no sistema e descoberta de recursos

[1] OSGi Alliance – `http://www.osgi.org`
[2] `http://www.sketchup.com/`

em tempo de execução. Além disso não fica claro como é feita a inclusão de novos recursos no sistema e não há uma interface comum onde os dispositivos podem ser manipulados (alterado seu estado).

6.2 CASS

É proposto em [71], o **CASS** (*Context-Aware Simulation System*). O trabalho visa a geração de informações de contexto associadas a dispositivos simulados, e manipulados através de regras de contexto pré-fixadas. As regras podem ser criadas diretamente, seguindo o padrão definido através de arquivos no formato XML, ou através de uma GUI. Além disso, inclui-se no gerenciamento da regra a possibilidade de verificação e detecção de conflitos. O projeto também conta com uma interface para visualização da atuação das regras de contexto no ambiente. Sensores e atuadores simulados podem ser inseridos no ambiente, além de avatares para representar usuários, que possuem permissões diferenciadas de acordo com seus papéis, como por exemplo *convidados* e *membros da família*.

O trabalho tem como maior contribuição a definição, implementação e manipulação do contexto através de regras, por isso não oferece suporte a dispositivos reais, nem a possibilidade de inclusão de novos dispositivos no ambiente, impossibilitando assim, a criação de um mecanismo de descoberta de recursos no ambiente.

6.3 SHSim

Encontramos em [92] o **SHSim** (*Smart Home Simulator*), uma ferramenta também baseada em OSGi. A ferramenta oferece facilidades na configuração e adaptação do ambiente em tempo de execução. Dispositivos podem ser inseridos e removidos sem necessidade de alteração no código e a transparência no uso de recursos reais ou simulados é abordada, necessitando porém que todos os componentes estejam baseados em OSGi. Assim como a IPGAP, a ferramenta exibe o ambiente e seus componentes em uma visão do topo em 2D. O simulador conta com um *serviço de tempo*, que permite aumentar ou diminuir a velocidade da simulação.

Apesar de discutir a possibilidade de utilizar dispositivos reais, o trabalho não deixa claro os procedimentos necessários para incluir esse tipo de dispositivo nos testes. A IPGAP permite essa possibilidade conforme descrito na Seção 4.4.3. Além disso não é

descrito o uso de regras de contexto definidas pelo usuário.

6.4 CCAS

O simulador descrito em [41], o **CCAS** (*Configurable Context-Aware Simulator*), assim como as propostas vistas nas Seções 6.1 e6.3, também foi construído sobre a plataforma OSGi. Pessoas, dispositivos e o ambiente doméstico podem ser simulados e configurados através de arquivos no formato XML. Os dispositivos são abstraídos em termos de *bundles* – componentes do OSGi, que podem ser dinamicamente parados e iniciados sem afetar o sistema como um todo.

É mencionada a possibilidade de inserção de novos dispositivos, inclusive reais, dinamicamente no sistema, e a questão da transparência entre dispositivos simulados e reais é também é abordada e obtida através da generalização da interface dos dispositivos e da utilização de módulos padronizados (comparáveis aos ARs). O trabalho cita a possibilidade de descoberta de novos componentes dinamicamente, mas não traz maiores detalhes a cerca deste tema. Regras de contexto também não são citadas.

6.5 ISS

O **ISS** (*Interactive Smart Home Simulator*) é proposto em [80], o qual coleta informações de contexto do ambiente simulado – como tempo e clima – provendo ao usuário a possibilidade de customizar o ambiente, como por exemplo o posicionamento de sensores e dispositivos. O sistema, construído na plataforma .NET, utiliza um servidor de contexto, baseado em *Web Services* que armazena as informações de contexto. Cada aplicação deve se conectar ao servidor para inserir e coletar dados de contexto, o que centraliza as informações de contexto.

Informações mais específicas como os dispositivos presentes no ambiente estão *hardcoded* no sistema e a questão da configuração e gerência do contexto do ambiente é estática, dificultando que a proposta seja estendida. Não são mencionados a integração com dispositivos reais, nem a possibilidade de inclusão de novos dispositivos no sistema, mesmo que simulados. Além disso, regras de contexto também não são mencionadas.

6.6 DiaSim

Em [17, 18] é apresentado o **DiaSim**, um simulador para aplicações pervasivas, parametrizado por diretivas escritas em uma linguagem de configuração própria [22]. A proposta conta com um módulo que gera classes abstratas em código Java a partir das diretivas de entrada, para que o desenvolvedor as estenda e implemente a lógica de seus dispositivos. As diretivas citadas também parametrizam o *design* do ambiente. O DiaSim exibe uma visualização 2D do topo, contendo os dispositivos do ambiente e avatares para simulação do ambiente residencial, renderizados com o auxílio do Siafu [59]. É citada a geração de estímulos para os dispositivos simulados através de uma abordagem probabilística.

O trabalho discute, mas não deixa claro como dispositivos reais são suportados, nem como realizar a substituição transparente entre dispositivos simulados e reais. Da mesma forma a questão da descoberta e registro de recursos. Não são mencionados meios para manipulação dos dispositivos por meio de aplicações ou regras de contexto, apenas através dos estímulos gerados, que alteram o estado do próprio dispositivo gerando eventos.

6.7 UbiWise

UbiWise [11, 12] é um projeto da **Hewlett-Packard (HP) Laboratories** e um dos simuladores para sistemas ubíquos/pervasivos pioneiros. Tem como base o **Wise** [83] (*Wireless Infrastructure Simulation*), um *toolkit* utilizado para auxiliar os pesquisadores no desenvolvimento de protótipos de novos dispositivos, através da simulação. O Wise continha a visão 2D da interface do dispositivo, e simulava aspectos como protocolos utilizados e conectividade, permitindo o pesquisador testar um dispositivo via *software*.

O UbiWise nasceu da combinação do Wise com o **UbiSim**, que tem por objetivo a simulação do ambiente em uma visualização 3D em primeira pessoa, construída através da *engine* do famoso jogo *Quake III Arena*. Com isso, o projetista tem em mãos duas ferramentas: o Wise, para interação com os dispositivos e o UbiSim, para interação com o ambiente. Dessa forma pretende-se simular a interação espontânea de um usuário, o mais próximo possível do mundo real.

O projeto discute a integração com dispositivos reais, citando uma aplicação com duas câmeras digitais – uma simulada e outra real – onde uma foto tirada em qualquer dessas câmeras pudesse ser visualizada na outra. Outra menção a esse aspecto é feita na descrição de uma aplicação de *sala de conferência*, onde foram utilizados dois *laptops* reais e sala de

conferência simulada. No entanto o trabalho não fornece maiores detalhes de como é feita a comunicação, nem descreve a possibilidade de inclusão de novos dispositivos ao sistema. Fica claro que simulação dos dispositivos não é transparente e além disso, a questão da descoberta e registro de recursos não é abordada, fazendo com que não seja facilmente estendido. O trabalho também não utiliza regras de contexto em sua abordagem.

6.8 TATUS

Em [66,67] é descrito o **TATUS**. Baseado na *engine* do jogo *Half-life*, a ferramenta também conta com uma visualização 3D e uma simulação realística do contexto do ambiente. O trabalho se assemelha ao UbiWise na maioria dos aspectos, porém adicionando um maior realismo nas simulações. Eventos são disparados a medida que o avatar do jogo se movimenta e altera o contexto do ambiente. Ademais, novos sensores simulados podem ser adicionados via SDK, através de arquivos de configuração no formato XML. O ambiente 3D é separado do modelo dos cenários de testes, implementados em Java, e se comunicam via troca de mensagens. Isso possibilita que diversos cenários ocorram no mesmo ambiente ao mesmo tempo, trazendo mais realidade aos testes.

O projeto cita a utilização de regras, mas não dá maiores detalhes sobre como foi é utilizado no sistema, nem como foi implementado. Além disso, não cita a possibilidade de integração com dispositivos reais. Não fica claro se a manipulação dos dispositivos não pode ser feita diretamente, e se seu estado pode ser visualizado na ferramenta.

6.9 UbiREAL

Em [62] encontramos o **UbiREAL** (*Ubiquitous Application Simulator with REAListic Environments*), que como sugere seu nome, provê uma simulação realística de aspectos de comunicação, contando com um módulo simulador de redes, além de temperatura, iluminação e sonoridade, entre outros, utilizando uma modelagem matemática – o que torna os resultados mais confiáveis e precisos. O trabalho cita a possibilidade de troca de dispositivos simulados por reais, necessitando apenas de pequenas modificações em seus *drivers*. Ademais, a ferramenta utiliza duas opções de visualização do ambiente: uma visão 3D em primeira pessoa, e uma visão do topo em 2D.

São utilizadas no projeto regras de contexto, definidas através de uma modelagem formal incluindo detecção de inconsistência e testes de exatidão (*correctness*), porém não

há uma interface para as compor em tempo de execução. Além disso, o projeto não aborda a questão de desenvolvimento e testes de aplicações pervasivas que utilizem os dispositivos e o ambiente, o que só pode ser feito através das regras. Não fica claro também se é possível adicionar novos dispositivos no ambiente simulado além dos pré-definidos, nem são citados aspectos relativos à descoberta de recursos no ambiente. Em geral, este trabalho não tem como foco o usuário final atuando em seu ambiente residencial e gerenciando seus dispositivos.

6.10 Discussão e Conclusões

Neste capítulo os trabalhos da literatura na área de simulação e prototipagem de aplicações ubíquas/pervasivas foram discutidos e comparados com a IPGAP, com o objetivo de posicionar o presente projeto no estado da arte. Os trabalhos foram comparados de acordo com as seguintes características:

- **Simulação de dispositivos:** a proposta simula os dispositivos e suas interações, de forma a testá-los antes mesmo de que exista o dispositivo físico. Essa característica auxilia o desenvolvedor a encontrar erros de evoluir requisitos mais cedo. Isso também permite diminuição nos custos do projeto, pois não exige ter todos os dispositivos necessários aos testes, podendo-se usar suas versões simuladas;

- **Suporte a dispositivos reais:** a proposta permite a utilização de dispositivos reais, tornando os testes da aplicação mais precisos e realistas, pois a comunicação e interação com este tipo de dispositivo é está sujeita a um maior número de variáveis;

- **Transparência na utilização de dispositivos:** a proposta permite a que um dispositivo simulado seja trocado por um dispositivo real transparentemente, ou seja, o desenvolvedor não necessita realizar alterações na interface de suas chamadas ao realizar essa troca;

- **Registro/descoberta de recursos do ambiente:** a proposta contém um mecanismo que registra e descobre dispositivos e outros recursos no ambiente. Essa funcionalidade permite que novos dispositivos entrem no sistema mais facilmente, o que é importante principalmente quando a ferramenta de prototipagem é utilizada pelo usuário final;

- **Manipulação do contexto dos dispositivos:** a proposta permite a manipulação dos recursos remotamente, ou seja, a alteração do estado dos dispositivos por meio

de chamadas remotas. Dessa forma é possível que os dispositivos se comuniquem e alterem o contexto do ambiente por meio de regras de contexto, aplicações ou mesmo diretamente pelo usuário;

- **Utilização de regras de contexto:** a proposta contém um mecanismo que utiliza regras de contexto que são continuamente monitoradas para manipular o ambiente de acordo com seu estado atual. A criação dessas regras é útil para acelerar o desenvolvimento e para manipulação do ambiente por parte do usuário final;
- **Simulação realista do ambiente:** a proposta conta com uma simulação realista do ambiente, no que diz respeito ao contexto externo aos dispositivos como espaço, tempo ou clima, entre outros aspectos. Isso permite que mais acurácia nos testes, pois os dispositivos e aplicações utilizariam um ambiente mais próximo da realidade;

Estes itens foram avaliados em cada trabalho analisado, e expostos na Tabela 6.1 com três possibilidades:

(i) **S** – significando que o trabalho possui esta característica;

(ii) **N** – significando que o trabalho não possui esta característica, ou não deixa claro que possui;

(iii) **D** – significando que o trabalho discute o uso dessa característica, mas não traz maiores detalhes de projeto ou não deixa claro como a utiliza, como a implementa ou implementaria;

Podemos notar que a maioria das propostas utiliza a simulação mais voltada para testes nos dispositivos, incluindo algumas vezes a possibilidade de utilizar dispositivos reais, até mesmo de forma transparente. Essa abordagem é importante para o desenvolvimento de dispositivos para prover a computação ubíqua, porém deixa de fora abordagens relacionadas ao desenvolvimento de *softwares* para computação ubíqua. Por exemplo, poderíamos ter uma empresa interessada em desenvolver uma aplicação que, em ocorrência de viagem de todos os moradores, controle os dispositivos da casa de tal forma que simule a presença de pessoas na casa, com o objetivo de aumentar a segurança. A empresa não deseja vender dispositivos, que aliás, devem ser os mesmos que já existem na casa (considerando que há o suporte necessário, como o SmartAndroid) – deseja-se apenas vender o *software* que realiza essas atividades. Como a empresa poderia testar essa aplicação?

A IPGAP, juntamente com o *framework* SmartAndroid, permite que seja desenvolvido uma aplicação desse tipo em seu ambiente. A utilizando dispositivos simulados em um primeiro momento farão como que a empresa comece a desenvolver a aplicação antes de adquirir os dispositivos que serão controlados, permitindo a evolução do produto mais rapidamente. Após essa etapa podem ser adicionados dispositivos reais para que a aplicação seja testada em uma situação mais realista. Note que o negócio neste caso é aplicação, não os dispositivos, ou seja, a aplicação é que está sendo testada, não os dispositivos. Logo, essa abordagem faz com que nossa proposta seja mais geral, no que diz respeito à desenvolvimento de aplicações ubíquas/pervasivas.

Por último podemos destacar que uma característica singular da IPGAP, é a utilização de uma abordagem centrada no usuário. Alguns trabalhos como [49] e [75], também utilizam essa abordagem, que provê ao usuário comum a habilidade de configurar aplicações. Para atingir esse objetivo é necessário que a ferramenta de desenvolvimento forneça uma interface simples e intuitiva. A IPGAP consegue atingir esse objetivo através de ícones que representam os dispositivos e serviços. Outros trabalhos também exploram esse paradigma de ícones, porém não fornecem um meio simples de se conectar aos diversos elementos do sistema. Os ícones da IPGAP representam *aplicativos* (ver Seção 4.4) que facilitam a configuração dos recursos. Além disso a composição de regras de contexto pelo usuário final fornece ao usuário a possibilidade de programar o ambiente de acordo com suas necessidades, sem introduzir maiores complicações (Seção 4.5.3).

Tabela 6.1: Resumo da comparação com a IPGAP

Ferramenta	Características							
	Dispositivos simulados	**Dispositivos reais**	**Adição de recursos**	**Simulação transp.**	**Descoberta de recursos**	**Manipul. de recursos**	**Regras de contexto**	**Simulação realística**
eHome Simulator	S	D	D	N	N	N	N	N
CASS	S	N	N	N	N	S	S	N
SHSim	S	D	S	S	N	N	N	N
CCAS	S	S	S	S	D	N	N	N
ISS	S	N	N	N	N	S	N	D
DiaSim	S	D	D	D	S	S	N	S
UbiWise	S	D	N	N	N	S	N	S
UbiREAL	S	S	N	D	N	N	S	S
TATUS	S	N	S	N	N	N	D	S
IPGAP	S	S	S	S	S	S	S	N

Capítulo 7

Conclusões e Trabalhos Futuros

Neste capítulo serão discutidas as conclusões gerais obtidas através desta proposta, bem como um resumo das principais contribuições e limitações deste trabalho. Na Seção 7.2, especificamente serão apresentados os trabalhos futuros.

7.1 Conclusões

Neste trabalho foi apresentada a concepção de uma proposta de interface para prototipagem e gerenciamento de aplicações pervasivas. A IPGAP foi concebida a partir das ideias provenientes do projeto **SCIADS**[1] (Sistema Computacional Inteligente de Assistência Domiciliar à Saúde) [21, 56], desenvolvido no Laboratório Tempo do Instituto de Computação da Universidade Federal Fluminense em parceria com pesquisadores do Instituto de Matemática e Estatística da Universidade do Estado do Rio de Janeiro. O monitoramento remoto de pacientes desenvolvido no SCIADS trouxe à discussão no grupo o tema de ambientes inteligentes, dando origem ao projeto **SmartAndroid**[2].

Criado com o objetivo de prover um *framework* para auxiliar o desenvolvedor na construção de aplicações específicas para ambientes inteligentes, o projeto SmartAndroid engloba diversos domínios de atuação da computação ubíqua/pervasiva, como saúde, segurança, conforto e entretenimento. Inclui-se no suporte ao desenvolvedor o registro/descoberta automática de dispositivos e recursos do ambiente, além de primitivas de comunicação remota síncrona e assíncrona (Capítulo 2).

Ao longo das reuniões e conversas informais dos integrantes do grupo sobre o suporte fornecido pelo SmartAndroid, veio à tona a ideia de se ter uma ferramenta que facilitasse os

[1] `http://www.tempo.uff.br/sciads`
[2] `http://www.tempo.uff.br/smartandroid`

testes em aplicações ubíquas/pervasivas – que conforme discutido no Capítulo 1, são uma gama de aplicações que demandam grande esforço e recursos financeiros para realização dos testes. Através da IPGAP, o desenvolvedor poderá testar sua aplicação pervasiva através da criação de protótipos funcionais, fornecendo as ferramentas necessárias para que esse processo seja realizado em menos tempo e à baixo custo. Destaca-se na IPGAP a utilização de simuladores e dispositivos reais, e a API de gerenciamento do ambiente provida pelo *framework* SmartAndroid para atingir os objetivos mencionados.

A proposta desta ferramenta é possibilitar uma transição graciosa entre os mundos virtuais e reais, criando um ambiente híbrido onde ambos dispositivos reais e simulados são intercambiáveis e se comunicam entre si. Além disso, o uso de aplicativos (Seção 4.4) que representam e gerenciam os recursos do ambiente (como fogão e TV) fazem com que os respectivos recursos possam ser manipulados mais facilmente, através de uma interface simples. Aplicativos novos também poderão ser instalados na IPGAP, permitindo que a ferramenta seja estendida de acordo com as necessidades do desenvolvedor, sem necessidade de alteração no código. Assim, uma infinidade de cenários podem ser criados, tornando os testes em aplicações pervasivas mais confiáveis e próximos da realidade. O desenvolvedor que utilizar a IPGAP para testes de suas aplicações também terá a sua disposição uma ferramenta para manipulação de avatares no mapa do ambiente para simular, por exemplo, informações de localização de pessoas.

A IPGAP poderá ser utilizada também pelo usuário final (Seção 4.2.2), que terá a possibilidade de manipular os dispositivos presentes em seu ambiente inteligente através da interface dos aplicativos de cada recurso, onde também será possível observar o estado de cada um deles, pois os eventos relativos aos recursos são propagados pelo *framework*, notificando a interface do aplicativo. Além disso, o usuário tem a disposição uma interface para criação de regras de contexto. Dessa forma este poderá configurar o ambiente em alto nível (*end-user programming*).

Entre as principais contribuições deste trabalho está a utilização de uma abordagem onde os recursos são descobertos no ambiente de forma autônoma, permitindo assim a inclusão de novos recursos sem que para isso tenha-se que recompilar o código da IPGAP ou mesmo interromper seu funcionamento. Além disso, um diferencial desta proposta é o foco também no usuário final, o qual poderá, através da IPGAP, controlar o ambiente inteligente em que se encontra e criar regras de contexto em alto nível.

7.2 Trabalhos Futuros

Ao longo do trabalho desenvolvido nesta dissertação, identificou-se algumas das possíveis etapas a serem realizadas no projeto com o objetivo de sugerir possíveis novas propostas de projetos. Na Seção 7.2.1 discutiremos os trabalhos futuros relacionados à IPGAP, e na Seção 7.2.2 uma discussão sobre as possibilidades futuras no que diz respeito ao *framework* em geral.

7.2.1 IPGAP

7.2.1.1 Visualização do Ambiente

Muitos cenários exigem uma simulação mais realística do ambiente para que sejam adequadamente testados, principalmente os que lidam com variáveis de contexto como umidade, temperatura e luminosidade. Como exemplo, temos o trabalho proposto em [62] (discutido no Capítulo 6), que traz à tona esses requisitos e propõe uma implementação.

Por isso, a implementação de uma visualização 3D do ambiente inteligente na interface de prototipagem também é um dos tópicos futuros, pois proporciona uma maior flexibilidade à proposta, que passa a dar suporte a uma maior gama de aplicações. Aliar uma melhor apresentação do ambiente às funcionalidades já providas pela IPGAP angariaria em uma melhor experiência para desenvolvedores na execução dos testes. Além disso, os usuários finais teriam disponível uma interface na qual pudessem interagir mais naturalmente.

7.2.1.2 Simulação Realística

A simulação realística de dados físicos como temperatura, umidade, tempo, entre outros, é um aspecto relevante para testes em aplicações que lidam com dados físicos. Como um trabalho futuro, os simuladores dos dispositivos (Seção 4.4.1) poderiam incluir um modelo físico que permitisse essa funcionalidade.

Como os simuladores são componentes externos à IPGAP, apenas eles seriam alterados, sem a necessidade de esforço na integração com a IPGAP.

7.2.1.3 Mapa do Ambiente

Um passo importante a ser tomado com relação à representação do ambiente na IPGAP é a criação do mapa de forma automática. Através da utilização de uma abordagem semelhante à apresentada em [55], pode-se detectar os cômodos do ambiente automaticamente e gerar a estrutura de dados requirida pela IPGAP.

A automatização desse processo simplificaria a configuração de um novo ambiente inteligente, além de torná-la menos suscetível a erros.

7.2.1.4 Emulação de Movimento

A Interface de Emulação de Movimento IEM (Seção 4.5.2) pode ser estendida, oferecendo ao desenvolvedor um *script* para definição do trajeto pré-definido para dispositivos móveis simulados. O *script* conteria diretivas informando a direção para onde os dispositivos devem ir e quanto tempo essa movimentação deve durar. Além disso seria possível a definição de pausas durante o percurso e do momento no tempo em que a movimentação deve ocorrer.

Através desse *script* poderiam ser disparadas ordens de movimentação em batelada para diversos dispositivos distintos, incluindo a relação temporal entre esses eventos. Isso facilitaria o processo de teste da aplicação, onde seria possível, por exemplo, a execução sequencial de testes automatizados e a repetição dos testes pré-definidos facilmente, uma espécie de *replay*.

Uma outra característica a ser implementada seria um módulo ciente dos obstáculos do ambiente. Isso permitiria a criação de movimentos randômicos dentro do mapa automaticamente, sem que os dispositivos ou avatares "esbarrem" em paredes e móveis. O mapa do ambiente já contém a definição de algumas partes "intransponíveis" como paredes, representadas no arquivo TMX (ver Apêndice A).

7.2.1.5 Composição de Regras de Contexto

A criação de regras de contexto na IPGAP (Seção 4.5.3) conta com as funcionalidades básicas que uma regra pode possuir, incluindo os operadores lógicos e relacionais simples. Essas funcionalidades são acionadas através de botões na tela da IPGAP. Um próximo passo a ser seguido nesse sentido, seria a utilização de um modelo de interação mais intuitivo na composição de regras. Em [49, 60] vemos exemplos de alguns trabalhos que

seguem essa abordagem.

Em vez do texto da regra ser apresentado na tela a medida que o usuário a estiver compondo, esta seria apresentada por meio de blocos gráficos representando as condições e ações. Como exemplo, em [49] vemos uma abordagem onde é utilizada a metáfora de um "quebra-cabeça" para realizar a composição das regras. Além disso, a combinação entre uma abordagem mais intuitiva com a utilização de reconhecimento de gestos traria para o usuário final diminuiria consideravelmente a curva de aprendizado da ferramenta, além da adesão à sua utilização.

7.2.1.6 Registro e Gerência de Configuração do Ambiente

Conforme discutido na Seção 4.2.2 a IPGAP pode ser utilizada como uma ferramenta de gerenciamento do ambiente pelo usuário final. Uma utilização interessante da IPGAP seria na visualização de "fotografias" do ambiente em um dado momento. Isso poderia ser feito através de um *parser* no log de eventos da IPGAP. Caso a quantidade de informações seja muito grande para o processamento desta funcionalidade, poderia-se utilizar um subconjunto menor de informações – as mais importantes para o usuário – de forma diminuir o tempo de processamento.

A possibilidade de visualização de estados anteriores do ambiente seria de utilidade para, por exemplo, identificar o que mudou no ambiente, através da realização de um *diff* entre estados do ambiente em momentos distintos, para fins de monitoramento e auditoria. Esta comparação poderia ser feita em diversos níveis de aproximação, para evitar mostrar diferenças entre, por exemplo, uma temperatura de 25.2 e outra de 25.3 graus Celsius.

O desenvolvedor também poderia se beneficiar com essa funcionalidade, no que diz respeito à configuração do ambiente de teste. A comparação do estado do ambiente em diversos cenários de teste pode ser útil para encontrar inconsistências no sistema, e identificar configurações não previstas pela aplicação.

7.2.2 *Framework* de Desenvolvimento de Aplicações

7.2.2.1 Monitoramento de Pacientes

A utilização do *framework* para prover um ambiente de monitoramento para idosos e portadores de doenças crônicas é factível e vislumbrada como um dos próximos passos do projeto. Através da integração com o SCIADS – que conforme discutido na Seção 7.1

possui as funcionalidades mencionadas – as possibilidades para os pacientes podem ser expandidas por exemplo, integrando seu Plano de Cuidados a outros dispositivos da casa.

A utilização de dispositivos aderentes ao padrão **Continua**[3], discutida em [31], torna a integração do SmartAndroid e SCIADS natural, pois estes são nativamente compatíveis com a tecnologia Android. A **Continua Health Alliance** é uma aliança formada por empresas da área médica e de tecnologia que foi criada com a finalidade de incentivar a integração de sistemas. A partir de sua versão 4.0, o Sistema Operacional Android passou a incluir o *Bluetooth Health Device Profile* (HDP) que suporta dispositivos certificados pela Continua, como monitores de frequência cardíaca, termômetros e balanças.

7.2.2.2 Redes Sociais

A integração do projeto com as Redes Sociais (RS) traria benefícios interessantes em vários aspectos. Em [61] é descrita uma RS dedicada à saúde, com foco em pacientes com insuficiência cardíaca chamada **Minha Saúde**[4]. Através da *Minha Saúde* os usuários têm à disposição informações e notícias sobre saúde, além da possibilidade de manterem contato e trocarem experiências com outros pacientes.

A *Minha Saúde* engloba o acompanhamento de prescrições médicas online, permitindo os pacientes visualizarem rapidamente seu histórico e evolução através de gráficos, com o intuito de aumentar a adesão ao tratamento. A proposta conta também com um sistema de recomendação de amizade, utilizando técnicas de processamento simbólico. A integração com o SmartAndroid possibilitaria a coleta automática de dados fisiológicos do paciente e de outros dados de contexto, possibilitando uma melhor interação com os usuários.

Outras aplicações de RS em ambientes inteligentes podem ser desenvolvidas, como compartilhamento de dados do ambiente com amigos e familiares. Um exemplo são as informações de consumo de energia de uma residência, que poderiam ser compartilhadas em uma RS na forma de um jogo, onde amigos e familiares competiriam para julgar quem gasta menos energia. Trabalhos semelhantes [9, 10, 46, 47] já existem na literatura, porém com um enfoque um pouco diferente. Além de prover entretenimento aos usuários, esse tipo aplicação tem a função de conscientização sobre consumo responsável de energia. A coleta das informações de energia do ambiente seria coletado através do SmartAndroid, e uma aplicação se encarregaria do compartilhamento das informações na RS, além da

[3] `http://www.continuaalliance.org/`
[4] `http://www.minhasaude.org`

contabilização dos resultados.

7.2.2.3 Linhas de Produto de *Software*

Vislumbra-se também, como trabalho futuro, a utilização dos conceitos de Linha de Produto de *Software* [64] no processo de desenvolvimento de aplicações pervasivas. Através do meta-modelo descrito em [27] pode-se descrever a configuração das características de cada ambiente inteligente e suas arquiteturas, possibilitando a criação de subsistemas. Por exemplo, poderia ser criado um subsistema de segurança envolvendo portas, janelas, alarmes e câmeras. Além disso, o uso do meta-modelo possibilitaria a adaptação dinâmica do ambiente inteligente aos requisitos oriundos do usuário final.

Referências

[1] Abowd, G. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering - ICSE '99* (New York, New York, USA, 1999), ACM Press, pp. 75–84.

[2] Abowd, G.; Dey, A.; Brown, P.; Davies, N.; Smith, M.; Steggles, P. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing*, H.-W. Gellersen, Ed., vol. 1707 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pp. 304–307.

[3] Abowd, G.; Mynatt, E.; Rodden, T. The human experience [of ubiquitous computing]. *IEEE Pervasive Computing 1*, 1 (Jan. 2002), 48–57.

[4] Ark, W.; Selker, T. A look at human interaction with pervasive computers. *IBM Systems Journal 38*, 4 (1999), 504–507.

[5] Armac, I.; Retkowitz, D. Simulation of Smart Environments. In *IEEE International Conference on Pervasive Services* (2007), IEEE, pp. 257–266.

[6] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. A view of cloud computing. *Commun. ACM 53*, 4 (Apr. 2010), 50–58.

[7] Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Computer Networks 54*, 15 (Oct. 2010), 2787–2805.

[8] Augusto, J.; McCullagh, P. Ambient intelligence: Concepts and applications. *Computer Science and Information Systems/ComSIS 4*, 1 (2007), 1–26.

[9] Bang, M.; Gustafsson, A.; Katzeff, C. Promoting new patterns in household energy consumption with pervasive learning games. In *Persuasive Technology*, Y. Kort, W. IJsselsteijn, C. Midden, B. Eggen, and B. Fogg, Eds., vol. 4744 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 55–63.

[10] Bang, M.; Torstensson, C.; Katzeff, C. The powerhhouse: A persuasive computer game designed to raise awareness of domestic energy consumption. In *Persuasive Technology*, W. IJsselsteijn, Y. Kort, C. Midden, B. Eggen, and E. Hoven, Eds., vol. 3962 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 123–132.

[11] Barton, J. J.; Vijayaraghavan, V. Ubiwise, a ubiquitous wireless infrastructure simulation environment. Tech. Rep. HPL-2002-303, Hewlett-Packard Laboratories, Palo Alto, 2002.

[12] Barton, J. J.; Vijayaraghavan, V. UBIWISE, a simulator for ubiquitous computing systems design. Tech. Rep. HPL-2003-93, Hewlett-Packard Laboratories, Palo Alto, 2003.

[13] Bellotti, V.; Edwards, K. Intelligibility and accountability: human considerations in context-aware systems. *Human-Computer Interactions 16*, 2 (Dec. 2001), 193–212.

[14] Bezerra, L. N. *Uso de ontologia em serviço de contexto e descoberta de recursos para autoadaptação de sistemas.* Dissertação de mestrado, Programa de Pós Graduação em Engenharia Eletrônica (PEL) – Universidade do Estado do Rio de Janeiro, 2011.

[15] Birrell, A. D.; Nelson, B. J. Implementing remote procedure calls. *ACM Transactions on Computer Systems 2*, 1 (Feb. 1984), 39–59.

[16] Bränzel, A.; Holz, C.; Hoffmann, D.; Schmidt, D.; Knaust, M.; Lühne, P.; Meusel, R.; Richter, S.; Baudisch, P. Gravityspace: tracking users and their poses in a smart room using a pressure-sensing floor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 725–734.

[17] Bruneau, J.; Consel, C. Diasim: a simulator for pervasive computing applications. *Software: Practice and Experience* (2012).

[18] Bruneau, J.; Jouve, W.; Consel, C. DiaSim: A parameterized simulator for pervasive computing applications. In *Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* (2009), IEEE.

[19] Brush, A. B.; Lee, B.; Mahajan, R.; Agarwal, S.; Saroiu, S.; Dixon, C. Home automation in the wild. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11* (New York, New York, USA, 2011), ACM Press, p. 2115.

[20] Cardoso, L. X. T. *Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software.* Dissertação de mestrado, Instituto de Computação – Universidade Federal Fluminense, 2006.

[21] Carvalho, S. T.; Erthal, M.; Mareli, D.; Sztajnberg, A.; Copetti, A.; Loques, O. Monitoramento Remoto de Pacientes em Ambiente Domiciliar. In *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC 2010* (Gramado, RS, Brasil, maio 2010), pp. 1005–1012.

[22] Cassou, D.; Bruneau, J.; Consel, C.; Balland, E. Toward a tool-based development methodology for pervasive computing applications. *Software Engineering, IEEE Transactions on 38*, 6 (2012), 1445–1463.

[23] Chen, Y.; Au, J.; Kazlas, P.; Ritenour, A.; Gates, H.; McCreary, M. Electronic paper: Flexible active-matrix electronic ink display. *Nature 423*, 6936 (2003), 136–136.

[24] Coelho, A. R. *Comunicação Orientada a Interesse em um Contexto de Computação Ubíqua e Pervasiva.* Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense, 2013.

[25] Copetti, A.; Leite, J.; Loques, O.; Neves, M. A decision-making mechanism for context inference in pervasive healthcare environments. *Decision Support Systems* (2012).

[26] Davies, N.; Landay, J.; Hudson, S.; Schmidt, A. Guest Editors' Introduction: Rapid Prototyping for Ubiquitous Computing. *IEEE Pervasive Computing 4*, 4 (Oct. 2005), 15–17.

[27] de Carvalho, S. T. *Modelagem de Linhas de Produto Dinâmicas para Aplicações Ubíquas.* Tese de doutorado, Instituto de Computação – Universidade Federal Fluminense, 2013.

[28] de Moraes, H. F.; Dutra, R. C.; Amorim, C. L. Repi: Um protocolo peer-to-peer para aplicações orientadas a interesses na internet. In *VIII Workshop de Redes Dinâmicas e Sistemas P2P (WP2P) – XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (Ouro Preto, MG, Brasil, maio 2012), pp. 60–73.

[29] Dey, A.; Abowd, G.; Salber, D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction 16*, 2 (Dec. 2001), 97–166.

[30] Dey, A. K. Understanding and Using Context. *Personal and Ubiquitous Computing 5*, 1 (2001), 4–7.

[31] Diogo, T. *Integração de Informações Médicas de Sensores: do Paciente à Nuvem.* Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense, 2013.

[32] Dow, S.; MacIntyre, B.; Lee, J.; Oezbek, C.; Bolter, J.; Gandy, M. Wizard of Oz Support throughout an Iterative Design Process. *IEEE Pervasive Computing 4*, 4 (Oct. 2005), 18–26.

[33] Dutra, R. C.; Granja, R. S.; Moraes, H. F.; Amorim, C. L. Repi: Rede de comunicação endereçada por interesses. In *VI Workshop de Redes Dinâmicas e Sistemas P2P (WP2P) – XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (Gramado, RS, Brasil, maio 2010), pp. 99–112.

[34] Edwards, W. At home with ubiquitous computing: Seven challenges. *Ubicomp 2001: Ubiquitous Computing* (2001), 256–272.

[35] Elrad, T.; Filman, R. E.; Bader, A. Aspect-oriented programming: Introduction. *Commun. ACM 44*, 10 (Oct. 2001), 29–32.

[36] Erthal, M. *Interagindo com o ambiente: um Framework para construção de aplicações sensíveis ao contexto.* Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense, 2013.

[37] Erthal, M.; Mareli, D.; Ferreira, D. B.; Loques, O. Interpretação de contexto em ambientes inteligentes. In *V Simpósio Brasileiro de Computação Ubíqua e Pervasiva – SBCUP 2013* (Maceió, AL, Brasil, julho 2013).

[38] Eugster, P.; Felber, P.; Guerraoui, R.; Kermarrec, A. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR) 35*, 2 (2003), 114–131.

[39] Ferreira, D. B.; Erthal, M.; Mareli, D.; Loques, O. Uma interface de prototipagem para aplicações pervasivas. In *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (Brasília, DF, Brasil, maio 2013), pp. 629–642.

[40] Fortin-Simard, D.; Bouchard, K.; Gaboury, S.; Bouchard, B.; Bouzouane, A. Accurate passive rfid localization system for smart homes. In *Proceedings of the 2012 IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA)* (Washington, DC, USA, 2012), NESEA '12, IEEE Computer Society, pp. 1–8.

[41] Fu, Q.; Li, P.; Chen, C.; Qi, L.; Lu, Y.; Yu, C. A configurable context-aware simulator for smart home systems. In *2011 6th International Conference on Pervasive Computing and Applications* (2011), IEEE, pp. 39–44.

[42] Garcia-Molina, H.; Germano, F.; Kohler, W. H. Debugging a distributed computing system. *Software Engineering, IEEE Transactions on SE-10*, 2 (1984), 210–219.

[43] Gregoski, M. J.; Mueller, M.; Vertegel, A.; Shaporev, A.; Jackson, B. B.; Frenzel, R. M.; Sprehn, S. M.; Treiber, F. A. Development and validation of a smartphone heart rate acquisition application for health promotion and wellness telehealth applications. *Int. J. Telemedicine Appl. 2012* (Jan. 2012), 1:1–1:1.

[44] Grimm, R.; Wetherall, D.; Davis, J.; Lemar, E.; Macbeth, A.; Swanson, S.; Anderson, T.; Bershad, B.; Borriello, G.; Gribble, S. System support for pervasive applications. *ACM Transactions on Computer Systems 22*, 4 (2004), 421–486.

[45] Gunter, D.; Tierney, B. Netlogger: a toolkit for distributed system performance tuning and debugging. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on* (2003), pp. 97–100.

[46] Gustafsson, A.; Bång, M.; Svahn, M. Power explorer: a casual game style for encouraging long term behavior change among teenagers. In *Proceedings of the International Conference on Advances in Computer Enterntainment Technology* (New York, NY, USA, 2009), ACE '09, ACM, pp. 182–189.

[47] Gustafsson, A.; Katzeff, C.; Bang, M. Evaluation of a pervasive game for domestic energy engagement among teenagers. *Comput. Entertain. 7*, 4 (Jan. 2010), 54:1–54:19.

[48] Ho, A.; Hand, S. On the design of a pervasive debugger. In *Proceedings of the sixth international symposium on Automated analysis-driven debugging* (New York, NY, USA, 2005), AADEBUG'05, ACM, pp. 117–122.

[49] Humble, J.; Crabtree, A.; Hemmings, T.; Åkesson, K.-P.; Koleva, B.; Rodden, T.; Hansson, P. "playing with the bits" user-configuration of ubiquitous domestic environments. In *UbiComp 2003: Ubiquitous Computing*, A. Dey, A. Schmidt, and J. McCarthy, Eds., vol. 2864 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 256–263.

[50] Jevring, M.; de Groote, R.; Hesselman, C. Dynamic optimization of bluetooth networks for indoor localization. In *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology* (New York, NY, USA, 2008), CSTST '08, ACM, pp. 663–668.

[51] Li, F.; Zhao, C.; Ding, G.; Gong, J.; Liu, C.; Zhao, F. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 421–430.

[52] Lim, C.-H.; Wan, Y.; Ng, B.-P.; See, C.-M. A real-time indoor wifi localization system utilizing smart antennas. *IEEE Transactions on Consumer Electronics 53*, 2 (2007), 618–622.

[53] Lim, H.; Kung, L.-C.; Hou, J. C.; Luo, H. Zero-configuration indoor localization over ieee 802.11 wireless infrastructure. *Wireless Networks 16*, 2 (Feb. 2010), 405–420.

[54] Lu, H.; Frauendorfer, D.; Rabbi, M.; Mast, M. S.; Chittaranjan, G. T.; Campbell, A. T.; Gatica-Perez, D.; Choudhury, T. Stresssense: detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), UbiComp '12, ACM, pp. 351–360.

[55] Lu, J.; Whitehouse, K. Smart blueprints: Automatically generated maps of homes and the devices within them. In *Pervasive Computing*, J. Kay, P. Lukowicz, H. Tokuda, P. Olivier, and A. Krüger, Eds., vol. 7319 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 125–142.

[56] Macedo, E. L.; Ferreira, D. B.; Lemos, G.; Stajnberg, A.; Loques, O. Suporte para coleta e persistência de dados de contexto em um sistema de monitoramento domiciliar remoto de pacientes. In *Computer on the Beach 2011* (Florianópolis, SC, Brasil, abril 2011), pp. 219–228.

[57] Mareli, D. *Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes*. Dissertação de mestrado, Instituto de Computação – Universidade Federal Fluminense, 2013.

[58] Mareli, D.; Erthal, M.; Ferreira, D. B.; Loques, O. Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes. In *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (Brasília, DF, Brasil, maio 2013), pp. 643–656.

[59] Martin, M.; Nurmi, P. A generic large scale simulator for ubiquitous computing. In *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on* (2006), pp. 1–3.

[60] McBryan, T.; Gray, P. A model-based approach to supporting configuration in ubiquitous systems. In *Interactive Systems. Design, Specification, and Verification.* Springer, 2008, pp. 167–180.

[61] Medina, E. *Desenvolvimento de uma Rede Social para Pacientes com Insuficiência Cardíaca e Sistemas de Recomendação de Amizade.* Dissertação de mestrado, Instituto de Computação – Universidade Federal Fluminense, 2013.

[62] Nishikawa, H.; Yamamoto, S.; Tamai, M.; Nishigaki, K.; Kitani, T.; Shibata, N.; Yasumoto, K.; Ito, M. UbiREAL: Realistic Smartspace Simulator for Systematic Testing. In *UbiComp 2006: Ubiquitous Computing*, P. Dourish and A. Friday, Eds., vol. 4206 of *Lecture Notes in Computer Science.* Springer Berlin / Heidelberg, 2006, pp. 459–476.

[63] Norbisrath, U.; Mosler, C. Tool Support for the eHome Specification, Configuration, and Deployment Process. In *MSPE* (2006), pp. 109–122.

[64] Northrop, L. Sei's software product line tenets. *Software, IEEE 19*, 4 (2002), 32–40.

[65] OMG – The Object Management Group. CORBA – Common Object Request Broker Architecture. http://www.omg.org/gettingstarted/corbafaq.htm, 2013.

[66] O'Neill, E.; Klepal, M.; Lewis, D.; O'Donnell, T.; O'Sullivan, D.; Pesch, D. A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments. In *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities* (2005), IEEE, pp. 60–69.

[67] O'Neill, E.; Lewis, D.; Conlan, O. A Simulation-based Approach to Highly Iterative Prototyping of Ubiquitous Computing Systems. In *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques* (2009), ICST.

[68] Oracle. Java Remote Method Invocation. http://docs.oracle.com/javase/6/ docs/-platform/rmi/spec/rmiTOC.html, 2013.

[69] Otsason, V.; Varshavsky, A.; LaMarca, A.; de Lara, E. Accurate gsm indoor localization. In *Proceedings of the 7th international conference on Ubiquitous Computing* (Berlin, Heidelberg, 2005), UbiComp'05, Springer-Verlag, pp. 141–158.

[70] Papapostolou, A.; Chaouchi, H. Rfid-assisted indoor localization and the impact of interference on its performance. *Journal of Network and Computer Applications 34*, 3 (May 2011), 902–913.

[71] Park, J.; Moon, M.; Hwang, S.; Yeom, K. CASS: A Context-Aware Simulation System for Smart Home. In *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)* (2007), IEEE, pp. 461–467.

[72] Reynolds, V.; Cahill, V.; Senart, A. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks - InterSense '06* (New York, New York, USA, 2006), ACM Press, p. 1.

[73] Rodrigues, A. L. B. *Uma infra-estrutura para monitoramento de sistemas cientes do contexto.* Dissertação de mestrado, Programa de Pós Graduação em Engenharia Eletrônica (PEL) – Universidade do Estado do Rio de Janeiro, 2009.

[74] Romero, J. The take-anywhere, do-anything display. *Spectrum, IEEE 47*, 1 (2010), 46–51.

[75] Sohn, T.; Dey, A. iCAP: an informal tool for interactive prototyping of context-aware applications. In *CHI '03 extended abstracts on Human factors in computing systems - CHI '03* (New York, New York, USA, 2003), ACM Press, p. 974.

[76] Sztajnberg, A.; Rodrigues, A. L. B.; Bezerra, L. N.; Loques, O. G.; Copetti, A.; Carvalho, S. T. Applying context-aware techniques to design remote assisted living applications. *International Journal of Functional Informatics and Personalised Medicine 2*, 4 (2009), 358.

[77] Talukder, N.; Ahamed, S. I.; Abid, R. M. Smart tracker: Light weight infrastructure-less assets tracking solution for ubiquitous computing environment. In *Proceedings of the 2007 Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking&Services (MobiQuitous)* (Washington, DC, USA, 2007), MOBIQUITOUS '07, IEEE Computer Society, pp. 1–8.

[78] Tang, L.; Yu, Z.; Zhou, X.; Wang, H.; Becker, C. Supporting rapid design and evaluation of pervasive applications: challenges and solutions. *Personal and Ubiquitous Computing 15*, 3 (2010), 253–269.

[79] Uckelmann, D.; Harrison, M.; Michahelles, F. An architectural approach towards the future internet of things. In *Architecting the Internet of Things*, D. Uckelmann, M. Harrison, and F. Michahelles, Eds. Springer Berlin Heidelberg, 2011, pp. 1–24.

[80] Van Nguyen, T.; Kim, J. G.; Choi, D. ISS: The Interactive Smart home Simulator. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on* (2009), vol. 03, pp. 1828–1833.

[81] Vaquero, L. M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev. 39*, 1 (Dec. 2008), 50–55.

[82] Varshavsky, A.; de Lara, E.; Hightower, J.; LaMarca, A.; Otsason, V. Gsm indoor localization. *Pervasive and Mobile Computing 3*, 6 (Dec. 2007), 698–720.

[83] Vijayraghavan, V.; Barton, J. J. WISE-A Simulator Toolkit for Ubiquitous Computing Scenarios. In *UBITOOLS-'01 WORKSHOP* (2002), Citeseer.

[84] W3C – World Wide Web Consortium. Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis, 2013.

[85] Want, R. An introduction to ubiquitous computing. In *Ubiquitous Computing Fundamentals*. Chapman & Hall, 2010, pp. 1–36.

[86] Weis, T.; Knoll, M.; Ulbrich, A.; Muhl, G.; Brandle, A. Rapid prototyping for pervasive applications. *Pervasive Computing, IEEE 6*, 2 (april-june 2007), 76 –84.

[87] Weiser, M. The Computer for the 21st Century. *Scientific American 3* (1991), 94–104.

[88] Weiser, M. Some computer science issues in ubiquitous computing. *Communications of the ACM 36*, 7 (1993), 75–84.

[89] Weiser, M. The world is not a desktop. *ACM Interactions 1*, 1 (Jan. 1994), 7–8.

[90] Weiser, M.; Brown, J. S. The coming age of calm technology. In *Beyond calculation*. Springer, 1997, pp. 75–85.

[91] yao Jin, G.; Lu, X.-Y.; Park, M.-S. An indoor localization mechanism using active rfid tag. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on* (2006), vol. 1, pp. 4 pp.–.

[92] Zhang, L.; Suo, Y.; Chen, Y.; Shi, Y. SHSim: An OSGI-based smart home simulator. In *2010 3rd IEEE International Conference on Ubi-Media Computing* (2010), IEEE, pp. 87–90.

APÊNDICE A – Ferramentas utilizadas

A.1 AndEngine

Conforme descrito no Capítulo 4, A IPGAP foi desenvolvida sobre a tecnologia Android, através das bibliotecas do **AndEngine**[1]. O AndEngine é um popular motor de jogos *open source*, que oferece APIs de fácil utilização para o desenvolvimento de jogos 2D para a plataforma Android, utilizando OpenGL.

O AndEngine dá suporte às principais técnicas de desenvolvimento de jogos 2D, como utilização de *sprites* e a renderização de mapas em *tilesets*, além de facilidades como manipulação de menus, câmeras, texturas, primitivas geométricas, reconhecimento de gestos, colisões, utilização de modelos físicos simplificados entre diversos outros recursos.

(a) Textura de um *sprite* animado

(b) Textura de um *tileset*

Figura A.1: Texturas utilizadas na IPGAP

Toda a parte visual da IPGAP foi implementada através do AndEngine. Os ícones dos recursos do ambiente (e.g. fogão, TV, avatares) foram construídos a partir da API

[1] http://www.andengine.org/

de *sprites* do AndEngine, incluindo as funcionalidades de toque e arrastar-e-soltar. Os *sprites* são representações gráficas que podem ser estáticas ou animações. Os *sprites* animados são formados por um conjunto de imagens mostrando o objeto em diferentes posições – uma de cada vez, de acordo com a interação desejada (Figura A.1(a)). Além disso, o AndEngine gerencia a **textura** (imagem) de cada *sprite* eficientemente, de forma a economizar espaço em memória.

Listagem A.1: Exemplo de um arquivo TMX

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0" orientation="orthogonal" width="10" height="10"
    tilewidth="32" tileheight="32">
 <tileset firstgid="1" name="mytileset" tilewidth="32" tileheight="32">
  <image source="mytileset.png" width="352" height="512"/>
 </tileset>
 <layer name="Fundo" width="10" height="10">
  <data>
   <tile gid="21"/>
  ...
   <tile gid="21"/>
   <tile gid="21"/>
  </data>
 </layer>
</map>
```

Para a construção do mapa da residência, fez-se uso da técnica de *tilesets*. Os *tilesets* são um grupo de pequenas imagens de tamanho fixo (*tiles*), que são utilizadas para compor o fundo de jogos 2D. Os *tilesets* se beneficiam do fato de que em geral partes das imagens de fundo sejam iguais. Assim, as imagens repetidas são apenas referências ao *tile* (parte do *tileset*), que é carregado uma só vez na memória. Na Figura A.1(b), vemos o *tileset* utilizado na IPGAP, com *tiles* de 32×32 *pixels*. O AndEngine renderiza o mapa através da descrição contida em um arquivo XML, formato que ficou conhecido como TMX (Tile Map XML). Um exemplo de mapa no formato TMX pode ser visto na Listagem A.1, onde nas *tags* do tipo *tile* o atributo *gid* referencia o identificador do *tile* no *tileset*.

A.2 Tiled

O **Tiled**[2] é um *software* para criação de mapas 2D a partir de *tilesets*, utilizando o formato de arquivo TMX. O Tiled contém um editor visual, onde é possível compor o mapa desejado, selecionado os *tiles* do *tileset* escolhido e preenchendo a tela. Dentre os recursos do Tiled, estão a criação de mapas tanto *ortogonais* (perpendicular) como

[2] http://www.mapeditor.org/

isométricos, a manipulação de camadas e identificação de trechos do mapa (e.g. definição do trecho do mapa que representa a cozinha da casa).

Na criação do mapa, deve-se especificar a quantidade de *tiles* que o mapa comporta em sua largura e altura, além das dimensões do *tile* e escolha da imagem com o *tileset*. No mapa da IPGAP, por exemplo, foi utilizado um mapa de 40 *tiles* de largura e 24 de altura com *tiles* de 32×32 *pixels*, resultando uma imagem de 1024×768 *pixels*.

O Tiled salva o mapa criado no formato TMX, onde cada *tile* do mapa referencia um *tile* do *tileset*, através de um número de identificação. Note que os *tiles* do *tileset* podem ser referenciados por diversos *tiles* no mapa, o que leva a economia de memória.

Outra forma de economizar memória, é compactando trechos do arquivo TMX. No exemplo da IPGAP, o arquivo TXM possui a descrição de 40×24 = 960 *tiles*, que combinados com outras meta informações pode gerar um arquivo grande. O Tiled utiliza um algoritmo de compactação para gerar um arquivo TMX de tamanho reduzido. Na Listagem A.2 podemos ver o arquivo TMX utilizado no mapa da IPGAP na íntegra, onde observamos nas linhas 32 e 37 o trecho do arquivo compactado.

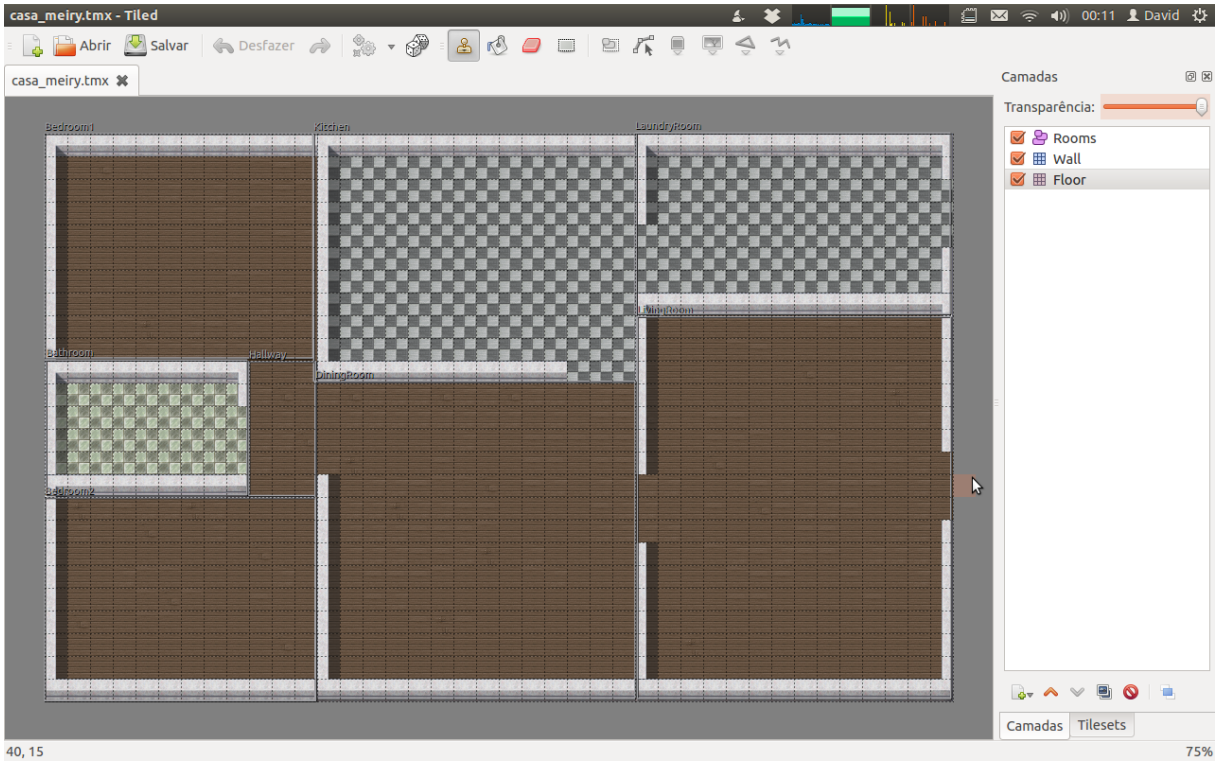


Figura A.2: Mapa da IPGAP construído no Tiled

Na Figura A.2 vemos a tela do Tiled, onde temos o mapa da IPGAP criado a partir do *tileset* da Figura A.1(b). Definimos no mapa duas camadas: uma camada com o chão

da casa, e outra com as paredes. Os *tiles* relativos às paredes da casa foram marcados como "bloqueados", para evitar que os avatares "passem por cima" delas. Pode-se observar essa marcação nas linhas 5 a 29 da Listagem A.2. Além disso, os cômodos da casa foram marcados e a área que os delimita identificadas no arquivo TMX resultante (ver linhas 40 a 49 da Listagem A.2).

Listagem A.2: Arquivo TMX utilizado pela IPGAP

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <map version="1.0" orientation="orthogonal" width="40" height="25"
       tilewidth="32" tileheight="32">
3   <tileset firstgid="1" name="mytileset" tilewidth="32" tileheight="32">
4    <image source="gfx/tiled.png"/>
5    <tile id="22">
6     <properties><property name="block" value="true"/></properties>
7    </tile>
8    <tile id="23">
9     <properties><property name="block" value="true"/></properties>
10   </tile>
11   <tile id="24">
12    <properties><property name="block" value="true"/></properties>
13   </tile>
14   <tile id="33">
15    <properties><property name="block" value="true"/></properties>
16   </tile>
17   <tile id="35">
18    <properties><property name="block" value="true"/></properties>
19   </tile>
20   <tile id="44">
21    <properties><property name="block" value="true"/></properties>
22   </tile>
23   <tile id="45">
24    <properties><property name="block" value="true"/></properties>
25   </tile>
26   <tile id="46">
27    <properties><property name="block" value="true"/></properties>
28   </tile>
29  </tileset>
30  <layer name="Floor" width="40" height="25">
31   <data encoding="base64" compression="gzip">
32    H4sIAAAAAAAAA+WWYQrDIAyFc6SdQG3+eZ/
        dnzGo9PFIou1KFfbjMek0fk1etJuIbB1V0OthNYbc4UuT+
        VBKv6vxMecK9e0p7bwr8VXj2Up8WNtZfOgtj0/l8OJMvq+KHL3RPHdHf/A+nI+0
```

```
        i8c6qDv4rDO156/3gBrbL/3r+ce6G1TO8ekeo8IYc5udMfok4mOuciF/
        SnEi4V7NS5gj9FePdTR/KvEdrcb+LP6/0rOr/uPcFYrBsa1exDnee+LcM/
        njuuK45aHCvsmYh2v5zkSf4rmC+cDzDp9ZdeQ5HNti4Pq3PCaKwX7htdmY4/
        Ehl9ejUf68frH6LBIzW3zoy17MCmutunp8eE9E/cletu6GEUarr6PatXVZ+
        r2fghgjivwa1S/6lvH2KU7cJxWdzWdUJr/Hv+kDwZvvKqAPAAA=
  </data>
 </layer>
 <layer name="Wall" width="40" height="25">
  <data encoding="base64" compression="gzip">
   H4sIAAAAAAAAA92WQQrAIAwE8wvrtXit/Yz/
        f0vNTaSou6IEA0svWTPGoHUicgFyYP6Mz2fdWQkQmi+VTwBvMM7H1DuVr/6Oei3z7Zw/
        y3waJ98vI7GKr8WrfNr32JCvOJ9OfqkyyyJ4gfL1+hM4aq+
        ZilO9vz5b5WN4dfDO9ZPg0evfSW/ExdVKxBupB6lnnY8+J5dvxvs34WD4mGN+
        pfOz5InHy/O3kQ/5HIpE/I30bPkq+CtagDwAA
  </data>
 </layer>
 <objectgroup name="Rooms" width="40" height="25">
  <object name="Bedroom1" x="0" y="0" width="378" height="316"/>
  <object name="Kitchen" x="379" y="0" width="455" height="349"/>
  <object name="LaundryRoom" x="832" y="-1" width="444" height="255"/>
  <object name="Bathroom" x="2" y="319" width="283" height="190"/>
  <object name="Bedroom2" x="0" y="513" width="382" height="286"/>
  <object name="DiningRoom" x="381" y="349" width="451" height="447"/>
  <object name="LivingRoom" x="835" y="258" width="442" height="538"/>
  <object name="Hallway" x="288" y="321" width="94" height="189"/>
 </objectgroup>
</map>
```