Uma Interface de Prototipagem para Aplicações Pervasivas

David Barreto[1], **Matheus Erthal**[1], **Douglas Mareli**[1], **Orlando Loques**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
CEP: 24210 – 240 – Niterói – RJ – Brasil

{dbarreto, merthal, dmareli, loques}@ic.uff.br

Resumo. *Este artigo descreve a Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP), que tem como objetivo prover uma plataforma de suporte à construção, teste e execução de aplicações para ambientes inteligentes (smart ambients). Para prover essa funcionalidade, a ferramenta proposta facilita a simulação de sensores e atuadores bem como meios para visualizar a interação com componentes reais presentes no ambiente. Assim, o desenvolvedor poderá construir suas aplicações sem a necessidade de se ter a infraestrutura completa de um ambiente inteligente.*

Abstract. *This article describes the Pervasive Applications Prototyping and Management Interface (IPGAP) that aims to provide a platform to support construction, test and execution of applications for smart ambients. In order to provide these features capabilities, our tool helps to perform simulation of sensors and actuators as well as means to visualize the interaction of real components which are inside the ambient. This way the developer will be able to construct applications without having a complete smart ambient infrastructure.*

1. Introdução

Desde as propostas de Mark Weiser na década de 1990 [Weiser 1991], os pesquisadores da área de computação ubíqua/pervasiva vêm propondo mudanças na interação homem-máquina, visando tornar o uso de dispositivos cada vez mais transparente no ambiente. Isso possibilita ao usuário, manter o foco na tarefa a ser realizada e não na ferramenta para realizá-la. A partir dessas ideias surgiu o conceito de ambientes inteligentes [Augusto and McCullagh 2007], onde sensores e atuadores interconectados em rede são capazes de fornecer informações relevantes sobre o ambiente para aplicações e usuários, bem como, efetivamente, agir neste ambiente e alterar seu estado.

O mercado de aplicações para plataformas *mobile* vem caminhando na direção acima descrita, com milhões de aplicações desenvolvidas e distribuídas para os usuários nos últimos anos, provendo serviços que cada vez mais estão se inserindo em seu cotidiano. Esse sucesso se deve aos crescentes avanços nas tecnologias de comunicação e, 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 [Lu et al. 2012], e um aplicativo que adquire a frequência cardíaca através do LED da câmera de um *smartphone* [Gregoski et al. 2012]. Entretanto, de modo geral essas aplicações ainda 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 justamente utilizar essas aplicações integradas a um ambiente inteligente, fornecendo seus serviços e informações à outras entidades.

Em comparação ao avanço no desenvolvimento de aplicações *mobile*, as aplicações ubíquas ainda são escassas no mercado. Podemos citar como causas desse efeito o alto custo de desenvolvimento demandado por elas, devido a questões como a falta de ferramentas adequadas para a criação e integração dessas aplicações, e a dificuldade em depurá-las [Weis et al. 2007]. 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.

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 fornece ao desenvolvedor um ambiente de testes para suas aplicações de maneira rápida e com baixo custo. Além disso, a IPGAP oferece 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, contendo uma GUI (*Graphic User Interface)* para composição de regras. Assim, o desenvolvedor poderá se utilizar da infraestrutura provida para criar uma aplicação pervasiva mais facilmente.

Consideremos uma aplicação para controle do consumo de energia em uma residência aos moldes de [Costanza et al. 2012]. Esta poderia coletar as informações de consumo de energia (em watts) de cada recurso no ambiente, e a partir dessas informações aplicar políticas predefinidas para tomar uma decisão, como restringir a ligação de grandes consumidores de energia em horários de pico ou escalonar a ligação desses aparelhos para horários com tarifação baixa. O desenvolvedor dessa aplicação, além de contar com uma API que possibilita reunir essas informações e atuar em recursos do ambiente, poderá testar sua aplicação na interface de prototipagem de forma rápida e barata, sem a necessidade de montar uma infraestrutura completa para um ambiente inteligente.

O restante deste artigo está organizado como a seguir. Na Seção 2, apresentaremos uma visão geral dos principais conceitos utilizados como base para o desenvolvimento da IPGAP. Na Seção 3, veremos mais detalhes sobre o funcionamento da ferramenta, seus conceitos, características e exemplos de utilização. Mostraremos na Seção 4 uma avaliação da IPGAP através de aplicações desenvolvidas. Os trabalhos relacionados serão apresentados e discutidos na Seção 5, e as conclusões e trabalhos futuros, na Seção 6.

2. Visão Geral da Infraestrutura da IPGAP

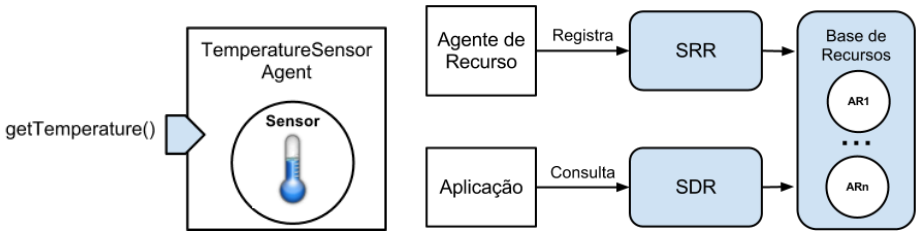
A IPGAP é pautada em conceitos desenvolvidos em nosso grupo de pesquisa, que foram postos em prática através do projeto **SmartAndroid**[1], desenvolvido para comprovação de conceitos propostos em três dissertações de Mestrado [Barreto 2012, Erthal 2012, Mareli 2012]. O projeto SmartAndroid contempla o desenvolvimento das APIs e todo o *framework* proposto, além da interface de prototipagem, utilizando a plataforma Android. Veremos a seguir uma breve descrição dos conceitos que servem de base para a implementações realizadas.

[1]Para mais informações visite www.tempo.uff.br/smartandroid

2.1. Agentes de Recurso

Sensores, atuadores, dispositivos e eletrodomésticos inteligentes, além de módulos de *software* que forneçam algum serviço para o ambiente, são definidos como *recursos*. Estes são encapsulados em Agentes de Recursos (AR), que podem ser compreendidos como elementos que expõem informações dos recursos juntamente com sua interface, de forma que outras entidades possam acessá-las de maneira uniforme.

Os ARs escondem detalhes de baixo nível do recurso encapsulado, diminuindo significativamente a complexidade de integração de um recurso no sistema. Por exemplo, os detalhes da coleta de dados de um sensor de temperatura seriam conhecidos apenas pelo seu AR, que se encarrega de fornecer uma interface simples para que os outros componentes do sistema tenham acesso as informações desse sensor. Na Figura 1(a) temos um AR que encapsula um sensor de temperatura, expondo o método `getTemperature()`.



(a) Um Agente de Recurso para um sensor de Temperatura

(b) Serviço Básicos do *framework*

Figura 1. Componentes do Sistema

2.2. Serviços Básicos

É necessário que os recursos do ambiente sejam descobertos para que as aplicações possam utilizá-los. Por esse motivo existe o *Serviço de Descoberta de Recursos* (SDR), que permite que estes sejam localizados por meio de consultas a uma base, populada com as referências dos ARs pelo *Serviço de Registro de Recursos* (SRR) (Figura 1(b)).

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 [Bezerra 2011] para um detalhamento). Na Figura 2 vemos a assinatura dos principais métodos de busca do SDR. Para mais informações sobre os serviços básicos utilizados no *framework*, veja [Sztajnberg et al. 2009, Mareli 2012].

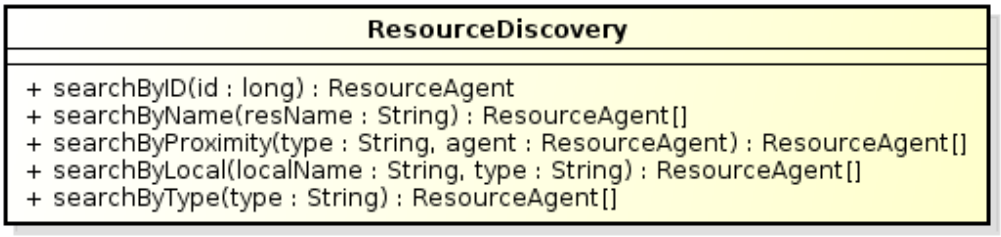


Figura 2. Métodos de Busca do SDR

Como exemplo de uso do mecanismo de descoberta, apresentamos uma aplicação de monitoramento de pacientes [Carvalho et al. 2010]. Essa aplicação utiliza sensores para inferir o estado de saúde do paciente, através da coleta contínua de dados do ambiente (e.g. temperatura, umidade), dados fisiológicos (e.g. pressão arterial, frequência cardíaca), bem como um plano de cuidados – um conjunto de prescrições feitas por um profissional de saúde, contendo os medicamentos que o paciente deve tomar, medições fisiológicas, exercícios físicos, e outras recomendações com seus respectivos horários.

Para aumentar a adesão do paciente ao tratamento, é salutar que este seja alertado no momento em que deve realizar uma tarefa, através de um dispositivo como TV, celular, *tablet*, etc. Porém, nem sempre um dispositivo está à vista do paciente. Por exemplo, enviar a mensagem *"Está na hora de tomar o remédio"* para o celular do paciente, pode não ser eficaz se o paciente não tiver o hábito de estar próximo ao celular. Por outro lado, enviar a mesma mensagem para todos os dispositivos de visualização da residência, apesar de à primeira vista ser mais eficaz, pode ser um estorvo para o paciente e os outros moradores da casa, além de ser uma abordagem muito intrusiva podendo causar constrangimentos e exposição desnecessária do paciente. Utilizando-se o SDR, a aplicação poderia fazer a consulta: *"o dispositivo de visualização mais próximo do paciente"* para obter uma referência para este recurso, e assim, enviar a mensagem de alerta. Dessa forma a aplicação seria menos intrusiva e reduziria a possibilidade do paciente não perceber o alerta. Um trecho de código com exemplos de uso do SDR poderá ser encontrado na Seção 4.1.

2.3. Contexto

Aplicações sensíveis ao contexto são integradas com o mundo físico, e respondem a estímulos do ambiente obtidos através de sensores. Essa é uma caracterização fundamental para sistemas ubíquos/pervasivos. Assim, podemos dizer que essas aplicações são interessadas nas informações dos recursos que sejam relevantes para o sistema, ou seja, no contexto dos recursos e do ambiente (veja [Abowd et al. 1999] para mais informações sobre contexto). Em uma aplicação feita para um ambiente inteligente, por exemplo, o contexto poderia ser: se uma lâmpada está acesa, o canal em que se encontra uma TV, a temperatura de um ar-condicionado, entre outros.

As aplicações construídas através do *framework* proposto, podem utilizar regras envolvendo o contexto dos recursos, conhecidas como *regras de contexto*. Essas regras são compostas por condições obtidas de informações de contexto dos ARs envolvidos e de um temporizador. Um *interpretador de contexto* avalia constantemente as condições da regra. Se verificado que a condição é verdadeira, este notifica os ARs interessados, ou seja, aqueles que vão de fato efetuar uma ação no ambiente (Figura 3). Essa notificação é realizada através do padrão *publish-subscribe*, largamente utilizado em sistemas distribuídos.

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 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 [Copetti et al. 2012]. Um estudo mais profundo sobre interpretação de contexto pode ser encontrado em [Erthal 2012].

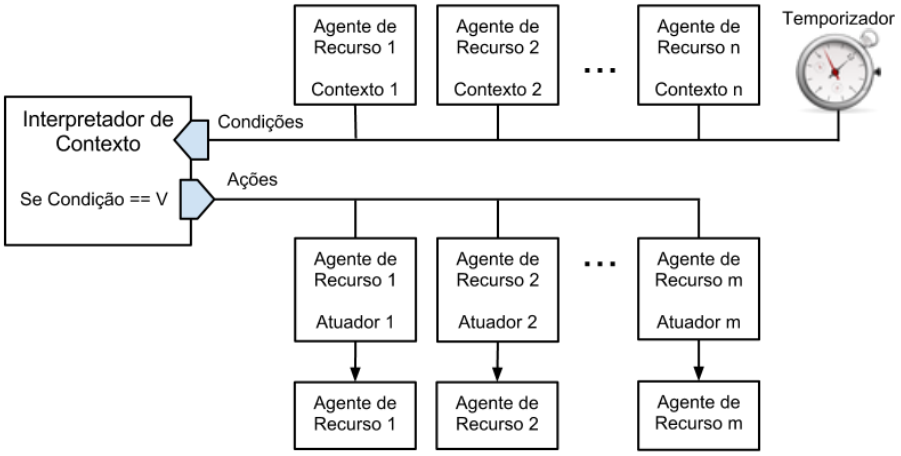


Figura 3. Interpretação de uma Regra de Contexto

3. A Interface de Prototipagem de Aplicações Pervasivas

A IPGAP foi desenvolvida para fornecer a seus usuários um mundo em que os dispositivos interajam entre si, e são facilmente acessíveis utilizando a API provida no *framework*. Um desenvolvedor de aplicações para ambientes ubíquos/pervasivos pode testar suas aplicações no ambiente da IPGAP, de modo a avaliá-las e, consequentemente, aperfeiçoá-las. Isso seria um desafio se fosse preciso desenvolver toda infraestrutura a partir do zero.

Através de aparelhos de fácil aquisição, que podem simular vários dispositivos de um ambiente inteligente, pode-se criar diversos cenários à baixo custo. Por exemplo, nossa prova de conceito foi implementada sobre o sistema operacional Android, sendo utilizado um *tablet* para visualização da ferramenta de prototipagem (Figura 4). Os simuladores dos dispositivos (ver Seção 3.4.1) foram instalados em celulares de baixo custo conectados por uma rede sem fio.

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 em fases iniciais. A ferramenta também dá suporte à integração de dispositivos reais no ambiente juntamente com os dispositivos simulados, criando assim um ambiente híbrido. Para as aplicações este fato é totalmente transparente, pois permite que os dispositivos reais e simulados sejam facilmente intercambiáveis. É importante salientar que os serviços básicos do sistema – como o SDR, SRR e os interpretadores de contexto vistos na Seção 2 – podem ser executados em máquinas mais robustas, assegurando, assim, requisitos de qualidade de serviço e tolerância a falhas.

Figura 4. IPGAP: Visão do mapa do ambiente

3.1. A IPGAP para o Desenvolvedor

O desenvolvedor tem a capacidade de, através da IPGAP, visualizar o ambiente alvo na tela de um computador ou *tablet*, em um formato similar ao de uma planta baixa (Seção 3.3). O ambiente pode então ser populado na tela com os recursos necessários à aplicação que está sendo desenvolvida. Por exemplo, para que uma aplicação de controle de iluminação seja testada, é necessário que se tenha lâmpadas disponíveis, além de sensores que detectem a presença de pessoas nos cômodos e avatares para representar essas pessoas. O desenvolvedor pode adicionar todos esses recursos no mapa da casa e executar sua aplicação. Para testá-la, uma das alternativas seria utilizar a IPGAP para movimentar o avatar no mapa, e observar os sensores detectando a presença ou ausência do indivíduo, bem como as lâmpadas se acendendo e apagando.

3.2. A IPGAP para o Usuário Final

A IPGAP possui a característica de ser um *software* intuitivo, por conta de sua GUI. Isso faz com que seja uma ferramenta muito conveniente para o usuário final (entenda-se "usuário final" como um utilizador do ambiente inteligente, potencialmente leigo na área da computação.), pois se torna um poderoso controle remoto de todos os dispositivos de sua residência. Assim, o usuário final pode, via rede sem fio, manipular os dispositivos e criar regras de contexto que atuarão em seu ambiente. Dessa forma, pode-se configurar todo o ambiente de acordo com as preferências do usuário, através de um dispositivo móvel como um *smartphone* ou *tablet*.

A situação ideal é que este usuário possa controlar toda sua casa remotamente, de qualquer parte do mundo através da internet. É claro que isso envolve uma série de questões sobre segurança de rede, níveis de permissões, entre outras. Estes aspectos são abordados com detalhes em [Mareli 2012].

3.3. Representação do ambiente

Anteriormente, mencionamos que os recursos do ambiente poderiam ser visualizados através de representações gráficas na tela de um dispositivo (como um *tablet*), para que

o desenvolvedor ou usuário final possa manipulá-los. Essas representações são posicionadas em um mapa esquemático do ambiente inteligente. Cada cômodo no mapa tem sua área pré-definida pelo usuário em um editor de mapas[2], que também é responsável por definir a aparência da casa representada. Na Figura 4 vemos o mapa do ambiente, populado com alguns recursos rodando em um *tablet*

Os recursos são exibidos no mapa pela IPGAP através de ícones, de acordo com o desejo do usuário, e suas posições são aproximadas em relação a posição real do recurso em metros. Dessa forma, o mapa do ambiente se transforma em uma espécie de "área de trabalho", semelhante à de sistemas operacionais como Windows, Linux e Android, contendo ícones que podem ser acionados para chamar outros processos.

Salientamos que os recursos representados no mapa não são meramente imagens, sendo na verdade um espelho do ambiente inteligente. Os recursos estão registrados no sistema, e podem ser descobertos e utilizados por aplicações.

3.4. 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). Nos ambientes *mobile*, existem lojas virtuais (como Google Play e Apple Store) que disponibilizam uma base com milhares de aplicativos que podem ser baixados pelos usuários. Dessa forma, pode-se facilmente customizar o ambiente com a instalação de novos *softwares*.

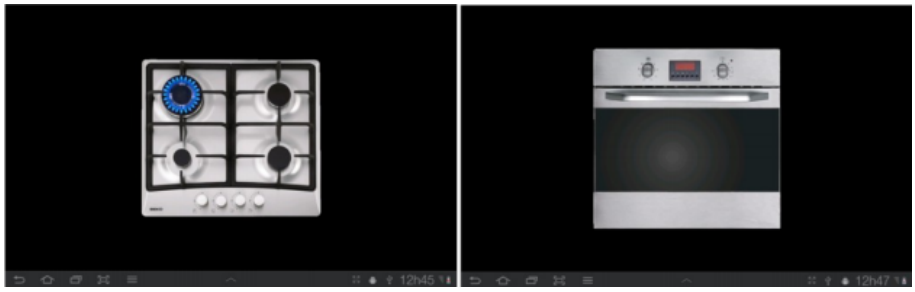


Figura 5. Aplicativo do fogão em um *tablet*. Visão das bocas e do forno

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 5), 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

[2]Foi o utilizado o *software* **Tiled** (www.mapeditor.org) para criação do mapa do ambiente de testes.

seu fabricante. Ver Seção 3.5), conforme mostramos no esquema da Figura 6. 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 3.3), passando a exibir a interface do dispositivo que representa, como na Figura 5.

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

3.4.1. 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 constantemente 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.

Figura 6. Aplicativo do fogão atuando no fogão real

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.

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.

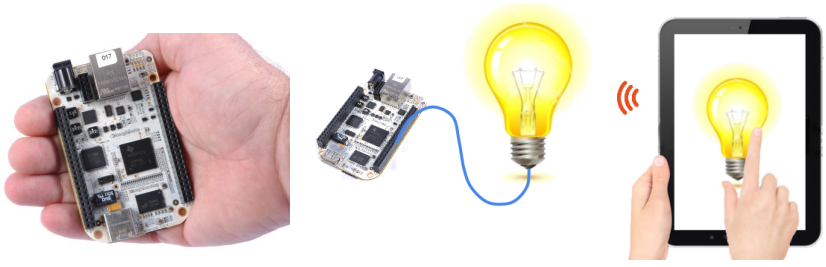
3.4.2. Instalação dos Aplicativos

Como citado anteriormente, nossa ferramenta permite a instalação de aplicativos no ambiente de prototipagem, de maneira similar ao que ocorre nos sistemas operacionais de plataformas *mobile*. 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.

3.5. Utilização de Recursos Externos

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 de ambientes inteligentes. Para isso os dispositivos devem possuir um chip embutido em seu próprio *hardware*, que implemente as APIs padronizadas do *framework*.

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 "tradução" 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.



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

Figura 7. Experimentos com Beaglebone

Nosso grupo está realizando testes com o Beaglebone [BeagleBoard 2012] com o objetivo de construir protótipos de diversos dispositivos. O Beaglebone (Figura 7(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 comum e controlá-la através de um relé ligado aos pinos de I/O do Beaglebone. A Figura 7(b) ilustra essa possibilidade.

4. Aplicações

Um conjunto de aplicações pervasivas foi construído utilizando-se nossa API e a IPGAP como plataforma de testes. Através do desenvolvimento dessas aplicações foi possível avaliar a viabilidade da proposta, assim como a flexibilidade e usabilidade da mesma. Foram implementadas aplicações para controle dos dispositivos da casa (como um controle remoto), controle de iluminação da residência e jogos *multiplayer* onde usuários do ambiente podem descobrir jogadores e jogar em conjunto, entre outras aplicações.

Uma das aplicações implementadas, nomeada de **MediaFollowMe**, proporciona ao usuário final a facilidade de repassar áudio/vídeo/imagens (ou qualquer outro tipo de mídia) de um aparelho para outro automaticamente, levando em conta informações como proximidade do dispositivo em relação ao usuário, como se a mídia o seguisse. Exemplificaremos esta aplicação a seguir.

4.1. MediaFollowMe

O usuário final está em casa assistindo a um filme em seu *Blu-ray* na TV da sala, quando em um ponto crítico da trama precisa se deslocar rapidamente para outro cômodo. Ele inicia através de seu *smartphone* o aplicativo **MediaFollowMe** e escolhe dentre uma lista de dispositivos presentes no ambiente (exibida na tela do aparelho) o *Blu-ray* da sala como fonte emissora da mídia.

A partir daí, a aplicação não necessita mais da interação direta do usuário, atuando autonomamente no ambiente. As referências dos ARs que representam o *Blu-ray* da sala e o próprio usuário são obtidas através do SDR, para saber de onde obter a mídia e para onde deve transmiti-la (conforme a posição do usuário). Além disso, a aplicação deve possuir as referências dos ARs dos sensores de presença da casa, a fim de receber um evento (ou seja, ser notificada) quando o usuário entrar ou sair de um cômodo. 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.

Listagem 1. Obtendo as referências e registrando interesse nos eventos

```
1 ResourceAgent[] pSensors = discovery.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, a aplicação MediaFollowMe se registra como interessada em receber eventos dos ARs dos sensores de presença do ambiente. Na Listagem 1 vemos um trecho de código exemplificando esse processo. Na linha 1 é feita uma busca por todas 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 a sala, o sensor de presença correspondente detectará que alguém saiu, e notificará através de um evento a todos os interessados nessa informação, incluindo o MediaFollowMe. Os eventos devem informar quem é o recurso que o originou (neste caso, o sensor de presença da sala), qual o contexto alterado (entrada ou saída do cômodo) e o 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.

Dessa forma, quando o usuário adentra a cozinha, o sensor de presença deste cômodo detecta a sua chegada e notifica à aplicação via eventos. A aplicação busca o dispositivo mais próximo do usuário e descobre uma TV na cozinha. Agora, o MediaFollowMe pode realizar uma chamada remota para o *Blu-ray* (fonte dos dados) solicitando que ele mude o receptor de vídeo da TV da sala para a TV da cozinha, e retoma a reprodução do filme (que estava em pausa). Isso é feito automaticamente e sem fio, resultando na transferência da mídia para esta TV.

Na Listagem 2 apresentamos o tratamento dos eventos recebidos pela aplicação. 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 de 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 subclasse de **ResourceAgent**, portanto pode ser passada como parâmetro na consulta de proximidade (linha 12), que retorna uma lista de recursos ordenados por proximidade em relação ao recurso passado (neste caso o usuário). Na linha 6 a aplicação testa se o usuário que entrou/saiu do cômodo em questão é o usuário requerido. Por fim, na linha 13 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, e na linha 14, é realizado uma chamada remota ao *Blu-ray* para reproduzir a mídia.

Listagem 2. 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
9        bluray.pause();
10     } else if(context.equals("IN") {
11
12       ResourceAgent[] views = discovery.searchByProximity("View", p);
13       bluray.streamTo(views[0]);
14       bluray.play();
15     }
16   }
17 }
```

Foi possível observar a aplicação em funcionamento e o comportamento dos recursos envolvidos através da IPGAP. Ao movimentar-se o avatar do usuário pelos cômodos da casa, pôde-se visualizar a mídia sendo exibida no dispositivo de visualização mais próximo a ele, conforme esperado.

5. Trabalhos Relacionados

Podemos encontrar na literatura outras propostas de ferramentas para prototipagem de aplicações pervasivas (veja [Tang et al. 2010] para uma *survey*). Em [Armac and Retkowitz 2007, Van Nguyen et al. 2009] encontramos ferramentas de teste e a avaliação de serviços para ambientes inteligentes. Entretanto, não incluem uma abordagem de descoberta de dispositivos recém adicionados no ambiente. A IPGAP abrange essa questão através do SDR e SRR (Seção 2).

Já em [Zhang et al. 2010] e [Fu et al. 2011] encontramos simuladores para ambientes inteligentes baseados em OSGi (*Open Service Gateway Initiative*), oferecendo a possibilidade de adição e remoção de dispositivos sem alterar o código do sistema, através da modificação de arquivos de configuração. Porém, estes trabalhos não focam no perfil do usuário final (Seção 3.2), pois não proveem uma interface clara para que o usuário configure seus dispositivos.

Em [Bruneau and Consel 2012] é apresentado um simulador para aplicações pervasivas, parametrizado por diretivas escritas em uma linguagem de configuração própria, porém não oferece suporte à configuração dinâmica das entidades do sistema. Além disso o trabalho não deixa claro a possibilidade de transparência entre dispositivos reais e simulados, recursos providos pela IPGAP.

Outra vertente de trabalhos sobre simulação de sistemas pervasivos pode ser encontrada nos trabalhos [Barton and Vijayaraghavan 2002, Nishikawa et al. 2006]. Esses projetos possuem uma visualização 3D do ambiente em que o usuário tem a possibilidade de controlar um avatar em uma visão de 3ª pessoa, como nos jogos de computador. Assim como a IPGAP, estes projetos permitem que se configure dispositivos no sistema e se interaja com eles. Porém, as ferramentas propostas nestes trabalhos focam em testes de interação dos dispositivos e a simulação do ambiente físico, não se preocupando com sensibilidade ao contexto, ou seja, não consideram um ambiente adaptativo onde o contexto pode causar a alteração do estado dos recursos.

6. Conclusão e Trabalhos Futuros

Neste artigo foi apresentada a IPGAP – Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas – que visa auxiliar o desenvolvedor a construir e testar aplicações para ambientes inteligentes. A IPGAP permite que a criação de protótipos funcionais seja realizada em menos tempo e à baixo custo, através do uso de um misto de simuladores e dispositivos reais, e uma API para gerenciamento do ambiente. As aplicações desenvolvidas podem ser facilmente instaladas no ambiente inteligente, permitindo que se realize testes com uma infinidade de cenários. Além disso, aplicativos para gerenciamento dos recursos do ambiente (como fogão e TV) podem ser instalados na IPGAP, permitindo que seja estendida de acordo com as necessidades do desenvolvedor, sem necessidade de alteração no código.

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, que pode através da IPGAP controlar o ambiente inteligente em que se encontra. As próximas etapas do projeto incluem o tratamento de informações de contexto mais complexas e a implementação de uma visualização 3D do ambiente inteligente na interface, além da utilização de uma abordagem baseada em contratos [Carvalho et al. 2011], para adaptação dinâmica da aplicação às necessidades do usuário final, possibilitando também a criação de subsistemas.

Referências

Abowd, G., Dey, A., Brown, P., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In Gellersen, H.-W., editor, *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer Berlin Heidelberg.

Armac, I. and Retkowitz, D. (2007). Simulation of Smart Environments. In *IEEE International Conference on Pervasive Services*, pages 257–266. IEEE.

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

Barreto, D. (2012). *Uma Interface de Prototipagem e Gerenciamento para Aplicações Pervasivas*. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense.

Barton, J. J. and Vijayaraghavan, V. (2002). Ubiwise, a ubiquitous wireless infrastructure simulation environment. Technical Report HPL-2002-303, Hewlett-Packard Laboratories, Palo Alto.

BeagleBoard (2012). Open hardware physical computing on arm and linux. http://beagleboard.org/.

Bezerra, L. N. (2011). *Uso de ontologia em serviço de contexto e descoberta de recursos para autoadaptação de sistemas*. Dissertação de mestrado, Universidade do Estado do Rio de Janeiro.

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

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

Carvalho, S. T., Murta, L., and Loques, O. (2011). A contract-based approach for managing dynamic variability in software product line architectures. Technical report, Laboratorio Tempo – Universidade Federal Fluminense.

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

Costanza, E., Ramchurn, S. D., and Jennings, N. R. (2012). Understanding domestic energy consumption through interactive visualisation: a field study. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 216–225, New York, NY, USA. ACM.

Erthal, M. (2012). *Interpretação de contexto em ambientes ubíquos inteligentes*. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense.

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

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

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

Mareli, D. (2012). *Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes*. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense.

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

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

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

Van Nguyen, T., Kim, J. G., and Choi, D. (2009). ISS: The Interactive Smart home Simulator. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, volume 03, pages 1828–1833.

Weis, T., Knoll, M., Ulbrich, A., Muhl, G., and Brandle, A. (2007). Rapid prototyping for pervasive applications. *Pervasive Computing, IEEE*, 6(2):76 –84.

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

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