Uma Interface de Prototipagem para Aplicações Pervasivas

David Barreto[1], Douglas Mareli[1], Matheus Erthal[1], André Coelho[1], Tácio Diogo[1], Sergio Carvalho[1,2], Orlando Loques[1]

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

[2]Instituto de Informática - Universidade Federal de Goiás (UFG)
Goiânia – GO – Brasil

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

{andre.ribeiro.coelho, taciosd}@gmail.com

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 uma smart home. Para prover essa funcionalidade, a ferramenta proposta faz a simulação de sensores e atuadores bem como provê a interação com componentes reais presentes na casa. Assim, o desenvolvedor poderá construir suas aplicações sem a necessidade de se ter a infraestrutura completa de uma smart home.*

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 homes. In order to provide this feature, our tool performs simulation of sensors and actuators as well as provides an interaction of real components which are inside the house. This way the developer will be able to construct applications without having a complete smart home infrastructure.*

1. Introdução

Desde as propostas de Mark Weiser na década de 1990 [Weiser 1991], os pesquisadores da área de computação ubíqua e pervasiva vem propondo mudanças na interação homem-máquina usual, visando que o uso de dispositivos computacionais se torne cada vez mais transparente no ambiente. Isso possibilita que o usuário mantenha 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, alterando seu estado.

O mercado de aplicações para plataformas *mobile* vem caminhando nessa direção, com milhões de aplicações desenvolvidas e distribuídas para os usuários nos últimos anos, provendo serviços que cada vez mais estão se inserindo em suas vidas cotidianas. Esse sucesso se deve aos crescentes avanços nas tecnologias de comunicação e, sobretudo, na alta demanda do uso de *smartphones* e o surgimento de sistemas operacionais mais robustos para esses dispositivos, como o Google Android e o Apple iOS. 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 outro 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 autossuficientes, ou seja, não compartilham as informações que geram, 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 e pervasiva é justamente utilizar essas aplicações da maneira que Weiser descreveu, integrados a um ambiente inteligente e fornecendo seus serviços e informações à outras entidades.

Em comparação a esse grande 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 (1) a falta de ferramentas adequadas para a criação e integração dessas aplicações, (2) a dificuldade em depurá-las e (3) problemas com a interoperabilidade de sensores, atuadores e outros dispositivos necessários à aplicação. 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, limitando o desenvolvedor em demasia.

Nossa proposta consiste em um *framework* que tem por objetivo gerenciar os diversos recursos presentes em um ambiente inteligente – como TVs, aparelhos de arcondicionado e lâmpadas – além de oferecer funcionalidades como a descoberta dos recursos disponíveis dinamicamente e o suporte para que estes recursos se comuniquem, através de invocação remota e eventos. Os recursos são encapsulados em *Agentes de Recursos* (ARs), que contêm a interface das operações disponíveis no recurso encapsulado. Por exemplo, para utilizar um aparelho de TV no sistema, deve-se criar um AR que conterá as operações deste dispositivo, como `ligar`, `trocarCanal`, `mostrarMensagem`, entre outras. Certamente esse AR conterá também as funcionalidades básicas do sistema.

Além disso é fornecida uma API para criação de regras que envolvem os recursos disponíveis no ambiente. Por exemplo, o desenvolvedor poderia utilizar esta API para compor a seguinte regra: “se a TV estiver ligada e ninguém estiver no mesmo cômodo que ela por mais de 50 min, desligar essa TV”. O sistema se encarrega de fazer o *binding* com os ARs requeridos, coletar as informações necessárias e, ao detectar a condição como verdadeira, notificar os ARs interessados nessa condição através de comunicação por eventos (*publish-subscribe*).

Porém, por mais poderosa que seja, uma API por si só muitas vezes não é o suficiente para auxiliar um desenvolvedor a criar protótipos de suas aplicações bem como testá-las e depurá-las. Por isso, nossa proposta conta com a Interface de Prototipagem e Gerenciamento de Aplicações pervasivas (IPGAP) que fornece um ambiente com dispositivos simulados, como TV, fogão, lâmpada, sensores e atuadores em geral, além da simulação dos usuários (moradores e visitantes) por meio de avatares que podem “caminhar” em uma representação gráfica do mapa do ambiente. Assim, o *framework* proposto torna-se uma plataforma de desenvolvimento e testes de aplicações pervasivas e uma ferramenta poderosa para o usuário final, que poderá controlar todos os recursos de seu

ambiente e criar regras de contexto de maneira amigável por meio de um editor.

Assim, o desenvolvedor poderia se utilizar de nossa infraestrutura 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 nosso ambiente através da interface de seus ARs, 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, entre outras ações. O desenvolvedor dessa aplicação além de contar com uma API que o possibilite reunir essas informações e atuar em recursos do ambiente, poderá testar sua aplicação em nossa interface de prototipagem de forma rápida e barata, sem a necessidade de montar uma infraestrutura completa para uma *smart home*.

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 de nossa ferramenta, seus conceitos e características, bem como diversos 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 apresentados na Seção 6.

2. Visão Geral da Infraestrutura da IPGAP

A interface de prototipagem é pautada em conceitos e implementações desenvolvidos em nosso grupo de pesquisa, que foram postos em prática através do projeto **SmartAndroid**, que contempla o desenvolvimento de nossa interface de prototipagem (para mais informações visite www.tempo.uff/smartandroid). Veremos a seguir uma breve descrição dos conceitos que servem de base para a implementação de nossa ferramenta.

2.1. Agentes de Recurso

Os recursos - sensores, atuadores, dispositivos inteligentes, eletrodomésticos inteligentes e módulos de software que forneçam algum serviço para o ambiente - são encapsulados (*wrap pattern*) em Agentes de Recursos (AR). Estes 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 são conhecidos apenas pelo seu AR (TemperatureSensorAgent) que também se encarrega de fornecer uma interface simples para que os outros componentes do sistema tenham acesso à essa, e outras operações (Figura 1(a)). Um AR pode ser implementado de diversas maneiras, como por exemplo *Web Services*.

2.2. Descoberta de Recursos

Para que as aplicações utilizem esses recursos, é necessário que estes sejam primeiramente descobertos. Em outras palavras, é necessário que seja obtida a referência para o recurso desejado, pois se trata de um sistema distribuído. Em nossa proposta, existe

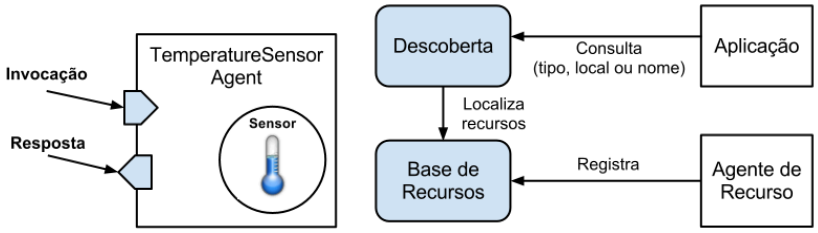
um componente que gerencia a descoberta de recursos em uma base, chamado *Serviço de Descoberta*. As referências dos recursos são armazenadas em um repositório, após o seu registro no sistema através do *Serviço de Registro*. Os recursos podem ser descobertos através de vários tipos de consulta, como **tipo**, **localização** e **nome**. Por exemplo, poderíamos buscar "todos os recursos do tipo Lâmpada", o que seria uma busca por tipo, ou "a TV mais próxima do usuário" e "o ar-condicionado da sala de estar" - buscas por localização - e por fim, "o recurso de nome *sensorTempDoQuarto*" que é uma busca por um nome específico. Assim será retornado ao chamador uma lista de referências que condizem com a consulta passada (Figura 1(b)). Um exemplo de código será apresentado mais tarde, na Seção 4.1. Para mais informações veja [Sztajnberg et al. 2009].

Como exemplo de uso do mecanismo de descoberta, apresentamos uma aplicação de monitoramento de pacientes [Carvalho et al. 2010]. Essa aplicação utiliza sensores para continuamente coletar dados do ambiente (e.g. temperatura, umidade) e dados fisiológicos do paciente (e.g. pressão arterial, frequência cardíaca), para inferir seu estado de saúde. Essa aplicação conta também com um plano de cuidados, um conjunto de prescrições feitas por um profissional de saúde, contendo os medicamentos que o paciente deve tomar, além de 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 sempre com o celular, quando estiver em casa. 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 *Serviço de Descoberta* provido pelo nosso sistema, 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 maneira diminuíram-se as chances de o paciente não perceber o alerta, além de agir de forma menos intrusiva.

2.3. Contexto

Aplicações sensíveis ao contexto, são aplicações integradas com o mundo físico, que respondem a estímulos do ambiente obtidos através de sensores. Essa é uma caracterização



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

(b) Serviço de Descoberta

Figura 1. Componentes do Sistema

fundamental para sistemas ubíquos e pervasivos. Assim, podemos dizer que essas aplicações são interessadas nas informações dos recursos que são relevantes para o sistema, ou seja, no contexto dos recursos e do ambiente (o leitor interessado pode encontrar uma boa definição de contexto em [Abowd et al. 1999]). Em uma aplicação feita para uma *smart home*, por exemplo, o contexto poderia ser: uma lâmpada está acesa, o canal em que se encontra uma TV, a temperatura de um ar-condicionado, entre outros.

As aplicações construídas tendo como base nossa proposta podem utilizar regras envolvendo o contexto dos recurso. Essas regras, conhecidas como *regras de contexto*, são compostas de condições e ações que podem ser tanto definidas programaticamente quanto em alto nível, através de uma GUI. Essas regras são processadas por um *interpretador de contexto*, que as avalia constantemente. Se verificado que a condição é verdadeira, este notifica os ARs interessados, ou seja, aqueles que vão de fato efetuar a ação da regra. Uma *smart home* frequentemente possui diversas regras de contexto ativas ao mesmo tempo, sendo estas 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.

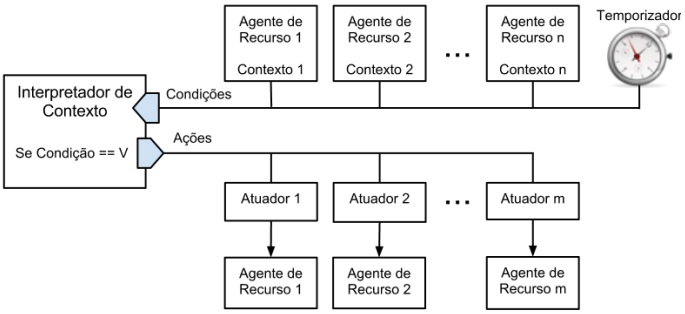


Figura 2. Interpretação de uma Regra de Contexto

Podemos perceber que as condições são alimentadas pelo contexto dos recursos em questão, os quais são combinados por conectivos lógicos como **e**, **ou** e **não**. Além disso, pode ser incluída na condição um temporizador como uma outra variável. Assim, seria possível escrever logicamente a regra de contexto apresentada na Seção 1 como:

```
TV.estaLigada && !(TV.Local.temAlguem) &&
   Temporizador.tempo == 50min
```

Para interpretar essa regra, o interpretador deve utilizar o *Serviço de Descoberta* para obter a referência da TV e instanciar um novo temporizador para marcar o tempo mínimo que a condição `TV.estaLigada && !(TV.local.temAlguem)` deve permanecer verdadeira. Caso no decorrer da contagem as condições `TV.estaLigada` ou `!(TV.local.temAlguem)` alterem seus valores para falso, o temporizador deve ser interrompido e reiniciado.

As ações são efetuadas pelos **atuadores** (que também são ARs) interessados na regra. Uma regra pode ter um ou mais interessados, que devem ser notificados através de um evento informando que a regra foi avaliada como verdadeira. Essa notificação é realizada através do padrão *publish-subscribe*, largamente utilizado em sistemas distribuídos. Ao

receber a notificação, o AR atuador vai de fato atuar no ambiente por meio de chamadas remotas aos ARs que irão executar as ações. No caso da regra acima mencionada, existirá um AR atuador, que vai obter a referência da TV em questão (passada como parâmetro no evento) e chamar a operação `desligar` da TV. Um esquema geral de como funciona um interpretador de contexto é mostrado na Figura 2.

3. A Interface de Prototipagem de Aplicações Pervasivas

A ferramenta que estamos propondo, chamada de IPGAP (Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas), visa fornecer a seus usuários um mundo em que os dispositivos interajam entre si, e são facilmente acessíveis através de uma API genérica. Um desenvolvedor de aplicações para ambientes ubíquos/pervasivos pode testar suas aplicações no ambiente da IPGAP, de modo a avaliá-las e consequentemente evoluí-las.

Isso seria um desafio se fosse necessário desenvolver toda infraestrutura a partir do zero, ou mesmo adquirir todos os equipamentos necessários. Nesta Seção apresentaremos a IPGAP e discutiremos suas principais funcionalidades e requisitos, além de aspectos de implementação e as possibilidades disponíveis.

3.1. Perfis de Utilização

Quanto à utilização da IPGAP, podemos listar os seguintes requisitos:

- Oferecer ao **desenvolvedor** os recursos necessários para teste e depuração de aplicações pervasivas e um ambiente contendo ARs que encapsulam os principais dispositivos presentes em um ambiente inteligente. O ambiente fornecido é capaz de executar uma grande variedade de cenários, como aplicações de telessaúde, segurança e entretenimento.

- Proporcionar ao **usuário final** a possibilidade de controlar os recursos de sua residência remotamente, através de uma GUI. Dessa forma, pode-se visualizar o estado dos recursos (e.g. canal da TV igual a 44) e atuar sobre eles (e.g. definir a temperatura do ar-condicionado para 17 graus). Entenda-se "usuário final" como um morador do ambiente inteligente, sendo este usuário potencialmente leigo na área da computação.

3.1.1. Possibilidades para o Desenvolvedor

O desenvolvedor tem a capacidade de, através da IPGAP, visualizar o ambiente alvo em uma tela gráfica - por exemplo, um *tablet* - no formato de um mapa, que pode ser populado com os recursos do ambiente necessários à aplicação que está sendo desenvolvida. Por exemplo, para que uma aplicação simples de controle de iluminação seja testada, é necessário que se tenha disponível lâmpadas, 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 (previamente definido em um editor - ver Seção 3.2) e executar sua aplicação. O teste comum para esse exemplo seria movimentar o avatar no mapa, como em um jogo, e observar os sensores detectando a presença ou ausência do avatar e consequentemente as lâmpadas se acendendo e apagando.

Através de aparelhos de fácil aquisição, que podem simular vários dispositivos de um ambiente inteligente, o desenvolvedor está apto a testar diversos cenários para suas aplicações à baixo custo. Por exemplo, nossa prova de conceito foi implementada sobre o sistema operacional Android da Google, sendo implantada em um *tablet*, que foi utilizado para visualização da ferramenta de prototipagem, juntamente com o mapa do ambiente. Os simuladores dos dispositivos (ver Seção 3.3.1) foram instalados em celulares de baixo custo, também rodando Android, e conectados por uma rede sem fio.

A utilização desses aparelhos diminui grande parte do custo inicial do projeto, pois não exige a aquisição de dispositivos reais, como sensores, entre outros dispositivos. Entretanto, a ferramenta 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, permitindo que os dispositivos reais e simulados sejam tratados da mesma forma, e por consequência, facilmente intercambiáveis. É importante salientar que os serviços básicos do sistema - como o *Serviço de Descoberta* e os Interpretadores de contexto vistos na Seção 2 - podem executar em máquinas mais robustas, para assegurar uma boa qualidade de serviço.

3.1.2. Possibilidades 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 útil para o usuário final (conforme dito anteriormente, um usuário não especialista), pois se torna um poderoso controle remoto de todos os dispositivos de sua residência. Assim, o usuário final pode, via uma rede sem-fio, manipular os dispositivos de sua casa bem como criar regras de contexto que atuarão em seu ambiente. A composição de regras se dá através de um editor gráfico, permitindo que as regras sejam compostas de forma intuitiva. Dessa forma, pode-se configurar todo o ambiente de acordo com as preferências do usuário, tudo isso através de um dispositivo móvel, como um *smarphone* 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, porém esse aspecto está fora do escopo deste trabalho (ver Seção 6).

3.2. 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 em questão, inclusive com suas posições aproximadas em relação aos cômodos da casa. Cada cômodo no mapa tem sua área predefinida pelo usuário em um editor de mapas, que também é responsável por definir a aparência da casa representada. Na Figura 3 vemos o mapa do ambiente, populado com alguns recursos rodando em um *tablet*.

Os recursos são mostrados no mapa seletivamente, de acordo com o desejo do usuário. A IPGAP utiliza ícones para representar os recursos do ambiente, cuja posição no mapa é aproximada 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 área de trabalho de sistemas operacionais como Windows, Linux e Android, contendo ícones que podem ser acionados para chamar outros processos (na Seção 3.3 discutiremos o assunto mais detalhadamente).

Figura 3. Visão do mapa do ambiente

3.3. 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 que disponibilizam uma base com milhares de aplicativos que podem ser baixados, muitas vezes gratuitamente, pelos usuários. Dessa forma, pode-se facilmente customizar o ambiente com a instalação de novos *softwares*. Os principais exemplos de lojas desse tipo são a Google Play para Android e a Apple Store para iOS.

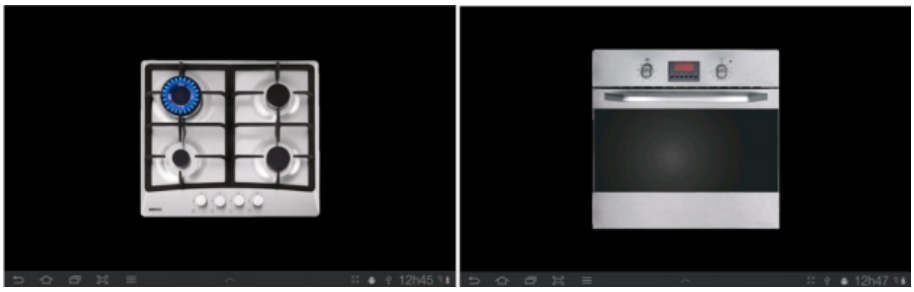


Figura 4. Aplicativo do fogão executando em um *tablet*. Visão das bocas e do forno

No contexto da IPGAP, um aplicativo é um processo que representa um determinado recurso do sistema, o qual pode invocar operações do AR deste recurso (potencialmente em outro *host*) e manipular seu estado interno como se fosse o próprio recurso. 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 basicamente implementa a mesma interface do recurso-alvo, porém sua implementação contém chamadas remotas para este recurso, agindo como um *proxy*. Além disso, um aplicativo contém uma GUI para representar a interface do recurso e exibir seu estado, o que permite

que o recurso em questão seja manipulado e seu estado atual visualizado de forma mais intuitiva.

Imagine um aplicativo capaz de exibir a interface de um fogão (Figura 4), onde o usuário pode através dela, por exemplo, acender uma boca ou o forno do aparelho. Ao acionar-se alguma funcionalidade do aplicativo do fogão, essa operação deve invocar a operação correspondente no AR do fogão real, conforme mostramos no esquema da Figura 5. Note que para isso, o aplicativo do fogão deve possuir um objeto *stub* que encaminha a chamada para o fogão real. Por sua vez, o fogão real deve possuir um AR que exporte sua interface para o ambiente da IPGAP. Um aplicativo pode ser acionado através da IPGAP por meio de sua representação no mapa, acionando seu ícone (ver Seção 3.2). Assim, este aplicativo toma a tela da IPGAP, passando a exibir a interface do dispositivo que representa, como na Figura 4.

Figura 5. Aplicativo do fogão atuando no fogão real

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 deverá ser notificado através de um evento.

3.3.1. Simuladores

Conforme discutimos na Seção 1, é comum que o desenvolvedor necessite de diferentes entidades presentes em um ambiente inteligente (como uma *smart home*) 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 disponibiliza uma série de pequenas aplicações que simulam os principais dispositivos presentes em um ambiente inteligente, como lâmpada, fogão, TV, DVD, 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. O que difere um aplicativo simulador de um aplicativo comum é que 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 - lembrando que do mesmo modo um dispositivo real teria um AR associado para funcionar no nosso sistema. 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.

3.3.2. Instalação dos Aplicativos

Como citado anteriormente, nossa ferramenta permite a instalação de aplicativos na interface de prototipagem, de maneira similar ao que ocorre nos sistemas operacionais Android e iOS. Dessa forma o desenvolvedor tem a possibilidade de estender nossa ferramenta 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.

Um desenvolvedor poderia inclusive criar seus próprios aplicativos e instalá-los na IPGAP. 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. Ligação com Recursos Externos

Recursos reais, externos ao *host* da IPGAP, como eletrodomésticos e sensores podem fazer parte do sistema, enriquecendo assim os testes realizados na aplicação a ser desenvolvida e possibilitando o uso da IPGAP pelos usuários finais para controlar suas residências. Existem duas formas de um dispositivo externo fazer parte do nosso sistema:

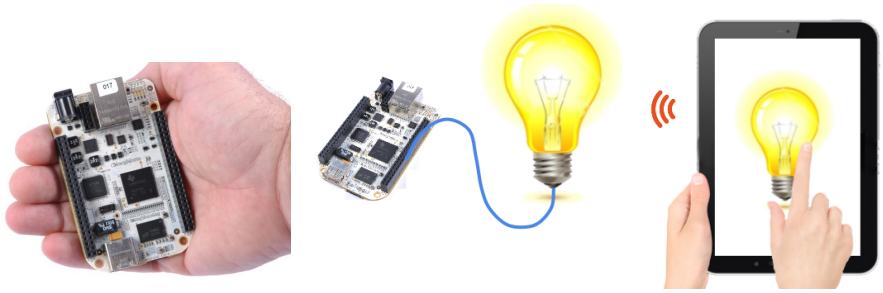
I - Adquirir o aparelho com nosso suporte embutido de fábrica.

II - Desenvolver um *wrapper* que encapsule o aparelho.

Para a possibilidade I, fica claro que o dispositivo conterá algum mecanismo para abrigar um AR que irá expor a interface do recurso para o sistema, além das bibliotecas necessárias para comunicação. Esse mecanismo poderia ser, por exemplo, um chip embutido no próprio *hardware* do dispositivo.

Para a possibilidade II, pode-se desenvolver um componente que implemente ambas as APIs (a API de nossa proposta e a API do recurso desejado). Quando esse componente recebe uma chamada de método através de nossa API, ele encaminha a chamada para o recurso-alvo utilizando a API do fabricante deste recurso. Ou seja, o componente faz a "tradução" de uma chamada do nosso sistema para uma chamada nativa da API do aparelho em questão. Esse componente, chamado de *wapper*, pode ser implementado em *software* ou mesmo em *hardware* ou uma solução híbrida. 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 6. Experimentos com Beaglebone

Nosso grupo está realizando testes com o Beaglebone [BeagleBoard 2012] com o objetivo de construir *wappers* para diversos dispositivos. O Beaglebone (Figura 6(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. Para realizarmos os testes, instalamos o 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 6(b) ilustra essa possibilidade.

Assim, pode-se facilmente incluir as bibliotecas necessárias sobre o sistema Android instalado, inclusive o aplicativo da lâmpada, que contém o AR deste recurso. Dessa forma, uma lâmpada comum pode se registrar no *Serviço de Registro* do sistema, e ser descoberta e utilizada pela IPGAP e aplicações diversas.

Ressaltamos que essa abordagem é ideal para testar aplicações de projetos em desenvolvimento, pois se trata de uma placa genérica, servindo para diversos propósitos, além de poder ser reutilizada, pois é programável. Após a fase de desenvolvimento, pode-se produzir em série toda essa aplicação em chips minúsculos à baixo custo, que poderiam ser embutidos nos dispositivos (inclusive em uma lâmpada) gerando assim a possibilidade I.

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 de nossa proposta, assim como diversos outros aspectos como flexibilidade e usabilidade. Foram implementadas aplicações para controle dos dispositivos da casa (como um controle remoto), controle de iluminação da residência, jogos *multiplayer* onde usuários do ambiente podem descobrir jogadores e jogar em conjunto, entre outras.

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. Estudo de Caso

O usuário final está em casa assistindo a um filme em seu *Blueray* 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 aplicativo) o *Blueray* da sala como fonte emissora da mídia.

Listagem 1. Obtendo referências dos ARs e registrando interesse em variáveis de contexto

```
1  //Obtém lista de todos os ARs registrados
2  List<ResourceAgent> agents = discovery.searchByType(Type.ALL);
3  //Mostra lista na tela e retorna dispositivo selecionado
4  ResourceAgent sourceChoosed = showListOnScreen(agents);
5  //Obtém AR que representa o usu´ario
6  ResourceAgent user = discovery.searchByName("David");
7  //Obtém lista dos sensores de presença da casa
8  List<ResourceAgent> sensors = discovery.searchByType(Type.
       PRESENCE_SENSOR);
9  /* Para cada sensor de presença, se registra como interessado em
10 receber eventos de entrada e saída de pessoas */
11 for (presence : sensors) {
12   presence.registerInterest(this, PresenceAgent.IN);
13   presence.registerInterest(this, PresenceAgent.OUT);
14 }
```

Nesse momento a aplicação obtém as referências dos ARs que representam o *Blueray* da sala e o próprio usuário através do *Serviço de Descoberta*, para respectivamente, saber de onde obter a mídia, e para onde deve transmiti-la (através da posição do usuário). Além disso, a aplicação deve possuir as referências dos ARs dos sensores de presença da casa para assim saber quando o usuário saiu do cômodo. Note que pessoas também são representadas no sistema através de ARs. O AR que representa uma pessoa na verdade agrega diversos outros ARs, que representam sensores corporais, emissores de RF-ID, entre outros dispositivos.

Após obter todas as referências que precisa, a aplicação se registra como interessada em receber determinados eventos destes ARs. Por exemplo, quando o usuário deixar a sala, o sensor de presença correspondente detectará que alguém saiu do cômodo, e notificará os interessados nessa informação através de um evento. Um trecho de código para aquisição das referências dos recursos e registro de interesse em seu contexto é ilustrado na Listagem 1. Este evento deve informar quem é o recurso que originou o evento (neste caso, o sensor de presença da sala) e qual o contexto alterado (o usuário saiu da sala). Nesse caso, a aplicação receberá nesse evento a referência do AR do usuário que saiu do referido cômodo.

Dessa forma, quando o usuário adentra a cozinha, o sensor de presença deste cômodo detecta a chegada do usuário e notifica à aplicação via eventos. A aplicação busca o dispositivo mais próximo do usuário e descobre uma TV na cozinha. Agora a aplicação pode realizar uma chamada remota para o *Blueray* (fonte dos dados) solicitando que ele mude o receptor de vídeo da TV da sala para a TV da cozinha. Isso é feito automaticamente e sem fio, resultando na transferência da mídia para esta TV. Na Listagem 2 podemos ver o modo de como essa busca é feita.

Listagem 2. Consultando recursos por proximidade

```
/* Obtém lista de ARs que representam dispositivos de visualização
(como TVs, tablets, celulares e computadores em geral)
em ordem de proximidade com o usuário. */
List<ResourceAgent> viewCloserToUser =
  discovery.searchByType(Type.VIEW, SearchOptions.PROXIMITY, user);
```

Uma possibilidade de melhoria para esta aplicação seria a utilização das preferências do usuário como critério de busca. Por exemplo, a busca da Listagem 2 poderia retornar como dispositivo de visualização mais próximo o próprio *smartphone* do usuário. Porém este poderia expressar a preferência por dispositivos com telas maiores ou com melhor definição de imagem, mesmo que não seja o mais próximo. Uma busca que utiliza esses critérios seria possível através de algoritmos de otimização combinatória que fariam um balanceamento entre essas duas informações para escolher o melhor dispositivo: a proximidade e a melhor visualização.

5. Trabalhos Relacionados

6. Conclusão e Trabalhos Futuros

Neste artigo apresentamos

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.

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

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

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.

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.

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, New York, NY, USA. ACM.

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.

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