Um *Framework* de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes

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

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

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

Abstract. *Due to recent advances in mobile computing and wireless communication technologies, we can see the emergence of a favorable scenario for building ubiquitous applications. This work proposes a new framework that aims the building of those applications, providing a set of concepts and implementation tools. We propose abstractions that allow developers to handle the resources spread in the environment in a simple and homogeneous way, and to interpret context informations. In order to demonstrate the feasibility of the proposal, the concepts were implemented in a platform named SmartAndroid. An ubiquitous applications prototyping interface was implemented over this platform to allow the testing of different environment configurations before the purchasing of all devices; and also a context rules composition interface, where end users can define their preferences in the environment.*

Resumo. *Com os recentes avanços da computação móvel e nas tecnologias de comunicação sem fio, percebe-se o surgimento de um cenário favorável à construção de aplicações ubíquas. Este trabalho propõe um novo framework para a construção de tais aplicações, provendo um ferramental conceitual e de implementação. São propostas abstrações que possibilitam aos desenvolvedores lidar com os recursos distribuídos no ambiente de maneira simples e homogênea, e interpretar as informações de contexto. A fim de se demonstrar a viabilidade da proposta, os conceitos foram implementados em uma plataforma chamada SmartAndroid. Sobre esta plataforma foi implementada uma interface de prototipagem de aplicações ubíquas onde configurações de ambientes podem ser testadas antes da aquisição de todos os dispositivos; e uma interface de composição de regras de contexto, onde usuários finais podem definir suas preferências no ambiente.*

1. Introdução

A Computação Ubíqua, como proposta por Weiser na década de 1990 [Weiser 1991], descrevia uma mudança no paradigma de interação entre o usuário e os sistemas computacionais. Weiser previu o surgimento do que chamou de "computação calma", onde a interação entre os usuários e os computadores ocorre de forma natural, sem ações explícitas. Uma aplicação ubíqua identifica as necessidades de seus usuários coletando, por meio de sensores, as informações do seu contexto de execução, e as atende provendo serviços, por meio de atuadores, os quais incluem diversos tipos de interfaces.

A construção e a manipulação de aplicações ubíquas representam grandes desafios para desenvolvedores, especialmente em termos do conhecimento técnico exigido e da

disponibilidade de dispositivos reais durante o desenvolvimento da aplicação. Alguns desses desafios podem ser assim destacados: (i) há dificuldades em se estabelecer um protocolo comum de comunicação entre os componentes do sistema distribuído, por conta da *heterogeneidade dos dispositivos envolvidos*; (ii) a interatividade das aplicações ubíquas é dificultada dependendo da quantidade e da *variedade de informações de contexto e serviços* disponíveis no ambiente; (iii) o desenvolvimento e o teste de aplicações exigem uma alta *disponibilidade de recursos*, como por exemplo, sensores (presença, iluminação, temperatura, etc), atuadores (chaves, alarmes, etc), *smart-tvs*, incluindo novos dispositivos embarcados, ou ainda de espaços físicos, tais como uma casa para aplicações do tipo *smart home*.

Muitos trabalhos têm por objetivo definir *frameworks* voltados à construção e ao gerenciamento de aplicações ubíquas [Helal et al. 2005, Cardoso 2006, Ranganathan et al. 2005]. Em [Augusto and McCullagh 2007] são apontados desafios na aquisição de conhecimentos do ambiente. Em [Helal et al. 2005] é proposto um *middleware* entre a camada física (compreendida pelos sensores e atuadores) e a camada de aplicação (onde se encontram o ambiente de desenvolvimento e as aplicações ubíquas). Em [Cardoso 2006] são propostos serviços para gerenciar componentes representativos do ambiente no nível de *middleware*. Há trabalhos ainda que abordam os Ambientes Inteligentes (AmbI) [Augusto and McCullagh 2007], onde uma variedade de dispositivos está disponível, como por exemplo em casas inteligentes (ou *smart home*), com televisores, termômetros, *smartphones*, e outros, os quais podem ser descobertos e configurados de acordo com suas especificidades. O trabalho de citeranganathan2005olympus, em especial, se preocupa em organizar estes componentes de forma a facilitar suas manipulações; a estruturação proposta permite ampliar o escopo de operações de suporte de um sistema ubíquo. Estes trabalhos, no entanto, não têm como foco a integração das questões acima identificadas: *heterogeneidade dos dispositivos*, *variedade de informações de contexto e serviços*, e *disponibilidade de recursos*.

Este artigo apresenta a proposta de um novo *framework* para o desenvolvimento de aplicações ubíquas em AmbI. O objetivo é fornecer suporte à programação, teste e execução de aplicações, permitindo lidar de forma consistente com sistemas de grande complexidade. Este *framework* destaca-se por tratar dos desafios já identificados na computação ubíqua [de Araujo 2003]. A *heterogeneidade de dispositivos* é tratada através da definição de um Modelo de Componentes Distribuídos, no qual o componente básico tem uma estrutura uniforme definida como um Agente de Recurso (AR), proposta inicialmente em [Cardoso 2006] como uma entidade de coleta de informações de contexto. Neste trabalho, o AR, além de manter informações de contexto, age como um componente que encapsula o código do dispositivo a ele associado, incluindo os aspectos de interação com os componentes da aplicação. Para a questão da *variedade de informações de contexto e serviços*, é proposto um Modelo de Contexto que define o armazenamento e a distribuição de tais informações, e provê o suporte para a criação de regras de contexto. Finalmente, em relação à questão sobre a *disponibilidade de recursos*, o *framework* inclui uma aplicação de Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) [Barreto 2012], voltada à visualização e ao teste de aplicações ubíquas, mesclando componentes reais e virtuais.

Os conceitos do *framework* foram concretizados sobre uma plataforma denom-

inada SmartAndroid [1] desenvolvida no contexto do projeto. A implementação no contexto deste projeto tem viabilizado avaliações no sentido de provar conceitualmente que o *framework* facilita o processo de construção de aplicações ubíquas. Uma das avaliações ocorreu durante o processo de transformação de uma aplicação com funcionamento estritamente local em uma aplicação ubíqua. Em outra avaliação, uma aplicação foi construída para verificar a viabilidade de implantação do Modelo de Contexto, por meio da exploração dos mecanismos de comunicação utilizados no Modelo de Componentes Distribuídos.

O artigo está assim organizado: a Seção 2 apresenta os conceitos básicos que orientam o desenvolvimento deste trabalho; a Seção 3 apresenta a arquitetura geral do *framework* incluindo o Modelo de Componentes Distribuídos e o Modelo de Contexto; na Seção 4 é apresentada a IPGAP; a Seção 5 apresenta uma prova de conceito demonstrando a viabilidade da construção de aplicações ubíquas utilizando o *framework* e uma aplicação que explora as principais características do AmbI; a Seção 6 apresenta uma comparação com trabalhos relacionados; e a Seção 7 apresenta as conclusões e trabalhos futuros.

2. Conceitos Básicos

Os *frameworks* para aplicações ubíquas utilizam em geral conceitos de Inteligência Ambiental [Augusto and McCullagh 2007], de Computação Sensível ao Contexto [Dey et al. 2001] e de Prototipagem de Aplicações Ubíquas [Weis et al. 2007]. A Inteligência Ambiental define o espaço onde estas aplicações funcionam. O comportamento de sistemas ubíquos é definido a partir de técnicas aplicadas na Computação Sensível ao Contexto. A prototipagem, por sua vez, é utilizada para manipular e testar o funcionamento do conjunto de aplicações no ambiente.

Os sistemas com enfoque na Computação Ubíqua aplicada no AmbI são baseados geralmente em uma arquitetura em camadas. Na camada inferior encontra-se o espaço físico com seus ocupantes, e em uma camada acima estão os sensores coletando informações de contexto e os atuadores provendo serviços para atender às necessidades destes ocupantes. Uma camada intermediária (*middleware*) é definida entre as tomadas de decisão e as interações com o ambiente, incluindo também conceitos e mecanismos para a construção de software dessa classe de sistemas. As decisões podem ser tomadas por um ocupante ou através de mecanismos de inteligência artificial. Há propostas na literatura que caracterizam este tipo de ambiente, dentre eles o termo "*smart home*" (casa inteligente) se mostra como um dos mais conhecidos [Helal et al. 2005, Augusto and McCullagh 2007, Ranganathan et al. 2005].

O *framework* proposto neste artigo envolve, além de um *middleware* com suporte a serviços, os conceitos de computação sensível ao contexto e de de comunicação e interação, apresentados nas próximas subseções.

2.1. Computação Sensível ao Contexto

O contexto exerce um papel de fundamental importância na computação ubíqua. Sintetizando propostas anteriores, Dey e Abowd [Dey et al. 2001] definiram que contexto é qualquer informação relevante usada para caracterizar a situação de entidades, especificamente: pessoas, lugares e coisas. No AmbI, "Lugares" são os cômodos, os andares de

[1] www.tempo.uff.br/smartandroid

uma edificação, ou espaços em geral que possibilitam a localização de outras entidades; "Pessoas" são indivíduos que povoam o ambiente e interagem com o mesmo; e "Coisas" são representações virtuais de objetos físicos ou componentes de software.

A sensibilidade ao contexto está em se determinar o que o usuário está tentando realizar a partir da aquisição de contexto. Por exemplo, se uma pessoa sai de casa e deixa a torneira aberta, provavelmente a tenha esquecido nesse estado. Neste caso, um sistema sensível ao contexto faz o que qualquer pessoa faria se detectasse esta situação (i.e., fecharia as torneiras). A aquisição do contexto de forma automatizada contribui para a construção de aplicações para AmbI que, de outra maneira, seriam inviáveis, por exigir a entrada de dados ou comandos diretamente por parte dos usuários. O *framework* proposto neste artigo oferece um suporte para a declaração e consulta destas informações (veja a Seção 3.1.1), além da definição de Interpretadores de Contextos (veja a Seção 3.2.2), que facilitam a construção de regras de contexto para atuar no AmbI.

2.2. Comunicação e Interação

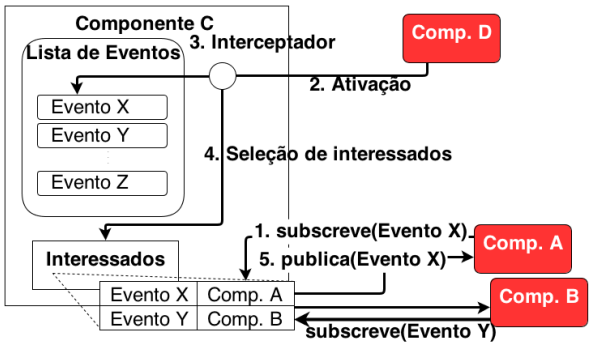

Figura 1. Esquema de *publish-subscribe*

De modo a atender requisitos típicos de ambientes distribuídos, foram incluídos dois mecanismos no *framework* proposto: a invocação remota de procedimentos (*Remote Procedure Call* - RPC) e a comunicação por eventos (ou interesses) seguindo o paradigma *publish-subscribe* [Eugster et al. 2003]. O mecanismo de RPC é utilizado para estabelecer a comunicação síncrona direta entre os componentes do AmbI. Uma entidade invocadora utiliza um *proxy* da entidade invocada, nele estando contida a sua interface de chamadas de procedimentos públicos. A chamada e o retorno do procedimento são enviados através de mensagens serializadas (no SmartAndroid através da notação JSON). Este mecanismo pode ser usado para enviar comandos aos dispositivos, por exemplo, fechar uma torneira, ou alterar remotamente o *setup* de um ar-condicionado.

O paradigma *publish-subscribe* é utilizado no *framework* proposto para a aquisição do status de componentes do ambiente. A Figura 1 ilustra sua inserção no esquema de componentes. Inicialmente, os Componentes A e B se subscrevem aos Eventos X e Y, respectivamente, do Componente C (Passo 1) e esta subscrição é armazenada no campo de Interessados. Posteriormente, o Componente D envia um comando, que ativa o Evento X (Passo 2). Em seguida, ocorre a interceptação (Passo 3) e o Componente A, interessado neste evento (Passo 4), é notificado (Passo 5).

Como suporte, a comunicação no *framework* necessita de uma infraestrutura básica de rede. Por conveniência, foi adotado Wi-Fi, que garante o mínimo de segurança

contra acesso de agentes externos à rede do ambiente e agrega benefícios de mecanismos de criptografia, como a WPA.

3. Descrição do *Framework*

O *framework* provê facilidades e padrões de programação tipicamente requeridos em aplicações sensíveis ao contexto focadas em AmbI. Dentre as facilidades está a capacidade de abstrair detalhes das partes da aplicação que envolvem a comunicação, a aquisição de contexto e a localização de recursos. Como abstração aos recursos, são utilizados componentes chamados Agentes de Recursos (AR). O AR é a unidade básica de modularização sendo utilizado na modelagem, implementação e gerenciamento das aplicações; por exemplo: sensores de temperatura da casa; sensores de vazamento de gás na cozinha; um medidor de pressão arterial; um atuador para fechar as janelas; etc.

Para consolidar a proposta do *framework* seus conceitos e mecanismos foram implementados numa plataforma que denominamos *SmartAndroid*. Essa opção se beneficia da acessibilidade da tecnologia *Android* [Saha 2008] facilitando a integração de dispositivos disponíveis em seu ecossistema e o desenvolvimento de sistemas embarcados. Isto permitiu um desenvolvimento rápido da plataforma facilitando a experimentação de aplicações ubíquas sofisticadas. Deve ser ressaltado que o *framework* usa conceitos de implementação imediata em outras plataformas distribuídas atuais. Adicionalmente, a técnica de *Wrappers* pode ser usada para integrar componentes de outras tecnologias em aplicações *SmartAndroid*. A arquitetura do *framework* é definida a partir de um Modelo de Componentes Distribuídos (Seção 3.1) e de um Modelo de Contexto (Seção 3.2).

3.1. Modelo de Componentes Distribuídos

Como visto anteriormente na Seção 2, sistemas para AmbI são geralmente estruturados nas camadas física, de *middleware* e de aplicação (Figura 2). O Modelo de Componentes Distribuídos segue esta estrutura para mapear as aplicações ubíquas no AmbI. Neste Modelo são definidos: a estrutura do AR e o Suporte ao Gerenciamento de Agente de Recursos (SGAR).

O conceito de AR foi definido para padronizar os componentes de interação do ambiente, e dessa forma resolver a questão da *heterogeneidade dos dispositivos*. A sua estrutura geral, como ilustrada na Figura 2 (a), é composta por nome único (“tvFamília”), hierarquia de tipos (“\Visual\TV”), localização corrente (“Sala”), classe (classe TV) com variáveis e métodos, e uma lista de interessados em informações de contexto. A hierarquia de tipos, que é composta por uma sequência de classes, foi inspirada na ontologia descrita em [Ranganathan et al. 2005] para classificar entidades (como os ARs). Esta hierarquia foi organizada para possibilitar a consulta e instanciação de ARs a partir de propriedades associadas a uma classe específica. Na Seção 3.1.1 são apresentados serviços que se beneficiam desta estrutura do AR.

A lista de interessados do AR é utilizada pelo Modelo de Contexto (ver Seção 3.2) para notificar outras instâncias sobre sua mudança de estados. Na definição deste modelo é visto que um AR armazena o seu estado por meio de seus atributos declarados e os expõe através da criação de Variáveis de Contexto (VC); as mudanças nas VCs podem ser provocadas por Operações (OP). Do ponto de vista da arquitetura, as VCs são as portas de saída de informação, e as OPs são as portas de entrada de comandos.

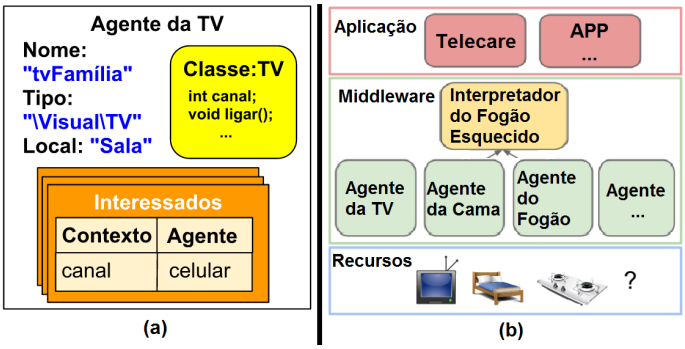

Figura 2. Arquitetura do *Framework*: (a) Estrutura do AR; (b) Camadas

Tabela 1. Operações do SGAR

Serviço	Operação	Argumentos	Descrição
SRR	*register*	Dados do AR	Insere dados do AR no Repositório
SRR	*unregister*	Dados do AR	Remove AR do Repositório
SDR	*search*	Consulta	Busca ARs por tipo, local e/ou nome
SLR	*getPlaces*	Nenhum	Retorna a lista de espaços do Mapa
SLR	*searchByProximity*	Posição	Retorna ARs mais próximos
AR	*subscribe*	AR e VC	Adiciona interesse do AR no VC
AR	*publish*	VC e Valor	Notifica ARs interessados no VC

Na Figura 2 (b) são apresentados componentes típicos de uma *smart home*. Na camada de recursos estão componentes físicos ou conceituais. Os recursos conceituais são aqueles simulados através de outro tipo de dispositivos (por conveniência usamos *smartphones* e *tablets*) ou criados através da IPGAP. A figura apresenta como exemplos de recursos um televisor, uma cama e um fogão. No *middleware* são apresentados os ARs representativos destes recursos, além de entidades de interpretação de contexto, que são definidas no nível da aplicação, mas com o suporte da camada intermediária. O interpretador descrito na figura verifica se o ocupante da casa esqueceu o fogão ligado e, para isso, avalia se o ocupante está deitado na cama e se o fogão está ligado durante determinado tempo; uma vez passado o tempo limite, ações são disparadas para acordar o ocupante, ou para desligar o fogão, atendendo às preferências do usuário. Os ARs referentes à cama e ao fogão coletam o status destes recursos físicos e os divulgam para ARs interessados. Isto permite que estas informações sejam utilizadas pelos componentes da camada da aplicação. Ainda no *middleware*, o SGAR (ver Seção 3.1.1) é definido como suporte para aplicações criarem, consultarem e instanciarem ARs.

3.1.1. Suporte ao Gerenciamento de Recursos

O SGAR é o responsável por gerenciar o conjunto de ARs no AmbI. O controle de registro e descoberta de ARs segue a ideia geral sobre serviços que manipulam recursos apresentada em [Cardoso 2006]. Neste trabalho, o SGAR é composto por três componentes, ou serviços básicos: o Serviço de Registro de Recurso (SRR), o Serviço de Descoberta de Recursos (SDR) e o Serviço de Localização de Recursos (SLR). A Tabela 1 destaca as principais operações de cada serviço de suporte. O SRR é utilizado para registrar (*register*)

ou remover (*unregister*) ARs no ambiente, o SDR localiza recursos para permitir o acesso remoto (*search*) e o SLR os localizam através de posições físicas e indica os recursos mais próximos (*searchByProximity*). Estas operações manipulam dados representativos dos respectivos ARs no Repositório de Recursos. As operações "*subscribe*" e "*publish*" são comuns a todos os ARs e funcionam com base no mecanismo de comunicação por eventos descrito na Seção 2.2. Os componentes (A, B, C, D) na Figura 1 correspondem aos ARs e os Eventos correspondem a mudança de estado das VCs.

O Repositório de Recursos, como apresentado na Figura 3, possui o Diretório de Recursos e o Mapa, o qual contém representado o conjunto de espaços físicos do AmbI. O Diretório contém os nomes das instâncias de AR existentes no ambiente, armazenados conforme seus respectivos tipos. No caso da figura há o tipo TV com a "tvFamília" e o tipo *tablet* com o "iPad" e o "Galaxy". Os dados consistem do nome, tipo, localização e a referência de acesso. A referência de acesso permite a interação de instâncias de ARs através do RPC (Seção 2.2). Cada espaço do Mapa possui uma área e um nome. No caso de uma *smart home* os cômodos da casa representam estes espaços. Cada entrada do Mapa referencia o conjunto de ARs contidos no respectivo espaço. ARs representado dispositivos com mobilidade têm suas referências atualizadas dinamicamente.

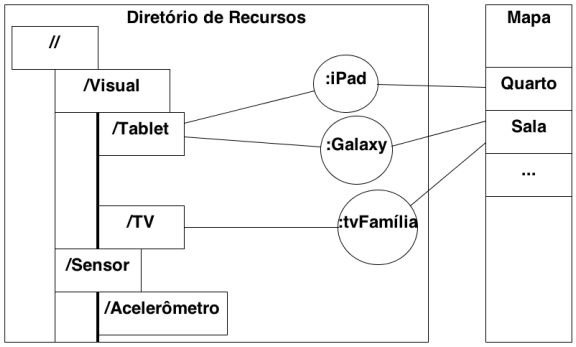

Figura 3. Repositório de Recursos

3.2. Modelo de Contexto

O Modelo de Contexto foi elaborado com o objetivo de tratar a questão da *variedade de informações de contexto e serviços* presentes no AmbI. Neste modelo, o *framework* utiliza uma abordagem flexível, onde o mecanismo de comunicação por eventos (visto na Seção 2.2) possibilita que ARs divulguem mudanças de contexto. A informação pode estar em qualquer lugar do AmbI, contudo, é provido um nível de abstração tal que a subscrição ou consulta à qualquer informação de contexto é feita de maneira padronizada. O processo de aquisição do contexto envolve a descoberta do AR de interesse através dos serviços da SGAR, com a posterior subscrição do mesmo. A abordagem adotada é flexível de tal maneira que, caso seja conveniente, pode ser criada uma unidade centralizadora para gerenciar as informações de contexto, segundo uma arquitetura de memória compartilhada do tipo *blackboard* (ou quadro-negro) [Winograd 2001].

3.2.1. Variáveis de Contexto e Operações

As informações de contexto respectivas de cada AR são expostas através de Variáveis de Contexto (VC). Por exemplo, um agente para uma *smart-tv* pode prover VCs para: qual a programação que está sendo exibida, qual a programação agendada, se a própria televisão está ligada, se está gravando alguma programação, e outras. Em outros termos, tudo o que diz respeito ao estado da televisão podem efetivamente ser coletado.

Uma Operação (OP) tem o papel de expor uma funcionalidade (ou serviço) do AR, possibilitando às aplicações interagirem ativamente no ambiente. Por exemplo, uma televisão integrada ao sistema pode oferecer OPs para desligá-la, mudar de canal, gravar alguma programação, mostrar uma mensagem na tela, perguntar algo ao usuário, gerar um alerta, pausar a programação, etc.

Tanto VCs como OPs definem interfaces, ou portas do AR. Diferentes aplicações, instaladas no mesmo ambiente podem usar estas interfaces para interagir com o próprio ambiente. A Figura 4 representa a subscrição às VCs "Em uso" do AR da cama e "Ligado" do AR do fogão. Na mesma figura pode-se observar a atuação no ambiente ao se utilizar as OPs "Mostrar mensagem" da televisão e "Disparar" do despertador. O Interpretador e o Atuador também representados na figura serão descritos na Seção 3.2.2.

As questões de segurança, inerentes ao problema, são resolvidas com duas técnicas: a própria segurança da rede e com a criação de domínios dentro de um AmbI. A segurança da rede (criptografia de pacotes em redes LAN, como WPA2 e WEP) evita que aplicações estrangeiras possam acessar os recursos de um ambiente. Domínios podem ser usados para isolar aplicações específicas dentro do AmbI, como, por exemplo, uma aplicação de monitoramento de pacientes [Carvalho et al. 2010].

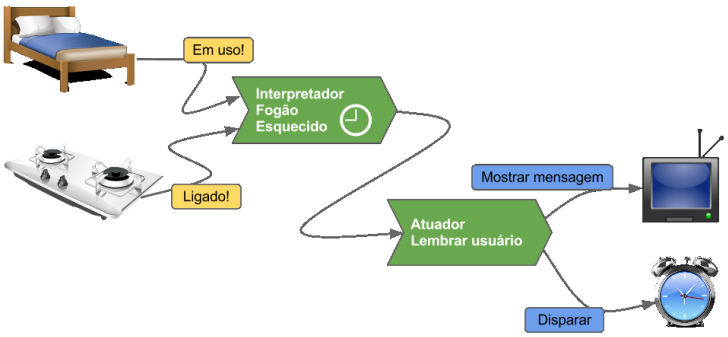

Figura 4. Interpretador de Regra

3.2.2. Interpretação de Contexto

A interpretação de contexto tem a função de agregar informações de contexto provenientes de diferentes fontes, considerando também a passagem de tempo, e avaliá-las segundo alguma lógica específica. O suporte à interpretação de contexto pelo *framework* possibilita uma separação de interesses, onde os desenvolvedores abstraem a implementação de regras de contexto e focam na lógica da aplicação.

A interpretação do contexto é desempenhada por entidades chamadas Interpretadores de Contexto (IC), que avaliam essas informações e notificam agentes atuadores interessados. Agentes atuadores são quaisquer ARs que se subscrevem em ICs para desempenhar ações; sejam estas ações no nível de software (e.g., guardar no histórico, enviar para um servidor remoto) ou no nível do ambiente, ao se chamar OPs de outros ARs (e.g., mostrar uma mensagem na televisão, disparar o despertador, mudar a temperatura do ar-condicionado).

Considere a seguinte regra simples: se uma pessoa (que mora sozinha) ligou o fogão, se deitou na cama, e passaram-se 15 minutos, então dispare o alarme do despertador e mostre na televisão a mensagem "O fogão foi esquecido ligado!". Este exemplo poderia ser implementando no sistema como representado na Figura 4. O IC recebe notificações da TV e do fogão com valores atualizados das VCs "ligado" e "em uso", respectivamente, e resolve internamente a temporização monitorada destas VCs. Uma vez que o IC tenha avaliado a regra como verdadeira durante o tempo de 15 minutos (previamente declarado), ele notifica o atuador "Lembrar usuário". O atuador, por sua vez, invoca as OPs "Mostrar mensagem" da TV e "Disparar" do despertador.

O IC pode ser encapsulado em um AR, assim estendendo suas funcionalidades. A arquitetura do IC é composta de um módulo que recebe as notificações e atualiza os valores em *cache*; uma estrutura de dados em árvore que não só armazena as referências para as VCs e os valores atualizados, mas também a lógica da regra; um módulo que avalia a árvore a cada atualização de valores; e um módulo que gerencia os temporizadores, controlado pelo módulo de avaliação.

A interpretação de contexto visa não só a construção de regras de contexto por parte das aplicações, mas também a definição das preferências dos usuários finais no sistema. Para isto, está sendo desenvolvida uma GUI que possibilitará a usuários sem experiência técnica a criar, editar, desabilitar, etc, regras para o seu dia-a-dia. A GUI permitirá que, com poucos toques, um usuário possa selecionar ARs em um mapa da casa, escolher as VCs, comparar com valores ou outras VCs (operadores de comparação: $=$, $\neq, <, >, \leq$ e $\geq$), montar a expressão lógica (operadores lógicos: "*E*", "*OU*", "*NÃO*"), e definir um conjunto de ações a serem desempenhadas no sistema. A definição do conjunto de ações pode ser feita tanto ao se escolher ARs atuadores para entrar em ação (e.g., Atuador Lembrar Usuário), como selecionando ARs no mapa e suas OPs em seguida (e.g., ar-condicionado - mudar temperatura - 20^{o}). Através do mecanismo de subscrição os ICs notificarão os ARs atuadores, que executarão tarefas no AmbI através de chamadas RPC.

4. Prototipagem de Aplicações Pervasivas

A prototipagem de aplicações pervasivas é fundamental para a depuração e teste em um AmbI. A questão de *disponibilidade de recursos* é resolvida através da IPGAP [Barreto 2012], um aparato ferramental que permite ao desenvolvedor de aplicações ter acesso a estes benefícios. Além disso, a IPGAP disponibiliza uma Interface Gráfica de Usuário (*Graphic User Interface* – GUI) que proporciona ao usuário final controlar remotamente recursos do ambiente.

Na Figura 5 é ilustrado um *tablet* executando o aplicativo da IPGAP e *smartphones* emulando recursos como TV, lampada e termômetro. A IPGAP faz o ma-

Figura 5. Interface de Prototipagem

peamento do ambiente e permite visualizar e instanciar cada um destes elementos e representá-los na sua GUI. Outros recursos, além dos representados pelos *smartphones*, podem ser simulados virtualmente por componentes de software. Assim, o desenvolvedor de aplicações não fica limitado pela disponibilidade de recursos físicos podendo simular os elementos indisponíveis.

O *framework* proposto permite criar aplicações de forma liberal, nas dimensões e quantidades desejadas para um ambiente real. Através da IPGAP, as restrições e viabilidade do funcionamento das aplicações desenvolvidas podem ser verificadas em ambiente próximo ao real. Isto permite avaliar o desempenho das implementações antes da sua implantação efetiva.

5. Avaliação

A avaliação do *framework* foi realizada em duas fases. Primeiro foi avaliado o fator de transparência de comunicação na construção de aplicações ubíquas. Esta avaliação ocorre através do processo de transformação de uma aplicação estritamente local Android em uma aplicação ubíqua através do SmartAndroid. Depois é avaliada a transparência no desenvolvimento de aplicações sensíveis ao contexto e manipulação das informações de contexto promovida pelo Modelo de Contexto.

5.1. Prova de Conceito do *Framework*

Para a primeira avaliação utilizou-se a aplicação do jogo da velha como base. Em sua versão Android (Figura 6), os jogadores interagem através de uma mesma tela de dispositivo (*tablet* ou *smartphone*). Na versão desenvolvida com o SmartAndroid os jogadores interagem sobre um mesmo tabuleiro em dispositivos diferentes. A Figura 6 mostra alguns detalhes do modelo de aplicação da versão Android e apresenta modificações sobre esta estrutura na versão SmartAndroid. Na versão original destacam-se os seguintes componentes: o Motor do Jogo (Motor), a estrutura do Tabuleiro e a identidade dos Jogadores que pode ser do tipo “X” ou “O” (Tipo <X,O>). O Motor atualiza o estado do Tabuleiro que é mostrado na GUI da aplicação. O Tipo do Jogador é verificado para indicar qual simbolo da jogada é marcado no Tabuleiro (“X” ou “O”).

Na versão SmartAndroid as principais estruturas do jogo da velha são encapsuladas no AR do jogo. O AR do Jogo é responsável por interceptar as indicações de jogadas dos respectivos jogadores representados no dispositivo (“X” ou “O”) e encaminhar para o outro jogador. O outro jogador recebe estas jogadas através de seu AR que as encaminha

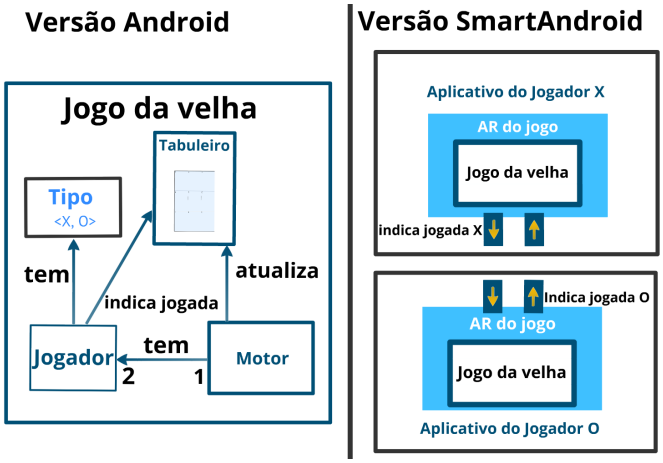

Figura 6. Comparação entre os modelos das aplicações Android e SmartAndroid

ao Motor de jogo que efetiva a atualização do tabuleiro com a jogada do adversário. A etapa de inicialização da nova aplicação foi desenvolvida a partir das operações definidas na Tabela 1. O Código 1 apresenta a descoberta de ARs de outros jogadores na linha 1, o registro do AR do jogador local na linha 2, e a subscrição de interesses entre ARs de jogo do AmbI nas linhas 5 e 6. Após isto, nenhuma alteração no nível de comunicação ou contexto torna-se necessária. Logo, podemos comprovar que o uso de AR tornou transparente a programação da aplicação ubíqua.

Código 1. Inicialização do jogo da velha no SmartAndroid

```
1 List ARs = SDR.search(TYPE, ARJogo) \\Consulta por tipo
2 SRR.register(arJogo) \\Registro do Agente do Jogo
3 if (ARs.size>0): \\Caso haja outros jogadores...
4     for (iAR : ARs): \\Subscrição entre ARs
5         iAR.subscribe("indica jogada", arJogo)
6         arJogo.subscribe("indica jogada", iAR)
```

5.2. Aplicação de Controle de Iluminação Residencial

Dentre os protótipos construídos como prova de conceito, foi criada a Aplicação de Controle de Iluminação Residencial (*Smart Light Controller* – SmartLiC) com o intuito de promover economia no consumo de energia elétrica reduzindo gastos com iluminação. Através de sensores de presença, identifica-se a presença/ausência de pessoas no cômodo, conforme a pessoa sai de um cômodo e passam-se T_{max} minutos sem que ela volte, então a luz daquele cômodo é desligada.

A Figura 4 representa um caso simples de utilização de interpretadores, onde este contém uma regra apenas e as atuações ocorrem em separado, desempenhadas pelos agentes atuadores. A fim de se prover maior flexibilidade para os desenvolvedores, o *framework* possibilita também que a interpretação de contexto seja desenvolvida no nível da aplicação, contudo, aproveitando o mecanismo de comunicação por eventos e a compilação da regra. Note que esta abordagem possibilita a utilização de um motor de regras (e.g., Jess, Drools), se for conveniente.

O Código 2 ilustra uma possível implementação da interpretação de contexto pela aplicação SmartLiC. Na etapa de inicialização, a aplicação utiliza o SLR para localizar

os cômodos, e obter a lâmpada e o sensor de presença de cada um, utilizando a busca do SDR. Com a referência para os ARs ("lampAR" e "presAR") e as ações a serem desempenhadas (desligar a lâmpada referenciada – "lampAR.turnOff"). Para evitar a repetição do código da regra, as referências são guardadas em separado da função da regra, na lista de configurações ("configList"). A regra é criada em "rule" como: *lâmpada ligada E o cômodo desocupado POR T_{max} unidades de tempo*, desempenhando uma relação lógica entre as diferentes VCs e a temporização da regra. Em seguida, subscreve-se às VCs dos ARs de interesse, começando o processamento da regra.

Na etapa de avaliação, também descrita no Código 2, é definido um bloco de comando para o tratamento das notificações recebidas dos ARs de interesse (lâmpadas e sensores de presença subscritos). Uma vez recebida uma atualização de uma VC de um dos ARS ("ar_ref"), é obtida a configuração referida, e seu valor em mantido *cache* é atualizado. Segue-se a avaliação da regra ("*evaluate*"), utilizando a configuração obtida, e considerando os temporizadores. A etapa de avaliação é bloqueante no caso da existência de temporizadores associados. A reação, em caso de validade da regra é desempenhada pela invocação das ações (OPs) registradas na configuração.

Código 2. Interpretação de Contexto no SmartLiC

```
1  Inicialização
2  placeList = SLR.getPlaces() Obtém cômodos
3  for each place from placeList: Seleciona um cômodo da lista
4      lampAR = SDR.search(Lamp, place) Obtém lâmpada do cômodo
5      presAR = SDR.search(Presence, place) Obtém sensor de presença do cômodo
6      configList.add(lampAR, presAR, lampAR.turnOff) Guarda configuração
7  rule = Lamp.on ∧ ¬(Presence.occupied) ∧ T > T_max Função de regra
8  subscribeTo(configList.getARs()) Subscreve às lâmpadas e sensores de presença
9
10 Avaliação da regra de contexto
11 on event received (ar_ref, vc, value): Quando chega uma notificação
12     config = configList(ar_ref, vc) Obtém configuração
13     config.value = value Atualiza cache
14     if (evaluate(rule, config)) then: Avalia regra com configuração
15         invoke(config.getActions()) Invoca ações
```

O SmartLiC é um projeto simples, mas que possibilita a avaliação de boa parte do ferramental proposto pelo *framework* e implementado no SmartAndroid. O foco tomado na implementação foi economia de energia, porém, outras funcionalidades podem ser agregadas ao serviço. Por exemplo, um sistema do tipo "engana ladrão", onde a aplicação acende lâmpadas da casa seguindo alguma ordem pré-estabelecida ou informações de um histórico, a fim de se enganar possíveis invasores quando os proprietários estão ausentes.

6. Trabalhos Relacionados

Em [Helal et al. 2005] é proposta uma arquitetura de camadas semelhante ao já apresentado neste trabalho. Além destas, há a camada de conhecimento que tem função similar ao SGAR, e a camada de contexto que tem função similar ao Modelo de Contexto, mas atua em uma granularidade maior. Nosso *framework* descreve mecanismos para construção e instalação de novas aplicações, algo que em [Helal et al. 2005] não fica evidente sobre como pode ser feito. Um conjunto de operações de alto nível para ambientes inteligentes (espaços ativos) é proposto em [Ranganathan et al. 2005]. As operações

básicas são semelhantes as funções de um sistema operacional, só que ao invés de manipular recursos, entidades do ambiente são manipuladas. Em nosso *framework* há as funções apresentadas na Tabela 1. Outras funções como parar, iniciar, suspender, reiniciar são definidas por cada AR e aplicação ubíqua do sistema, permitindo uma distribuição do controle de serviços do ambiente.

Boa parte dos *frameworks* propostos evita levantar pontos sobre a sobrecarga de comunicação ocorrida em sistemas ubíquos. O artigo [Villanueva et al. 2009] é uma proposta com abordagem distribuída para a parte de suporte (SGAR), onde cada componente possui o suporte replicado e que utiliza comunicação *multicast* para realizar os serviços. O custo de espaço desta abordagem é muito alto devido a ocorrência de replicação de dados sobre o ambiente, e acaba ocasionando uma sobrecarga no consumo de energia devido ao excesso de comunicação. O JaCa-Android [Santi et al. 2011] utiliza o esquema de captura de mudança de estado em recursos através de escuta de eventos, semelhante a comunicação por eventos apresentada na Seção 2.2, mas limitado a componentes em um mesmo dispositivo. Além de não ser voltado a sistemas distribuídos, esta proposta tem sua abordagem limitada pela a plataforma Android por utilizar mecanismos específicos da tecnologia para a captura de eventos.

O *framework* busca como diferencial que novas aplicações ubíquas possam ser concebidas de forma dinâmica e adaptativa. A proposta disponibiliza ao desenvolvedor de aplicações um ferramental para montar AmbIs personalizados. O desenvolvedor adquire como benefício a separação de interesses, que reduz a carga de codificação de requisitos essenciais de um sistema (e.g. segurança, manutenção), e possui ainda um suporte para construção de regras sobre o contexto do ambiente, permitindo prover em alto nível serviços aos usuários do sistema inteligente.

7. Conclusão e Trabalhos Futuros

Neste artigo, foi apresentado o *Framework* de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes, com abstrações que visam facilitar o desenvolvimento e a implantação de aplicações que interajam de forma ubíqua com o ambiente e seus ocupantes. Através do *framework*, o desenvolvedor cria serviços que atendam a requisitos preestabelecidos de acordo com as características de um AmbI. Após a concepção de serviços para o AmbI, a implantação ocorre a partir da instalação do conjunto de aplicações sobre uma infraestrutura de rede Wi-Fi segura. Os serviços, após implantados, podem ser manipulados pelos ocupantes através da IPGAP e das interfaces de operações de cada aplicação instalada. Na avaliação, que se deu por meio da implementações das aplicações do jogo da velha com múltiplos participantes e do SmartLiC, foi constatada a qualidade do *framework* em lidar com problemas de um AmbI.

A proposta possui potencial para ampliar suas funcionalidades e atender a outros desafios importantes na área de computação ubíqua e computação sensível ao contexto. A segurança atual está limitada às configurações da rede Wi-Fi, futuramente pretende-se agregar ao suporte restrições de acesso a determinados ocupantes e entre diferentes domínios de aplicações. A economia de energia em dispositivos pode ser aprimorada reduzindo a sobrecarga de comunicação em quantidade e qualidade. Na parte de Computação Sensível ao Contexto, pretende-se identificar as preferências do usuário de forma menos intrusiva através de técnicas de mineração de dados aliadas à aprendiza-

gem de máquina. Para permitir um controle de serviços maior no nível do usuário, está sendo definida uma interface amigável para a definição de regras de contexto. Como resultado desses incrementos, aplicações críticas como um sistema de assistência domiciliar a saúde [Carvalho et al. 2010] serão beneficiadas.

Referências

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

Barreto, D. (2012). Interface de prototipagem e gerenciamento de aplicações pervasivas. Dissertação de mestrado em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro.

Cardoso, L. (2006). 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, Niterói, Rio de Janeiro.

Carvalho, S., Erthal, M., Mareli, D., Sztajnberg, A., Copetti, A., and Loques, O. (2010). Monitoramento remoto de pacientes em ambiente domiciliar. *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-Salao de Ferramentas, Gramado, RS, Brasil*.

de Araujo, R. (2003). Computação ubíqua: Princípios, tecnologias e desafios. In *XXI Simpósio Brasileiro de Redes de Computadores*, volume 8, pages 11–13.

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

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

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., and Jansen, E. (2005). The gator tech smart house: A programmable pervasive space. *Computer*, 38(3):50–60.

Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R., and Mickunas, M. (2005). Olympus: A high-level programming model for pervasive computing environments. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 7–16. IEEE.

Saha, A. (2008). A Developer's First Look At Android. *Linux For You*, (January):48–50.

Santi, A., Guidi, M., and Ricci, A. (2011). Jaca-android: an agent-based platform for building smart mobile applications. *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, pages 95–114.

Villanueva, F., Villa, D., Santofimia, M., Moya, F., and Lopez, J. (2009). A framework for advanced home service design and management. *Consumer Electronics, IEEE Transactions on*, 55(3):1246–1253.

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*, 265(3):94–104.

Winograd, T. (2001). Architectures for context. *Human-Computer Interaction*, 16(2):401–419.