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. *With recent advances in mobile computing and wireless communication technologies, we can see the emergence of a favorable construction of ubiquitous applications. This paper proposes a new framework for building such applications, providing a conceptual and implementation tools. We propose abstractions that allow developers to handle the resources distributed in the environment and interpret the context informationin in a simple and homogeneous way. In order to demonstrate the feasibility of the proposal, the concepts were implemented in a platform called SmartAndroid. This platform also includes an prototyping interface for ubiquitous applications, where settings can be tested before the acquisition of all devices, and interface composition of contextual rules, where end users can set preferences on 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 testados 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 90 [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 explicitas. Uma aplicação ubíqua identifica estas necessidades naturais obtendo informação de contexto através de sensores, e as atendem provendo serviços através de atuadores. Este tipo de sistema de aplicações é associado a um espaço denominado ambiente inteligente (AmbI) [Augusto and McCullagh 2007].

A construção e manipulação de aplicações ubíquas representam um grande desafio para desenvolvedores com pouco conhecimento técnico e recursos escassos. Alguns problemas estão em maior evidência, como os a seguir. Na construção e teste de aplicações surge a necessidade de um contingente de recursos como dispositivos embarcados e espaço físico. Também há uma dificuldade de estabelecer um protocolo comum de comunicação entre os componentes do sistema distribuído. E por fim, a interatividade das aplicações ubíquas é dificultada dependendo da quantidade e variedade de informações de contexto disponível no ambiente. Estes são alguns dos desafios que motivam a proposta de um *framework* com o objetivo de facilitar a aplicação dos conceitos de computação ubíqua em ambientes inteligentes.

Muitos trabalhos têm por objetivo definir, no estado da arte, um arcabouço de técnicas para construção e gerenciamento de aplicações ubíquas [Helal et al. 2005, Cardoso and Loques 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 and Loques 2006] são propostos serviços para gerenciar componentes representativos do ambiente no nível de *middleware*. Uma variedade de dispositivos e propriedades é encontrada no AmbI. Como exemplo há a casa inteligente (ou *smarthome*), onde televisores, termômetros, *smartphones*, e outros, podem ser encontrados inseridos em cômodos com suas especificidades. O trabalho em [Ranganathan et al. 2005] se preocupa em organizar estes componentes de forma a facilitar suas manipulações. Esta estruturação permite ampliar o escopo de operações de suporte de um sistema ubíquo.

Neste trabalho é proposto um novo *framework* para o desenvolvimento de aplicações ubíquas em AmbI. Objetiva-se dar suporte à programação, teste e execução de aplicações, permitindo lidar de forma consistente com sistemas de grande complexidade. Este *framework* destaca-se por cobrir grande parte dos desafios destacados na computação ubíqua [de Araujo 2003], como o tratamento da heterogeneidade de dispositivos, o tratamento de informações de contexto e a descoberta de serviços. A heterogeneidade 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). Segundo [Cardoso and Loques 2006], AR é a entidade de coleta de informações de contexto. Neste trabalho, esta definição é ampliada para qualquer módulo que interage com elementos deste modelo de componentes. Para o tratamento de informações de contexto é proposto o Modelo de Contexto que define o armazenamento e distribuição de tais informações, e provê suporte para a criação de regras de contexto. Além disso, no *framework* é proposta a aplicação de Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) (Trabalho submetido ao SBRC 2013) que permite a visualização e o teste de aplicações ubíquas, as quais mesclam componentes reais e virtuais.

Os conceitos do *framework* foram efetivados sobre a plataforma Android através do projeto SmartAndroid (*www.tempo.uff.br/smartandroid*). Neste projeto foi possível avaliar a qualidade de suporte do *framework*. Esta avaliação ocorreu durante o processo de transformação de uma aplicação com funcionamento estritamente local em uma aplicação

ubíqua. Com isso foi possível provar conceitualmente que o *framework* facilita o processo de construção de aplicações ubíquas. Outra avaliação foi realizada para testar a viabilidade de implantação do Modelo de Contexto, foi então construída uma aplicação que explora os mecanismos de comunicação utilizados no Modelo de Componentes Distribuídos.

O restante deste artigo é organizado da forma a seguir. 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 uma prova de conceito demonstrando a viabilidade da construção de aplicações ubíquas utilizando o *framework* e é demonstrada uma aplicação que explora as principais características do AmbI. Após a avaliação, a Seção 5 apresenta uma comparação com trabalhos relacionados. E para finalizar, a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Conceitos Básicos

Os *frameworks* para aplicações ubíquas costumam utilizar os conceitos de Inteligencia 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 Inteligencia Ambiental define o espaço onde estas aplicações funcionam. Os comportamentos de sistemas ubíquos são definidos a partir de técnicas aplicadas na Computação Sensível ao Contexto. E a prototipagem é utilizada para manipular e testar o funcionamento do conjunto de aplicações no ambiente.

Os sistemas com enfoque na Computação Ubíqua aplicada no AmbI geralmente possuem uma arquitetura em camadas como base. Na camada mais profunda encontra-se o espaço com seus ocupantes, em uma camada acima estão os sensores captando suas interações e atuadores promovendo ações com estes indivíduos. Uma camada intermediária (*middleware*) é definida entre as tomadas de decisões e as interações com o ambiente. As decisões podem ser tomadas por um ocupante ou através de mecanismos de inteligencia artificial. Há diversos exemplos na literatura que utilizam especificações deste tipo de ambiente, dentre eles o termo "smart home" (casa inteligente) se mostra como um dos mais populares [Helal et al. 2005, Augusto and McCullagh 2007, Ranganathan et al. 2005]. Além do *middleware* como suporte a serviços, nosso *framework* inclui o Modelo de Componentes Distribuídos [Brown and Kindel 1998] (Seção 3.1), conceitos e mecanismos de comunicação (Seção 2.3), e o Modelo de Contexto (Seção 3.2) que segue o conceito de Sensibilidade ao Contexto apresentado na Seção 2.2.

2.1. Prototipagem de Aplicações Pervasivas

No contexto do *framework*, a prototipagem de aplicações pervasivas é fundamental para a depuração e teste em um AmbI. Num desenvolvimento paralelo, a IPGAP é proposta como um aparato ferramental que permite ao desenvolvedor de aplicações ter acesso a estes benefícios. Além disso, a IPGAP possui uma Interface Gráfica de Usuário (*Graphic User Interface* – GUI) que proporciona ao usuário final controlar remotamente recursos do ambiente.

Na Figura 1 é ilustrado um *tablet* executando o aplicativo da IPGAP e *smartphones* simulando recursos como TV, lampada e termômetro. A IPGAP faz o

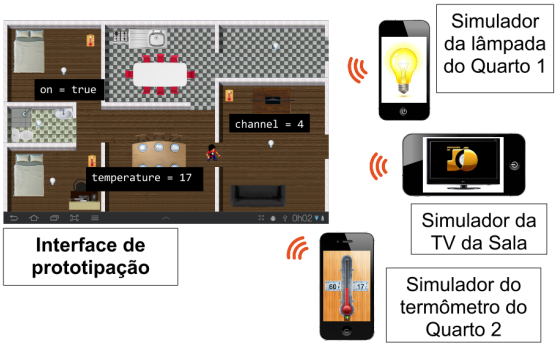

Figura 1. Interface de Prototipagem

mapeamento do ambiente e permite visualizar e instanciar cada um desses elementos e representá-los na sua GUI. Na figura há outros recursos além dos representados pelos *smartphones*, estes são recursos criados na própria IPGAP. Assim, o desenvolvedor de aplicações não fica limitado pela disponibilidade de recursos físicos podendo simular os elementos indisponíveis.

O *framework* proposto permite ao desenvolvedor criar aplicações nas dimensões e quantidade que desejar para um ambiente real. A IPGAP permite verificar a viabilidade do funcionamento das aplicações desenvolvidas em ambiente próximo ao real. As restrições deste AmbI real são semelhantes as que ocorrem no AmbI simulado na IPGAP. Isto permite avaliar o desempenho antes da implantação efetiva das aplicações.

2.2. Computação Sensível ao Contexto

O contexto exerce um papel de fundamental importância na computação ubíqua. Durante a década de 90, diversos autores procuraram definir contexto, muitas delas pouco apuradas. Baseados em definições de autores anteriores, Dey e Abowd [Dey et al. 2001] definiram que contexto é qualquer informação relevante usada para caracterizar a situação de entidades (i.e., pessoa, lugar ou objeto).

As três entidades reconhecidas por [Dey et al. 2001] como de maior importância possuem representatividade no *framework*, são elas: lugares, pessoas e coisas. No AmbI, "Lugares" são os cômodos, os andares de 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, pode ser que este a tenha esquecido. 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 trabalho oferece um suporte para a declaração e consulta destas informações (vide Seção 3.1.1), além da definição de Interpretadores de Contextos (vide Seção 3.2.2), que facilitam a construção de regras de contexto para atuar no AmbI.

2.3. Comunicação

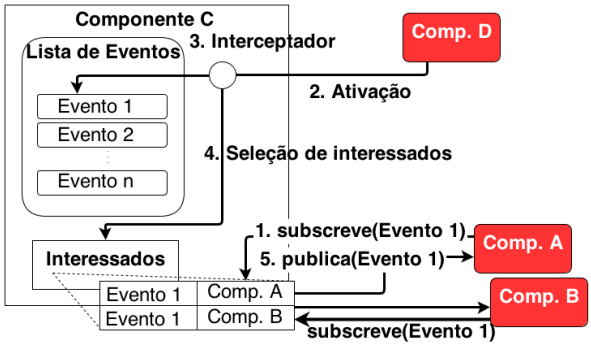

Figura 2. Esquema de *publish-subscribe*

Foram utilizados dois mecanismos no *framework*: a invocação remota de procedimentos (*Remote Procedure Call* - RPC) e a publicação de eventos por interesse através do 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 está contida a sua interface de chamadas de procedimentos públicos. A chamada e o retorno do procedimento são enviados através de mensagens serializadas. Este mecanismo é utilizado pela IPGAP para controlar remotamente os componentes do AmbI.

O paradigma de *publish-subscribe* é utilizado no Modelo de Contexto para aquisição de mudança de estado (vide Seção 3.2). A Figura 2 ilustra suas etapas no esquema de componentes. Inicialmente, os Componentes A e B se subscrevem ao Evento 1 do Componente C (Passo 1) e esta subscrição é armazenada no campo de Interessados. Posteriormente, o Componente D ativa o evento subscrito (Passo 2). Ocorre a interceptação (Passo 3) e os Componentes A e B interessados neste evento (Passo 4) são notificadas (Passo 5).

Como suporte, a comunicação no *framework* necessita de uma infraestrutura básica de rede Wi-Fi, o que garante o mínimo de segurança contra acesso de agentes externos a rede do ambiente e agrega benefícios de mecanismos de criptografia como a Proteção de Acesso a Wi-Fi (Wi-Fi Protected Access - WPA).

3. Descrição do *Framework*

O *framework* provê facilidades e padrões de programação a aplicações sensíveis ao contexto focadas em AmbI (incluindo *smart homes*). Dentre as facilidades está a capacidade de tornar transparente as 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 do ambiente distribuído proposto pelo *framework*, e é utilizado na modelagem das aplicações. Se uma aplicação necessita de uma entidade distribuída, pode representá-la através de um AR, 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* foi implementado o projeto para a plataforma *Android* [Saha 2008] denominado *SmartAndroid*. Este projeto se beneficia da

acessibilidade da tecnologia e de sua portabilidade a diversos tipos de dispositivos embarcados. Isto possibilita que o *framework* seja mais abrangente e possibilite o desenvolvimento sistemas de aplicações ubíquas mais elaborados ao longo do tempo. A arquitetura do *framework* é definida a partir do Modelo de Componentes Distribuídos (Seção 3.1) e do 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 3). O Modelo de Componentes Distribuídos segue esta estrutura para mapear as aplicações ubíquas no AmbI. O AR foi definido para padronizar os componentes de interação do ambiente, e dessa forma resolver questão da heterogeneidade de recursos. A sua estrutura geral, como ilustrada na Figura 3 (a), é composta por nome único ("tvFamília"), hierarquia de tipos ("\Visual\TV"), localização ("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 sequencia de classes, foi inspirada na ontologia utilizada 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.

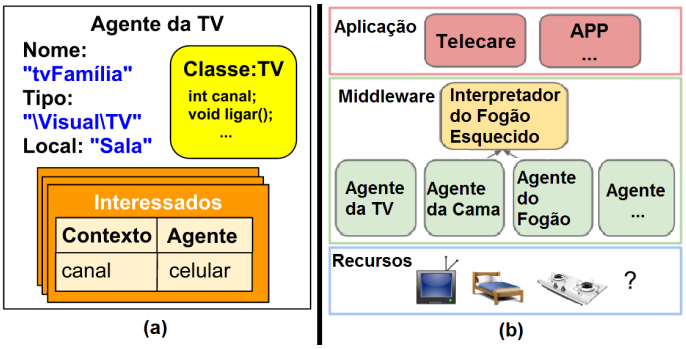

Figura 3. Arquitetura do *Framework*: (a) Estrutura do AR; (b) Camadas

A lista de interessados do AR é utilizado pelo Modelo de Contexto (Seção 3.2) para notificar outras instancias sobre sua mudança de estados. Um AR armazena o seu estado por meio de seus atributos declarados e os expõe através da criação de Variáveis de Contexto (VC), como será visto na Seção 3.2.1. Geralmente as mudanças nas VCs são provocadas por Operações (OP), como será estudado na mesma seção. Do ponto de vista da arquitetura, as VCs são as portas de saída de informação, e as OPs são as portas de entrada de comando.

Na Figura 3 (b) são apresentados componentes 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 (como *smartphones* e *tablets*) ou criados através da IPGAP. A figura apresenta como recursos exemplo um televisor, uma cama e um fogão. No *middleware* são apresentados os seus ARs representativos, além do AR de interpretação de contexto para verificar se o fogão foi esquecido ligado por um ocupante da casa. Este interpretador verifica se alguém está presente na cama e por quanto

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

tempo o fogão está ligado, se um tempo limite for ultrapassado uma ação é disparada para que o fogão seja desligado. Os ARs referentes a cama e ao fogão coletam os dados destes recursos físicos e os representam na camada do *middleware*, isto permite que estas informações sejam utilizadas pelos componentes da camada da aplicação. As aplicações utilizam o SGAR (Seção 3.1.1) para criar, consultar e instanciar ARs no *middleware*.

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 and Loques 2006]. Neste trabalho, o SGAR é composto por três componentes, ou serviços: 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 verifica suas existências (*search*) e o SLR verifica suas posições físicas para indicar 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 todo AR e são possibilitadas pelo mecanismo de comunicação por eventos descrito na Seção 2.3. Os componentes (A, B, C, D) na Figura 2 correspondem aos ARs e os Eventos correspondem a mudança de estado das VCs. O uso das Operações "publish" e "subscribe" é explorado pelos Interpretadores de Contexto apresentados na Seção 3.2.2.

O Repositório de Recursos, como apresentado na Figura 4, possui o Diretório de Recursos e o Mapa, o qual contém representado o conjunto de espaços físicos do AmbI. O Diretório contém os dados dos ARs armazenados conforme o tipo de cada um, 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 ARs através do RPC (Seção 2.3). 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 um conjunto de AR contidos no respectivo espaço.

3.2. Modelo de Contexto

O *framework* utiliza uma abordagem distribuída em seu Modelo de Contexto, onde a informação de contexto é mantida e disponibilizada pelos ARs. A informação pode estar

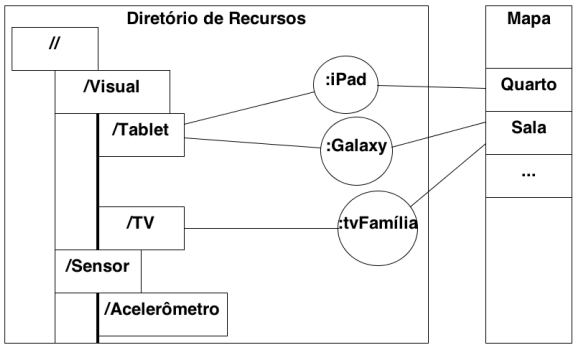

Figura 4. Repositório de Recursos

em qualquer lugar da 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, como apresentado na Seção 2.3.

Embora este trabalho desobrigue a utilização de um serviço de gerenciamento de contexto centralizado, ele também não o inviabiliza. A abordagem adotada é flexível de tal maneira que, caso seja necessário, pode ser criado 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). Se, por exemplo, há um agente para a televisão registrado no sistema e ele provê 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, enfim, tudo o que diz respeito ao estado da televisão e é efetivamente coletado.

Uma Operação (OP) tem o papel de expor uma funcionalidade (ou serviço) do AR, possibilitando às aplicações interagir ativamente no ambiente. Por exemplo, uma televisão integrada ao sistema pode oferecer OPs como 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 a VC como a OPs são interfaces, ou portas, do AR. Diferentes aplicações, instaladas no mesmo ambiente podem usar estas interfaces para interagir com o próprio ambiente. A Figura 5 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 WAP2 e WEP) evita que aplicações estrangeiras possam acessar os recursos de um ambiente. O domínio é uma

abstração que provê restrições de acesso aos ARs que contém, o que impõe uma barreira para aplicações maliciosas, que devem requisitar permissão de acesso.

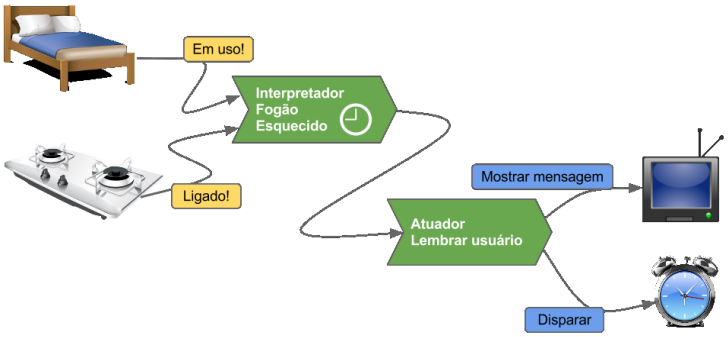

Figura 5. Interpretador de Regra

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 em específico. O suporte à interpretação de contexto pelo *framework* possibilita uma separação de interesses, onde os desenvolvedores abstraem a implementação de regras de contexto e se focam na lógica da aplicação.

A intepretação 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, imprimir na tela, 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: se uma pessoa (que mora sozinha) ligou o fogão, se deitou na cama, e passaram-se 15min, então dispare o alarme do dispertador e mostre na televisão a mensagem “O fogão foi esquecido ligado!”. Este exemplo simplista poderia ser implementando no sistema como representado na Figura 5. 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 avaliou a regra como verdadeira durante o tempo de 15min (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 é também um AR, e estende 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 possibilite à 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 - 20º). Através do mecanismo de subscrição os ICs notificarão os ARs atuadores, que executarão tarefas no AmbI através de chamadas RPC.

4. Avaliação

A qualidade do *framework* foi avaliada 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.

4.1. Prova de Conceito do *Framework*

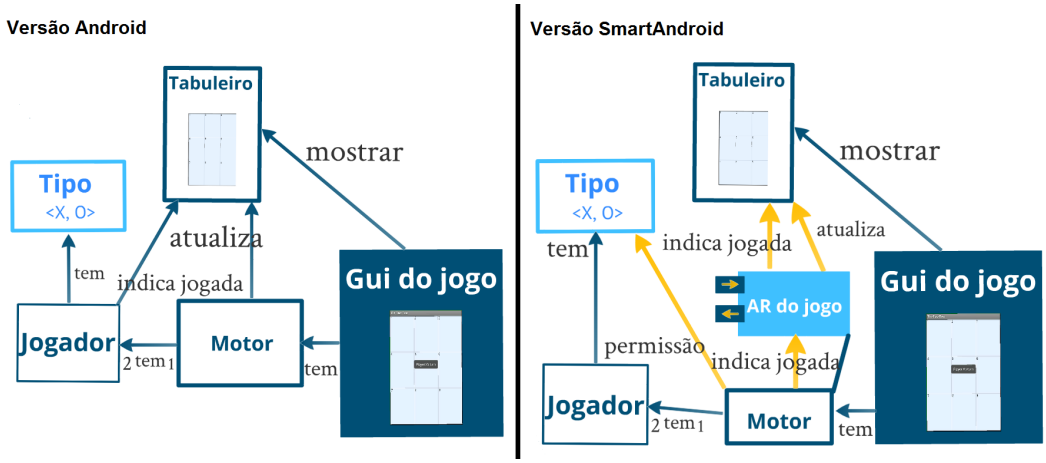

Figura 6. Comparação entre os modelos das aplicações Android e SmartAndroid

Para a primeira avaliação utilizou-se a aplicação do jogo da velha como base. Em sua versão *Android*, 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 ilustra o modelo de aplicação das duas versões. A versão original possui o componente da GUI (GUI do Jogo), 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 estado do Tabuleiro e o mostra na GUI. O Tipo do Jogador é verificado para indicar qual simbolo da jogada é marcado no tabuleiro ("X" ou "O").

Na versão SmartAndroid é inserido o AR do jogo que é associado ao Motor de um dos Jogadores (Jogador "X") utilizador de um dispositivo. O AR do Jogo intercepta a indicação de jogada deste Motor e encaminha para ARs de outros participantes (um deles o do Jogador "O"). Estes ARs atualizam o tabuleiro com a jogada vinda por RPC do Jogador "X" ("atualiza" do AR do Jogo). Para evitar que um jogador jogue na vez do outro, por exemplo, que o Jogador "X" jogue no turno do Jogador "O", é verificado se o Tipo do Jogador da vez é o mesmo do utilizador que, interagindo com a GUI, tenta a jogada ("permissão"). Na Figura 6 as ligações unidirecionais em amarelo indicam as alterações realizadas no processo de transformação. Usando as operações definidas na Tabela 1, o Código 1 apresenta a descoberta dos outros ARs na linha 1, o registro do AR local na linha 2, e a subscrição de interesses entre ARs do AmbI nas linhas 5 e 6. Após isto, nenhuma alteração no nível de comunicação ou contexto foi 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 gameList = SDR.search(TYPE, GameAgent); \\Consulta por tipo
2  SRR.register(gameAgent); \\Registro do Agente do Jogo (gameAgent)
3  if (gameList.size>0) { \\Caso haja outros jogadores...
4     for (iGameAgent : gameList) { \\Subscrição entre ARs
5        iGameAgent.subscribe("indica jogada", gameAgent);
6        gameAgent.subscribe("indica jogada", iGameAgent);
7     }
8  }...
```

4.2. Aplicação de Controle de Iluminação Residencial

Dentre os protótipos construídos como provas de conceito, foi criado 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 proximidade, identifica-se a presença/ausência de pessoas no cômodo, conforme a pessoa sai de um cômodo e passam-se T_{max} segundos sem que ela volte, então a luz daquele cômodo é desligada. Em adição, foi especificado também que, se é noite e uma pessoa entrou em um cômodo, então a lâmpada daquele cômodo deve ser ligada – a possibilidade de bloquear esta função evita ocasionais desconfortos.

O Código 2 ilustra a inicialização do SmartLiC, onde são explorados os serviços de registro, localização, descoberta e a instanciação de regras de contexto. Uma vez obtidos os cômodos (ou lugares), a aplicação obtém cada lâmpada e sensor de proximidade deste cômodo. Uma vez com as identificações destes ARs e com o valor de T_{max} definido, cria-se uma expressão para a regra (*ruleExpression*) e define-se um conjunto de ações a serem desempenhadas (*actions*).

Código 2. Inicialização do SmartLiC

```
1  List places = SLR.getPlaces(); \\Obter todos os lugares
2  for (place : places) { \\Para cada lugar...
3     \\Obter lâmpada
4     lampAR = SDR.search(TYPE, Lamp, place);
5     \\Obter sensor de proximidade
6     proxAR = SDR.search(TYPE, Proximity, place);
7     \\Escrever expressão da regra de contexto
```

```
8           ruleExpression = lampAR.on and proxAR.occupied and T > Tmax;
9           \\Escrever conjunto de ações
10          actions = lampAR.turnOff;
11          \\Sistema se encarrega de instanciar IC e atuador
12          newRule(ruleExpression, actions);
13      } ...
```

A execução do SmartLiC exige um tempo de resposta apurado. Com muitas instancias de lâmpadas este tempo pode aumentar, devido ao esquema de publicação por canais *unicast* relativos a cada uma de suas respectivas ARs. Um problema que pode ser resolvido através de uma disseminação de eventos mais eficiente. Uma proposta estudada é a das Redes Endereçadas Por Interesses (REPI) [Dutra and Amorim 2010], na qual sua utilização permite que o interesse do AR em uma VC seja registrado localmente, as mensagens são identificadas com o interesse e apenas nós da rede que tenham o mesmo as manipulam.

5. 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 evitam levantar pontos sobre a sobrecarga de comunicação decorrente 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 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 ao *publish-subscribe*, mas limitado a 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 concebidos de forma dinâmica e adaptativa. A proposta permite ao desenvolvedor de aplicações um aparato ferramental para montar um AmbI personalizado. 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). Ainda possui um

suporte para construção de regras sobre contextos do ambiente, permitindo prover em alto nível serviços aos usuários do sistema inteligente.

6. Conclusão e Trabalhos Futuros

Este trabalho esta embasado nas dissertações [Mareli and Loques 2012, Erthal and Loques 2012, Barreto and Loques 2012] em produção. Neste artigo, foi apresentado o *Framework* de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes, com funcionalidades conceituais que visam facilitar o desenvolvimento e a implantação de aplicações que interajam de forma ubíqua com o ambiente e seus ocupantes. Através dele, o desenvolvedor cria serviços que atendam a requisitos preestabelecidos de acordo com as características de um AmbI. Após a concepção de serviços para a 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 as 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 a aprendizagem de máquina. E para permitir um controle de serviços maior a nível de usuário, pretende-se proporcioná-lo uma interface compreensível para 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. and Loques, O. (2012). Interface de prototipagem e gerenciamento de aplicações pervasivas. Dissertação de mestrado de computação em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro.

Brown, N. and Kindel, C. (1998). Distributed component object model protocol–dcom/1.0. *Online, November*.

Cardoso, L. and Loques, O. (2006). Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software. Dissertação de mestrado em computação, 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 Brasi-*

leiro 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.

Dutra, R. and Amorim, C. (2010). Modelo de comunicaçao endereçada por interesses. Technical report, Technical report, Relatório Técnico ES 733/PESC-COPPE-UFRJ.

Erthal, M. and Loques, O. (2012). Interpretação de contexto em ambientes ubíquos inteligentes. Dissertação de mestrado de computação em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro.

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.

Mareli, D. and Loques, O. (2012). Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes. Dissertação de mestrado de computação em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro.

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.