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

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

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

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

Abstract.

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 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. Esta plataforma inclui também 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], prevê 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 indireta. Uma aplicação ubíqua identifica as necessidades do usuário obtendo informação de contexto através de sensores, e provê 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 mais em evidência. 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 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 tem por objetivo definir, no estado da arte, um arcabouço de técnicas para construção e 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, a qual compreende os sensores e atuadores, e a camada de aplicação, na qual 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*. Uma variedade de dispositivos e propriedades são encontradas no AmbI. Preocupado em organizar estes componentes, o trabalho em [Ranganathan et al. 2005] propõe uma estrutura de representação da camada física em hierarquia de tipos através de ontologia. Esta estruturação permite ampliar o escopo de operações de suporte de um sistema ubíquo.

Neste trabalho é proposto um *framework* de desenvolvimento de aplicações ubíquas em AmbI. O objetivo é dar suporte a 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 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 consiste de um conjunto de interpretadores 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.

A qualidade de suporte do *framework* foi avaliada 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. Para testar a eficiência do modelo de regras, foi construída uma aplicação que explora os mecanismos de comunicação utilizados no modelo de componentes distribuídos e utiliza as informações básicas de um agente de recurso, como localização e identificação de tipo.

Este artigo é organizado da seguinte forma: a Seção 2 apresenta os conceitos básicos utilizados ao longo do texto. 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 eficiência do *framework* em construir aplicações ubíquas e demonstrada uma aplicação que explora, através da implantação em uma *smarthome*, 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 como base os conceitos de Inteligencia Ambiental, de Computação Sensível ao Contexto e de Prototipagem de

Aplicações Ubíquas. A Inteligencia Ambiental define o espaço onde estas aplicações funcionam. O comportamento de sistemas ubíquos são definidos a partir de técnicas aplicadas na Computação Sensível ao Contexto. E os esquemas de prototipagem são utilizados para manipular e testar o funcionamento do conjunto de aplicações no ambiente.

A Computação Ubíqua se inclui dentre as inúmeras áreas abrangidas pela Inteligência Ambiental. Em [Augusto and McCullagh 2007] é apresentada uma arquitetura em camadas como base para sistemas no AmbI. 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, as quais podem ocorrer por um ocupante ou mecanismos de inteligencia artificial, e as interações com o ambiente. O *middleware* é a parte que melhor define a função geral de nosso *framework*, as aplicações ubíquas são definidas e gerenciadas nesta camada para promover a ligação entre o ambiente e os serviços computacionais.

Um AmbI é o meio onde se agrega inteligencia computacional. Há diversos exemplos na literatura que utilizam especificações deste tipo de ambiente, dentre eles o termo ”smarthome” (casa inteligente) se mostra como um dos mais populares. Há diversas propostas no estado da arte explorando este tipo de espaço [Helal et al. 2005, Augusto and McCullagh 2007, Ranganathan et al. 2005]. Estes trabalhos costumam explorar a heterogeneidade de dispositivos e as rotinas bem definidas de seus ocupantes tendo a casa inteligente como cenário. Apesar do *framework* proposto neste trabalho abranger qualquer tipo de AmbI, boa parte dos exemplos de aplicações abordados tem como cenário o tipo residencial.

Para que as aplicações ubíquas possam ser integradas a elementos de baixo nível do AmbI é necessário que este espaço seja representado no nível de *middleware*. Segundo [Kramer 2007], a abstração quando bem definida pode melhorar a visualização de problemas do mundo real e consequentemente ajudar na definição de uma resolução clara e objetiva por meio computacional. Este trabalho segue neste sentindo definindo um modelo de componentes arquiteturais distribuídos [Brown and Kindel 1998] para representar de forma geral sistemas construídos pelo nosso *framework*.

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. No trabalho sobre IPGAP é proposto 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.

Estas capacidades destacadas são proporcionadas pelas seguintes funcionalidades: mapeamento do ambiente, instalação de novas aplicações ubíquas e ligação com aplicações já existentes. Através destas funcionalidades é possível ter uma ambiente preenchido de recursos simulados (como televisor virtual, geladeira virtual, etc.) e fazê-los executar junto a recursos reais que podem ser representados em sua GUI. A GUI da IPGAP foi elaborada apropriadamente para o universo dos *tablets* e *smartphones*, nela podem ser visualizados os recursos reais e virtuais localizados no mapa como ilustrado na Figura 1.

Figura 1. Interface de Prototipagem

Na Figura 1 é ilustrado um *tablet* executando o aplicativo da IPGAP e *smartphones* simulando recursos como TV, lampada e um termômetro. Através da IPGAP é possível visualizar cada um desses elementos e representá-los no mapa. Repare que na figura há outros recursos além dos representados pelos *smartphones*, estes são recursos criados na própria IPGAP. Com isso o desenvolvedor de aplicações não fica limitado pela disponibilidade de recursos físicos, ou seja, não é preciso ter disponíveis a mesma quantidade de dispositivos que uma aplicação real exigiria para que ocorra sua execução em um AmbI simulado.

O *famework* 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 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] compuseram a seguinte definição para contexto:

> "Qualquer informação que pode ser usada para caracterizar a situação de entidades (i.e., pessoa, lugar ou objeto) que são consideradas relevantes para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e aplicação. Contexto é tipicamente a localização, identidade, e estado de pessoas, grupos, e objetos computacionais e físicos"

Segundo [Dey et al. 2001] três entidades de contexto foram identificadas como de maior importância, são elas: lugares, pessoas e coisas. No ambiente de uma *smarthome*, "Lugares" são os cômodos, os andares de uma edificação, ou espaços em geral; "Pessoas" são indivíduos que povoam o ambiente e interagem com o mesmo; e "Coisas" são representações virtuais de objetos físicos ou componentes de *software*.

Após a identificação das entidades, são introduzidas quatro categorias de contexto essenciais: identidade, localização, estado e tempo. "Identidade" corresponde à identificação única de uma entidade. "Localização" não só diz respeito à posição de uma pessoa e sua inserção em uma área ou subárea, mas diz também respeito à proximidade de uma pessoa em relação à outra, ou à uma coisa. "Estado" identifica características intrínsecas da entidade (e.g., temperatura de uma sala, sinal vital de uma pessoa). "Tempo" é uma informação de contexto quando ajuda a caracterizar a situação, geralmente usado com outras informações de contexto, indicando um instante ou período durante o qual alguma informação contextual é conhecida ou relevante.

Uma meta da aquisição de contexto é determinar o que o usuário está tentando realizar, por exemplo, se uma pessoa sai de casa e deixa a torneira aberta, pode ser que este a tenha esquecido. A aquisição do contexto de forma automatizada contribui para a construção de aplicações para ambientes inteligentes 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 uma maneira de se representar os recursos espalhados pelo ambiente e os recursos no nível simulado de software, tornando homogênea a comunicação entre os componentes, de tal maneira que é simples se declarar e consultar as informações de contexto. Através de serviços padrões é possível se encontrar as entidades e obter seu contexto, onde esta pesquisa pode ser feita tanto utilizando a identificação da entidade, como o seu tipo ou localização física. A sensibilidade ao contexto é caracterizada pela reação do sistema aos eventos de mudança do contexto, portanto, o *framework* introduz o Interpretador de Contexto

3. Proposta do *Framework*

O *framework* provê facilidades e padrões de programação a aplicações sensíveis ao contexto focadas em AmbI (incluindo *smarthomes*). O *framework* torna mais transparente as partes da aplicação que envolvem a comunicação, a aquisição de contexto e a localização de recursos no ambiente.

Ao invés de se lidar diretamente com sensores e atuadores espalhados no ambiente, o *framework* utiliza, como abstração, componentes chamados Agentes de Recursos (AR). Os ARs são elementos de primeira ordem da plataforma, representando recursos de hardware ou software disponíveis no ambiente. Eles têm como responsabilidade coletar as diversas medidas dos sensores ou componentes de software presentes no contexto de execução do das aplicações [Bezerra 2011]. Os ARs resolvem a complexidade de tratamento com os componentes de baixo nível e disponibilizam interfaces padrões de interação tanto local como em rede. Os ARs podem ser criados no AmbI para representar, por exemplo: uma televisão que expõe suas informações de contexto e provê serviços para as aplicações; sensores de temperatura da casa; sensores de vazamento de gás na cozinha; um medidor de pressão arterial; um atuador para fechar as janelas; etc.

O AR é a unidade básica de modularização do ambiente distribuído proposto pelo *framework*, assim sendo, o mesmo pode também ser utilizado na modelagem das aplicações. Se uma aplicação necessita de uma entidade distribuída, pode representá-la através de um AR. Por exemplo, um desenvolvedor poderia construir um jogo, representando através de ARs os jogadores, o tabuleiro, os monstros, recursos do cenário, etc.

Para possibilitar a comunicação entre os diferentes ARs, o SmartAndroid dispõe

de um serviço que gerencia estes recursos, e mantém informações de identificação e localização dos mesmos, dentre outras. Este serviço chama-se Serviço de Gerenciamento de Agentes de Recursos (SGAR) e encontra-se centralizado no ambiente distribuído. Como apresentado na Figura 4, 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 diferença entre o SDR e o SLR está no fato de que o primeiro se refere à localização na rede de computadores, e o segundo se refere à localização física do dispositivo no ambiente.

O SGAR tem por objetivo aumentar a visibilidade das informações de contexto dos ARs e tornar a comunicação transparente para o programador das aplicações. A instalação e desinstalação de aplicações ubíquas são viabilizadas pelo SRR. O reconhecimento dos ARs por outras aplicações é possibilitado pelo SDR. E as localizações de ARs de elementos físicos do AmbI podem ser consultadas através da SLR.

3.1. Modelo de Componentes Distribuídos

O modelo de componentes distribuídos mapeia as aplicações ubíquas em um ambiente inteligente genérico. Em geral este mapeamento é dividido em três camadas essenciais:

1. A camada física ou conceitual dos recursos
2. A camada intermediária (*middleware*) com os ARs dos respectivos recursos e Interpretadores de Contexto
3. A camada das aplicações que agregam os componentes do *middleware* em aplicativos.

O modelo de componentes distribuído pode variar dentre estas camadas de acordo com as aplicações ubíquas implantadas no ambiente.

Na Figura 2 exemplifica-se uma *smarthome* sendo gerenciada pelo *framework*. 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 virtualmente 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 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.

O AR foi definido para padronizar os componentes de interação do ambiente, e dessa forma resolver questão da heterogeneidade apresentada em [de Araujo 2003]. A sua estrutura geral é composta por nome único, tipo, localização, código(que define a classe) com variáveis e métodos e uma lista de interessados em informações de contexto. O tipo é definido por uma hierarquia que vai se especializando desde o mais genérico (raiz) até o tipo da classe (que representa o código do AR corrente). Todo AR possui o tipo raiz como super pai. A lista de interessados possui registros com dois campos, um com a variável do AR corrente e outro com a referência do outro AR interessado nesta informação.

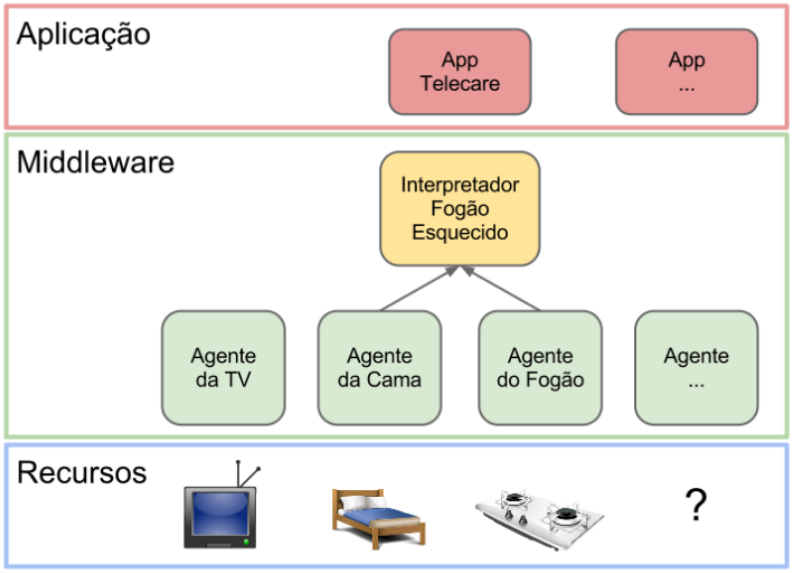

Figura 2. Arquitetura do *Framework*

A definição de tipo do AR é uma etapa importante para a definição de aplicações ubíquas. Em [Ranganathan et al. 2005] é utilizado uma ontologia para classificar entidades (como os ARs) para possibilitar a consulta a partir de propriedades predefinidas. Uma aplicação de apresentação de slides é utilizada como exemplo para ilustrar a importância desta ontologia. A aplicação consiste em mostrar os slides no mesmo cômodo residencial onde uma determinada pessoa se encontra. Então, ao invés de definir dispositivos específicos para realizar esta tarefa, a aplicação consulta por dispositivos com as seguintes propriedades: ser do tipo visual e estar no cômodo onde a pessoa alvo está presente. Assim, quando esta pessoa muda de local, o dispositivo instanciado para mostrar a apresentação passa a ser o correspondente ao mesmo local dela. Em nossa proposta, o SDR possui a busca por tipo que retorna instancias com propriedades predefinidas, tal como em [Ranganathan et al. 2005]. A forma de utilização de instancias podem ser dinamicamente definidas por Interpretadores de Contexto (vide Seção 3.3.2).

Para exemplificar um AR, um exemplo de instancia representando um televisor em ambiente residencial é ilustrado na Figura 3. O nome identifica unicamente a TV no ambiente residencial (”tvDaSala”), a localização é definida como a sala de estar. O tipo ”TV”, nome da classe corrente, é definido como especialização do tipo ”Visual”. O código possui as variáveis e as operações específicas da classe. E a lista de interessados possui cada registro representando um interesse, o qual contém o campo da variável e outro campo com o AR interessado na mudança de valor da respectiva variável. Por exemplo, o registro da Figura 3 indica que o AR do celular está interessado na mudança de canal da TV.

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 3.3.1. Da mesma forma ele pode declarar suas possíveis 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.

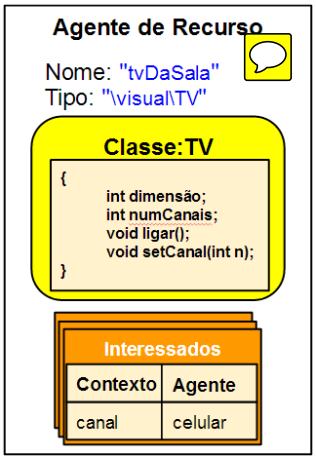

Figura 3. Instância do Agente de Recurso

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 (através do SRR) e descoberta (através do SDR) de ARs segue a ideia geral sobre serviços de registro e descoberta de recursos apresentada em [Cardoso 2006]. Neste trabalho foi acrescentado o SLR, o qual tem a função de localização física. A Figura 4 ilustra as principais operações de cada serviço de suporte. O SRR é utilizado para registrar ou remover o registro, o SDR é utilizado na busca e o SLR na localização de ARs. Estas operações manipulam dados representativos dos respectivos ARs no Repositório de Recursos.

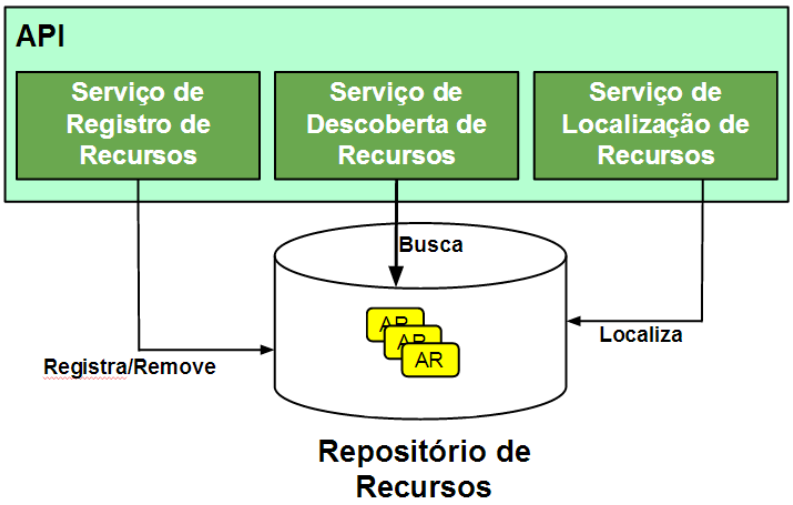

Figura 4. Serviços de Suporte

O Repositório de Recursos possui o Diretório de Recursos e o Mapa do ambiente. O Diretório contém os dados dos ARs armazenados conforme o tipo de cada um. Estes dados consistem do nome, tipo, localização e a referência de acesso. A referência de acesso permite a interação através do mecanismo de comunicação por invocação remota (RPC) (vide Seção 3.2). O Mapa possui o conjunto de espaços físicos definidos no ambiente. Cada espaço possui uma área e um nome. No caso de uma *smarthome* os cômodos

da casa representam estes espaços. Cada entrada do Mapa referencia um conjunto de AR contidos no respectivo espaço como pode ser visto na Figura 5.

Na Figura 5 é apresentada a organização de um conjunto de recursos no Repositório. Neste caso há três recursos registrados, uma TV (”tvDaSala”), e dois *tablets* (”iPadQuarto”e ”tabSala”). Os três fazem parte do conjunto de ARs do tipo ”Audiovisual”, estes tipos funcionam como subdiretórios de recursos. E por fim, o Mapa indexa cada um ao respectivo cômodo indicado pelo atributo de localização.

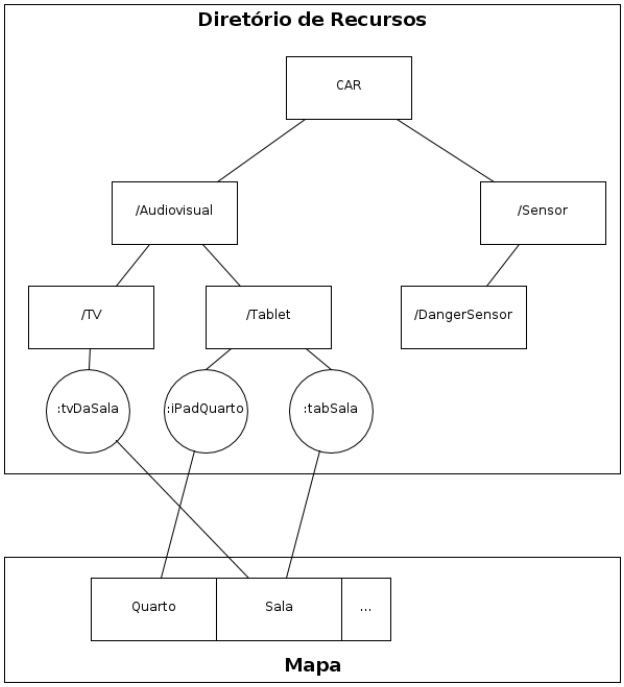

Figura 5. Repositório de Recursos

O SRR é responsável por adicionar, remover e inclusive atualizar informações sobre os ARs no Repositório de Recursos. Através deste serviço, as aplicações podem adicionar novos tipos no Diretório de Recursos e espaços no Mapa. E em cada AR pode ser atualizada a informação sobre localização neste Mapa.

O SDR é responsável por prover acesso as instancias de ARs registradas. O SDR possui três tipos de descobertas: por referência, por nome e por tipo. As consultas por referências e nome podem ou não retornar uma instância única, estes dois tipos de buscas podem ser utilizados quando uma aplicação deseja saber se um AR específico está registrado. As consultas por tipo retornam todos os ARs contidos no subdiretório correspondente. Por exemplo, na Figura 5 o tipo “Audiovisual” contém três ARs, os quais são “tvDaSala”, “iPadQuarto” e “tabSala”. Logo, uma consulta neste caso retornaria uma lista contendo as referências para cada um dos três ARs.

O SLR, além de prover acesso as instâncias de ARs físicas, permite obter os ARs mais próximos a um ponto específico do AmbI. A consulta por localização verifica os ARs indexados a um local especificado do Mapa. No exemplo da Figura 5, uma consulta por ARs da sala retornaria a “tvDaSala” e o “tabSala”. A consulta por proximidade leva em conta a subdivisão de espaços do ambiente. Então quando for consultado qual

o AR mais próximo a “tvDaSala”, mesmo que o “iPadQuarto” esteja mais próximo fisicamente, “tabSala” será o retornado como mais próximo. Isto ocorre porque os ARs do mesmo cômodo são verificados primeiro, depois são verificados ARs de cômodos vizinhos e assim por diante. Em exemplos de aplicações de notificação de mensagens esta lógica é favorecida, já que uma aplicação de lembrete de remédios vai querer notificar o paciente através do dispositivo visual mais proximamente visível, ou seja, de preferência no mesmo cômodo sem possíveis paredes atrapalhando.

3.2. Comunicação

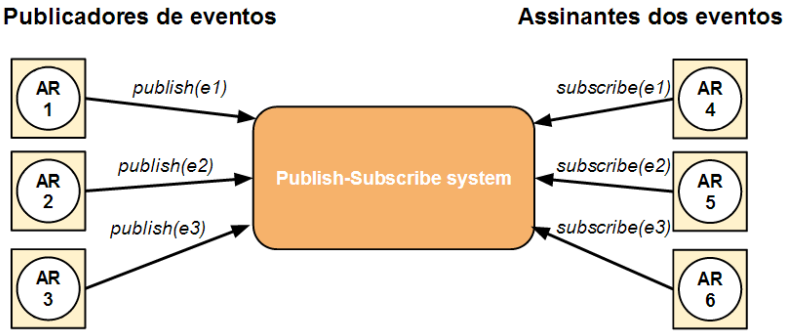

Figura 6. Esquema de *publish-subscribe*

Em relação a comunicação, foi definido um esquema de publicação e interesse por eventos para atender a um requisito essencial do *framework*, a interpretação de contexto. Este esquema é denominado de *publish-subscribe*, o qual muito utilizado em sistemas distribuídos. Este paradigma corresponde à uma comunicação assíncrona, que envolve por parte da entidade interessada no registro de interesse na variável do AR alvo. No esquema ilustrado na Figura 6, os ARs 4, 5 e 6 se subscrevem a eventos que podem ser gerados pelos ARs 1, 2 e 3 respectivamente. Como auxilio a este esquema é utilizado conceitos de Programação Orientada a Aspectos (POA) [Kiczales et al. 1997], no qual é utilizado para definir pontos de aviso (*advices*) para VCs alvos de interesse. Estes pontos de aviso indicam o disparo de uma notificação *unicast* a todos os ARs interessados.

Existe ainda possibilidade de comunicação direta com os ARs, o SDR pode ser utilizado para encontrar as referências de acesso direto a um em específico. Neste caso além de possuir a referência é necessário instanciar um *proxy* correspondente a mesma classe do AR alvo. Com o *proxy* instanciado, o mecanismo de RPC [Birrell and Nelson 1984] pode ser utilizado para fazer as invocações remotas das suas operações. A IPGAP utiliza a comunicação direta como meio para controlar remotamente os ARs visualizados no mapa.

O *framework* possui uma flexibilidade de comunicação que amplia o escopo de desenvolvimento de aplicações ubíquas, enquanto que muitas abordagens se restringem a comunicação baseada em eventos por ser mais econômica. A parte de comunicação do *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 WPA (sigla em inglês para Proteção de Acesso a Wi-Fi).

3.3. Modelo de Contexto

Uma infraestrutura para construção de aplicações sensíveis ao contexto (ou aplicações ubíquas) pode ter uma abordagem centralizada ou distribuída. Em abordagens centralizadas faz-se necessário a utilização de um serviço de gerenciamento de contexto, responsável por manter a informação contextual e oferecer interfaces para subscrições e consultas. Abordagens como esta correspondem à uma arquitetura do tipo *blackboard* (ou quadro-negro), onde uma entidade envia uma mensagem para uma memória compartilhada entre todas as entidades, e também pode se subscrever para receber mensagens que respeitam algum padrão especificado [Winograd 2001]. Todas as comunicações ocorrem através de um servidor centralizado e as mensagens são redirecionadas para os interessados.

Este trabalho adotou uma abordagem distribuída, onde a informação de contexto não é armazenada em um servidor centralizado, mas é mantida pelos ARs. O processo de aquisição do contexto envolve a descoberta do AR de interesse através do SDR, com a posterior subscrição do mesmo, como apresentado na Seção 3.2. Assim sendo, a informação fica distribuída pelo ambiente e é informada aos interessados diretamente, enquanto na abordagem centralizada são utilizados dois pulos. Embora a informação esteja distribuída em diferentes dispositivos, possivelmente com diferentes sistemas, o *framework* provê um nível de abstração tal que o acesso a qualquer informação é feito de maneira padronizada.

3.3.1. 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, possivelmente 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, enfim, tudo que diz respeito ao estado da televisão e é efetivamente coletado.

O papel da Operações (OP) é expor as funcionalidades 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.

A Figura 7 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.

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. O SmartAndroid possui implementado uma estrutura de tipos que facilita a descoberta de ARs de interesse e possibilita o conhecimento de suas interfaces. Na implementação foi criado uma hierarquia de tipos com foco em *smarthomes*, porém novos tipos podem ser criados.

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 ambiente. 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.

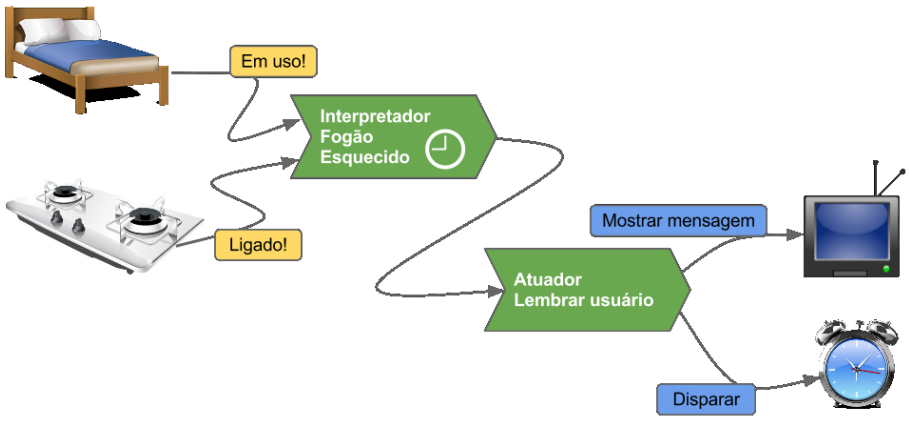

Figura 7. Interpretador de Regra

3.3.2. 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 mantém suas preocupações com a 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 7. 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 à cada atualização de valores; e um módulo temporizador, 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 [1] com valores ou outras VCs, 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).

4. Avaliação

Para consolidar a proposta do *framework* foi desenvolvido um 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 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 regras.

4.1. Prova de Eficiência Conceitual

Para a primeira avaliação utilizou-se a aplicação do jogo da velha como base. Nesta aplicação em sua versão *Android*, os jogadores interagem através de uma mesma tela de dispositivo (*tablet* ou *smartphone*). Na sua versão desenvolvida com o SmartAndroid os jogadores interagem sobre um mesmo tabuleiro em dispositivos diferentes. A Figura 8 ilustra o modelo de aplicação das duas versões. A versão original possui o componente da GUI (GUI do Jogo da Velha), Motor do Jogo, a estrutura do Tabuleiro e a identidade dos Jogadores (Jogador) que pode ser do tipo “X” ou “O” (Tipo ¡X,O¿). O Motor do Jogo atualiza o Tabuleiro, motor o qual atualiza periodicamente a GUI mostrando o novo estado da estrutura. O tipo do Jogador é verificado para indicar qual simbolo da jogada é marcado no tabuleiro (“X” ou “O”).

Na versão *SmartAndroid* é inserido um AR do jogo para interação remota (Agente do Jogo), o qual esta associado ao Motor de Jogo do dispositivo de um dos Jogadores (Jogador “X”). O Agente do Jogo intercepta a indicação de jogada deste motor e encaminha para o Agente de Jogo do outro jogador (Jogador “O”). Este AR do Jogador “O” atualiza o tabuleiro com a jogada do Jogador “X” (“atualiza” do “Game Agent”). Para evitar

[1] Possíveis comparações: $=, \neq, <, >, \leq$ e $\geq$

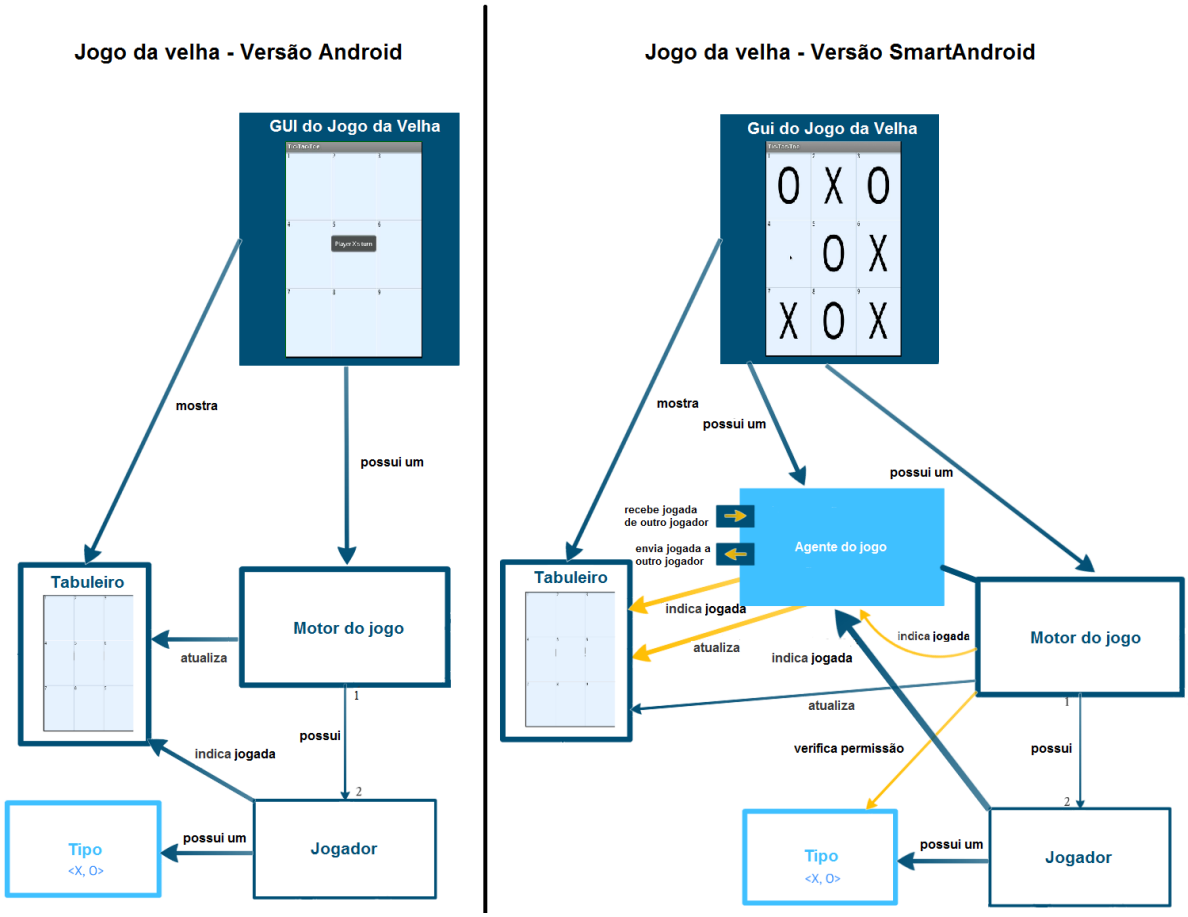

Figura 8. Comparação entre os modelos das aplicações Android e SmartAndroid

que um jogador jogue na vez do outro, por exemplo, que o Jogador "X" jogue no turno do Jogador "O", verifica-se o Tipo do Jogador da vez é o mesmo do que tenta a jogada ("verifica permissão"), tentativa que ocorre através de interação do usuário com a GUI. Na Figura 8 as ligações unidirecionais em amarelo indicam as alterações realizadas no processo de transformação. A parte de inicialização do jogo foi modificada para registrar o AR do jogo e este registrar interesse em outros ARs que já registraram participação no mesmo jogo. Este registro ocorre para o recebimento da notificação de novos movimentos. Através do SDR é verificada a existência deste tipo de AR. Caso não haja nenhum então o AR passa a representar o Jogador "X", caso haja um apenas então o AR representa o Jogador "O", caso haja mais de um, o AR é apenas observador do jogo.

Foi observado que o uso de elementos do suporte como SDR e SRR e do esquema de *publish-subscribe* facilitou o processo de transformação da aplicação. Na etapa de inicialização do jogo, o SRR foi utilizado para registrar o AR do jogo e o SDR foi utilizado para descobri-los e o esquema de *publish-subscribe* foi utilizada para cada jogador registrar interesse nos movimentos do respectivo adversário. Após isso, a aplicação segue seu fluxo de execução padrão com o os procedimentos adicionais destacados em amarelo na Figura 8. Portanto, após a etapa de inicialização nenhuma alteração no nível de comunicação ou contexto foi necessária, todas as alterações posteriores ocorreram para promover a interação com o AR do jogo local ao dispositivo. Logo, podemos comprovar

que o uso de AR tornou transparente a programação da aplicação ubíqua.

4.2. Aplicação de Controle de Iluminação Residencial

Dentre os protótipos construídos como provas de conceito, a Aplicação de Controle de Iluminação Residencial (*Smart Light Controller* – SmartLiC) possibilita a experimentação dos serviços de registro, descoberta e localização dos recursos, assim como o uso da interpretação de contexto, da simulação de ARs, da IPGAP, dentre outros.

O intuito do SmartLiC é promover economia no consumo de energia elétrica reduzindo gastos com iluminação. O sistema identifica através de sensores de proximidade a presença/ausência de pessoas no cômodo, conforme a pessoa sai de um cômodo e passam-se 30 segundos sem que ela volte, então a luz daquele cômodo é desligada. Em adição, foi implementado também que se é noite e uma pessoa entrou em um cômodo, então a lâmpada daquele cômodo deve ser ligada, com a opção também de bloquear esta função.

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

Diversos trabalhos vem sendo desenvolvidos nas áreas de Computação Ubíqua e Computação Sensível ao Contexto aplicados a ambientes inteligentes. Alguns abordam a arquitetura do *framework* [Helal et al. 2005] [Ranganathan et al. 2005] e outros enfatizam aspectos de comunicação [Villanueva et al. 2009] [Santi et al. 2011].

Em [Helal et al. 2005] é proposta uma arquitetura de camadas. Dentre estas há a camada física com seus sensores, atuadores e outros dispositivos. Logo acima encontra-se a camada de plataforma de sensores onde ocorre a homogenização dos recursos físicos. A camada de serviços possui a representação dos recursos (serviços). Em analogia com nossa proposta, os serviços podem ser vistos como ARs, existem os mais básicos que representam diretamente os recursos e os mais complexos que é composto de vários serviços. As camadas de conhecimento e de contexto manipulam esta camada de serviços, a camada de conhecimento além de possuir elementos com funções similares ao do SGAR, na qual aplica um motor de raciocínio que permite avaliar se alguns serviços complexos estão disponíveis no ambiente. E a camada de contexto tem função similar ao nosso modelo de regras só que atuando em uma granularidade mais grossa. O gerenciamento destes serviços ocorre através dos elementos da camada de aplicações (aplicações ubíquas). A conclusão é que o nosso *framework* possui uma estrutura semelhante, porém há neste trabalho uma série de mecanismos não considerados em [Helal et al. 2005] para a construção e instalação de novas aplicações. Além disso o *framework* não se restringe a se aplicar apenas a um ambiente do tipo *smarthome*.

Em [Ranganathan et al. 2005] é proposto um conjunto de operações de alto nível para ambientes inteligentes (espaços ativos). As operações básicas são semelhantes as

funções de um sistema operacional, só que ao invés de manipular recursos deste tipo sistema, entidades do ambiente são manipuladas. Em nosso *framework* há apenas funções de manipulação de base de dados (inserir, deletar, atualizar, consultar). As 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 relutam em 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). O serviço de descoberta ocorre através de um protocolo de comunicação do tipo *multicast*. O módulo de descoberta possui portas especializadas para escuta, busca e manipulação em cada componente do sistema. Em analogia com a nossa proposta de registro de recursos, pode ser estipulado registro local de recursos através da porta de escuta, tendo assim cada componente um repositório local de recursos. Porém 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 a comunicação excessiva característica do padrão *multicast*.

O JaCa-Android [Santi et al. 2011] utiliza o esquema de captura de mudança de estado em recursos através de escuta de eventos, o qual é semelhante ao *publish-subscribe*, mas limitado a um mesmo dispositivo. O JaCa-Android é uma plataforma baseada em agentes para lidar com concorrência, eventos assíncronos, gerenciamento de recursos e sensibilidade ao contexto para possibilitar a construção de aplicações de manipulação de eventos. Para seu desenvolvimento são utilizadas as ferramentas de programação orientada a agentes. 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 típico de uma programação orientada a aspectos (POA), tipo de programação que reduz a carga de codificação de requisitos essênciais 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 artigo apresentou o *Framework* de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes, um arcabouço de funcionalidades conceituais que visam facilitar o desenvolvimento e a implantação de aplicações que interajam de forma ubíqua com o ambiente e seus ocupantes. O desenvolvedor, a partir do uso do modelo de componentes e o de contexto, 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. Incluem-se dentre estas manipulações, a instanciação das interpretadores de contexto e a manipulação

direta dos ARs através dos serviços do SGAR.

Com propósito de demonstrar e avaliar o *framework*, foram apresentadas duas implementações de aplicações: o jogo-da-velha com múltiplos participantes, e a aplicação de controle de iluminação residencial. A primeira visa provar a eficiência de uso do modelo de componentes distribuídos através da transformação de uma aplicação local em uma aplicação ubíqua, a qual permite participantes de uma mesma partida em diferentes dispositivos. A segunda aplicação tenta explorar boa parte dos benefícios da proposta, como a criação de interpretadores de contexto e a utilização do serviço de localização, para controlar a iluminação da residência provendo conforto e economia de energia. Outras aplicações estão em desenvolvimento para ampliar o escopo de avaliação tais como um sistema de saúde residencial [Carvalho et al. 2010] e uma aplicação de visualização de vídeo (e.g., filmes, programas televisivos) que segue para a televisão do cômodo para onde o usuário se moveu.

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 das interações promovidas pelas aplicações ubíquas estão limitadas as configurações da rede Wi-Fi, pretende-se evoluir para que este requisito seja refinado na camada do *framework*, na qual restrições de acesso a determinados ocupantes e entre diferentes domínios de aplicações são pontos importantes da abordagem futura. Outro ponto está na economia de energia, boa parte dos dispositivo possuem fontes de energias restritas, então atender este requisito visa tanto aumentar seu tempo de vida como reduzir o gasto total do ambiente. Os gastos devido a comunicação são os mais preocupantes, logo pretende-se reduzir o tamanho das mensagens de eventos e invocações remotas, através de mecanismos de compressão de dados, e diminuir o número de notificações de eventos devido a mudanças de estados pouco significativas. 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. Além disso, proporcionar ao usuário uma interface compreensível para definição de regras de contexto permitindo um controle facilitado e dinâmico dos serviços do AmbI.

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 *Handheld and Ubiquitous Computing*, pages 304–307. Springer.

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

Bezerra, L. (2011). Uso de ontologia em serviço de contexto e descoberta de recursos para autoadaptação de sistemas. Master's thesis.

Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59.

Brown, N. and Kindel, C. (1998). Distributed component object model protocol–dcom/1.

Cardoso, L. X. T. (2006). Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software.

Carvalho, S. T., Erthal, M., Mareli, D., Sztajnberg, A., Copetti, A., and Loques, O. (2010). Monitoramento Remoto de Pacientes em Ambiente Domiciliar. In *XXVIII SimpÃ³sio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010), pages 1005 – –1012, Gramado, RS, Brasil.*

Chen, G. and Kotz, D. (2002). Solar : An Open Platform for Context-Aware Mobile Applications. (June):41–47.

Chen, Y.S. and Chen, I.C. and Chang, W. (2010). Context-aware services based on OSGi for smart homes. Ubi-media Computing (U-Media), 2010 3rd IEEE International Conference on*, 11:392.*

de Araujo, R. (2003). Computação ubíqua: Princípios, tecnologias e desafios. XXI Simpósio Brasileiro de Redes de . . .*, pages 45–115.*

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. pages 1–50.

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.*

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. In AkÃÂit, M. and Matsuoka, S., editors, ECOOP'97 ÃÂÂ Object-Oriented Programming*, volume 1241 of* Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg.*

Kramer, J. (2007). Is abstraction the key to computing? Commun. ACM*, 50(4):36–42.*

Lee, Y., Iyengar, S., Min, C., Ju, Y., Kang, S., Park, T., Lee, J., Rhee, Y., and Song, J. (2012). Mobicon: a mobile context-monitoring platform. Communications of the ACM*, 55(3):54–65.*

Liu, H. and Parashar, M. (2003). Dios++: A framework for rule-based autonomic management of distributed scientific applications. Euro-Par 2003 Parallel Processing*, pages 66–73.*

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.*

Sudha, R., Rajagopalan, M., Selvanayaki, M., and Selvi, S. (2007). Ubiquitous semantic space: A context-aware and coordination middleware for ubiquitous computing. In Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, pages 1–7. IEEE.*

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.*

Wang, Q. (2005). Towards a rule model for self-adaptive software. ACM SIGSOFT Software Engineering Notes*, 30(1):8.*

Weis, T., Knoll, M., and Ulbrich, A. (2007). Rapid prototyping for pervasive applications. IEEE Pervasive.

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.*