Interpretação de Contexto 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 – Brasil

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

Resumo. *A sensibilidade ao contexto vem adquirindo cada vez mais visibilidade no atual cenário de desenvolvimento de aplicações. Este trabalho propõe nesta área uma nova solução para a representação, distribuição e interpretação de informações de contexto, possibilitando a construção de aplicações para ambientes inteligentes. Através da infraestrutura descrita neste artigo, os recursos do ambiente são expostos em um nível mais alto, tornando a criação de regras de contexto mais simples e dinâmica para desenvolvedores; e através de uma interface gráfica, usuários finais podem facilmente criar e personalizar regras no ambiente. Os conceitos propostos foram implementados em uma plataforma voltada à construção de smart homes.*

Abstract. *Context awareness is reaching increasingly visibility in the current applications development scenario. The goal of this paper is to propose a new solution for representing, distributing and interpreting context informations, thus allowing the construction of applications for intelligent ambients. The infrastructure described in this paper exposes the ambient resources in a higher level, leading to a simpler and more dynamic creation of context rules; and by means of a graphical interface, end users are able to easily create and customize rules in the ambient. The proposed concepts have been implemented as a platform focused on building smart homes.*

1. Introdução

Os recentes avanços no desenvolvimento da computação móvel em termos de custo, de miniaturização, de consumo de energia e outros, têm indicado um cenário favorável ao crescimento da aplicação da computação ubíqua. Este termo foi criado por Weiser (1991) para descrever os sistemas que permeiam o ambiente, identificando as necessidades dos usuários e oferecendo serviços. A partir dessas ideias surgiu o conceito de ambientes inteligentes (AmbI), onde uma variedade de dispositivos está disponível e conectada em rede, tanto fornecendo informações relevantes sobre o ambiente para aplicações e usuários, quanto atuando no ambiente (e.g., *smart homes*).

Diversos desafios surgem durante a criação de uma aplicação ubíqua do ponto de vista do desenvolvedor; e durante a configuração do ambiente, do ponto de vista do usuário. Em [Brush et al. 2011] são estudadas as diferenças entre as visões de longa data da computação ubíqua e a realidade, levantando questões referentes à capacidade de personalização do ambiente, confiabilidade, complexidade das interfaces de usuário, dentre outras, que impactam na aceitação pelos usuários finais. Para o desenvolvimento, existe uma carência de infraestrutura que permita lidar de forma consistente com sistemas

de grande complexidade. Como identificado por [de Araujo 2003], a heterogeneidade dos dispositivos envolvidos dificulta a criação de aplicações ubíquas no que concerne à comunicação. Além disso, a quantidade e a variedade de informações de contexto e serviços trazem um desafio à interatividade das aplicações.

Este trabalho tem como foco a interpretação de contexto, complementando o *framework* proposto em [Mareli et al. 2013], que provê uma infraestrutura e um suporte conceitual para a implementação de aplicações ubíquas. A solução aqui proposta engloba a representação da informação contextual, a distribuição desta informação no AmbI, e a interpretação do contexto baseada em regras. Adicionalmente é proposta uma interface para criação, configuração e testes das regras de contexto integrada à Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) [Barreto et al. 2013].

Os conceitos apresentados neste trabalho foram concretizados sobre uma plataforma denominada *SmartAndroid*, cujo desenvolvimento tem viabilizado a construção de casos de uso que evidenciam a efetividade da abordagem para prover sensibilidade ao contexto a novas aplicações.

O artigo está assim organizado: a Seção 2 apresenta os conceitos de suporte que orientam o desenvolvimento deste trabalho, incluindo a apresentação do *framework*, que este trabalho complementa; a Seção 3 apresenta uma proposta para a representação, distribuição e interpretação de informações de contexto, assim como para a personalização do ambiente em alto nível; na Seção 4 é apresentado o *SmartAndroid*, enfatizando a implementação dos conceitos aqui propostos; a Seção 5 traz os trabalhos relacionados; e a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Conceitos de Suporte

Este trabalho complementa o *framework* proposto em [Mareli et al. 2013], que provê uma infraestrutura e um suporte conceitual tipicamente requeridos em aplicações sensíveis ao contexto focadas em AmbI. Dentre as facilidades providas pela infraestrutura está a capacidade de abstrair detalhes das partes da aplicação que envolvem a comunicação, a aquisição de contexto e a localização de recursos. Como uma forma de abstração dos recursos são utilizados componentes denominados Agentes de Recursos (AR).

2.1. Sensibilidade ao Contexto

O contexto exerce um papel de fundamental importância na Computação Ubíqua. Sintetizando propostas anteriores, Dey e Abowd [Dey et al. 2001] definiram que contexto é qualquer informação relevante usada para caracterizar a situação de entidades, especificamente: 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 um idoso, que mora sozinho, liga o fogão e vai dormir, provavelmente o esqueceu ligado. Neste caso, um sistema sensível ao contexto faz o que qualquer pessoa faria se detectasse esta situação (i.e., acordaria o idoso e/ou desligaria o fogão). 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.

2.2. Agentes de Recurso

Os Agentes de Recurso (AR) podem ser compreendidos como entidades que representam recursos, definidos como sensores, atuadores, dispositivos e eletrodomésticos inteligentes, além de módulos de software que forneçam algum serviço para o ambiente. Os ARs encapsulam as especifidades dos recursos e expõem suas informações para o ambiente através de suas interfaces, de forma que outras entidades possam acessá-las de maneira uniforme diminuindo a complexidade de integração. Por exemplo, os detalhes da coleta de dados de um sensor de temperatura seriam conhecidos apenas pelo seu AR, que se encarrega de fornecer uma interface simples para outros componentes do sistema.

A estrutura geral do AR possibilita a resolução do desafio da heterogeneidade de dispositivos, como abordado em [Mareli et al. 2013]. Esta estrutura possui, dentre outros atributos, uma hierarquia de tipos, que é composta por uma sequência de classes, tendo sido inspirada na ontologia descrita em [Ranganathan et al. 2005] para classificar entidades. A hierarquia possibilita a consulta e instanciação de ARs a partir de propriedades associadas a uma classe específica.

A Figura 1 representa uma arquitetura em camadas para o *framework*. Na camada "Recursos" encontram-se os dispositivos, ou módulos de software, que efetivamente interagem com os usuários (e.g., cama, fogão, TV). Na camada "Infraestrutura" encontram-se os ARs (e.g., Agente Cama, Agente Fogão, Agente TV), que representam os recursos e expõem suas interfaces de maneira uniforme para as aplicações ou outros ARs. Encontra-se também na camada intermediária o Suporte ao Gerenciamento de Recursos (SGAR) (Subseção 2.3.2), e os ARs interpretadores e atuadores (Subseção 3.2), definidos no nível da aplicação, mas com o suporte da camada intermediária. Na camada "Aplicações" encontram-se aplicações desenvolvidas por terceiros (Seção 4) que usufruem de maneira segura das abstrações providas pela camada intermediária.

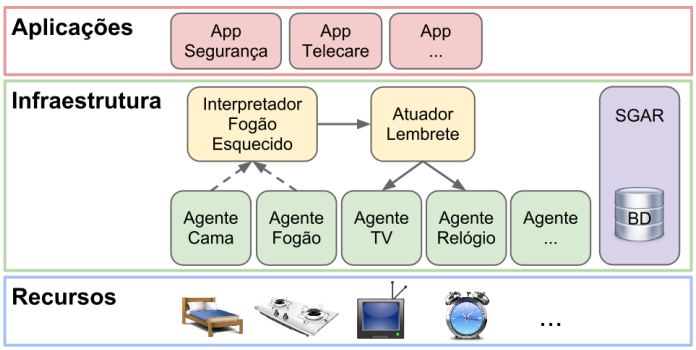

Figura 1. Arquitetura do *Framework*

2.3. Infraestrutura de Comunicação e Gerenciamento

O objetivo do *framework* é fornecer suporte à programação, teste e execução de aplicações, permitindo lidar de forma consistente com sistemas de grande complexidade. A infraestrutura do *framework* provê mecanismos de comunicação apresentados na Subseção 2.3.1, e inclui o AR como unidade básica de modularização, cuja gerência

é desempenhada pelo Suporte ao Gerenciamento de Agentes de Recursos (SGAR) (veja Subseção 2.3.2).

2.3.1. Comunicação e Segurança

De modo a atender requisitos típicos de ambientes distribuídos, o *framework* dispõe de dois mecanismos de comunicação: a invocação remota de procedimentos (*Remote Procedure Call* – RPC), para possibilitar a comunicação síncrona, e a comunicação por eventos seguindo o paradigma *publish-subscribe* [Eugster et al. 2003], para possibilitar a comunicação assíncrona.

Como suporte, a comunicação no *framework* requisita uma infraestrutura básica de rede. Por conveniência, foi adotado Wi-Fi, que garante o mínimo de segurança contra acesso de agentes externos à rede do ambiente e agrega benefícios de mecanismos de criptografia, como o WPA. São inclusos também no *framework*, mecanismos de segurança para garantias de permissão de acesso, distinção de contas de usuários (e.g., admin, usuário-adulto, usuário-criança, convidado), proteção do domínio de aplicações, etc, como os propostos em [Dixon et al. 2012].

2.3.2. Suporte ao Gerenciamento de Recursos

O SGAR é um conjunto de serviços responsável por gerenciar os ARs no AmbI. A Figura 1 apresenta o SGAR localizado na camada intermediária e, internamente, apresenta o Banco de Dados (BD), que corresponde ao Repositório de Recursos. Três componentes, ou serviços básicos, compõem o SGAR: o Serviço de Registro de Recursos, o Serviço de Descoberta de Recursos e o Serviço de Localização de Recursos.

É necessário que os recursos do ambiente sejam descobertos para que as aplicações possam utilizá-los. Com este intuito, o serviços de descoberta e de localização permitem a busca a ARs no Repositório de Recursos, que contém referências a ARs já registrados. Através do serviço de descoberta, podem ser feitas consultas baseadas tanto no nome, como no tipo do AR. O serviço de localização possibilita buscas baseadas na localização física (e.g., um AR em determinado cômodo, o AR de um tipo mais próximo de outro referenciado).

3. Proposta

A proposta deste trabalho consiste em oferecer uma solução para a representação da informação de contexto e para a sua distribuição, como apresentado na Subseção 3.1. A partir desta base, é apresentada uma solução para a interpretação de contexto na Subseção 3.2 e, adicionalmente, é apresentada a interface de usuário na Subseção 3.2.2.

3.1. Variáveis de Contexto e Operações

Um AR possui como interfaces as Variáveis de Contexto (VC) e as Operações (OP), que possibilitam a interação entre as aplicações e os recursos do AmbI. As VCs expõem as informações de contexto de cada AR. Por exemplo, o agente de uma *Smart TV* pode prover VCs definindo, por exemplo, qual a programação que está sendo exibida, qual a

programação agendada, se a TV está ligada, se a TV está gravando alguma programação, etc. Em outros termos, são definidas informações que dizem respeito ao estado da TV e que possam ser efetivamente coletadas.

Uma OP, por sua vez, tem o papel de expor uma funcionalidade (ou serviço) de um AR, possibilitando às aplicações interagirem ativamente no ambiente. Por exemplo, uma TV integrada ao sistema pode oferecer OPs para desligá-la, mudar o volume, gravar programações, mostrar mensagens na tela, pausar a programação, etc.

3.2. Interpretação de Contexto

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

Em nossa proposta, uma entidade denominada Interpretador de Contexto (IC) relaciona diversas informações de contexto por meio de uma expressão lógica que é avaliada quando o contexto muda, e notifica os agentes atuadores interessados quando a avaliação alcança um estado verdadeiro. Atuadores são quaisquer ARs que se subscrevem em ICs para desempenhar ações dado o recebimento de uma notificação do IC. As ações podem ser tanto no nível de software (e.g., guardar no histórico, enviar para um servidor remoto) quanto no nível do ambiente (e.g., mostrar uma mensagem na TV, disparar o despertador, mudar a temperatura do ar-condicionado).

As ligações entre ICs e Atuadores podem ser feitas e desfeitas dinamicamente, uma vez que são baseadas no mecanismo de subscrição de ARs. Um usuário que esteja personalizando o sistema (criando regras de contexto) é capaz de ligar ao IC não só um atuador que esteja criando, mas outros que já existem (veja exemplo na Seção 4).

Considere a seguinte regra simples: se uma pessoa (que mora sozinha) ligou o fogão, se deitou na cama, e passaram-se 15 minutos, então dispare o alarme do relógio e mostre na TV a mensagem “O fogão foi esquecido ligado!”. Este exemplo poderia ser implementando como representado na Figura 1. O IC recebe notificações da cama e do fogão com valores atualizados das VCs (e.g., “em uso” e “ligado”, respectivamente), e resolve internamente a temporização monitorada destas VCs (veja Subseção 3.2.1). Uma vez que o IC tenha avaliado a regra como verdadeira durante o tempo de 15 minutos (previamente declarado), ele notifica o atuador “Lembrete”. O atuador, por sua vez, invoca as OPs da TV e do relógio (e.g., “Mostrar mensagem” e “Disparar alarme”, respectivamente).

O IC é encapsulado em um AR, estendendo, assim, suas funcionalidades. Sua criação ocorre quando é definida uma expressão lógica no objeto do IC recém instanciado, segundo um padrão próprio adotado em arquivos tipo JSON, e um *parser* constrói uma estrutura lógica interna, como mostrado na Figura 2. A estrutura lógica abrigada pelo IC, contém uma estrutura em árvore que representa a expressão e contém não só a lógica entre as VCs das condições, como também os valores atualizados das mesmas. Enquanto em funcionamento, o IC recebe atualizações das VCs, tratando-as em seguida para atualizar sua estrutura lógica. O módulo de avaliação é notificado de mudanças da estrutura lógica e avalia se a expressão se tornou verdadeira ou falsa. Se a expressão alcançou um estado verdadeiro, considerando as possíveis temporizações, notifica os atuadores.

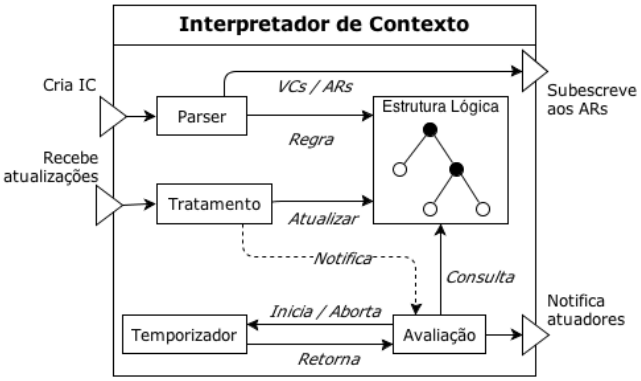

Figura 2. Estrutura do Interpretador de Contexto

O *framework* é suficientemente flexível para possibilitar a interpretação de contexto a partir dos ICs, assim como a partir da agregação de um motor de regras (e.g., Jess, Drools, CLIPS, motores proprietários). Um motor de regras é uma ferramenta geralmente centralizada, que recebe informações de contexto de diversas fontes e processa regras previamente inseridas, onde é comum usar um algoritmo como o RETE [Forgy 1982], que processa regras construídas apenas com a cláusula "E" eficientemente.

A multiplicação de regras de contexto no AmbI, inclusive por diferentes usuários e aplicações, pode facilmente levar a casos de conflito. O conflito ocorre quando dois ICs possuem expressões que se intercedem enquanto os respectivos atuadores possuem ações opostas. Por exemplo, considere as seguintes regras:

- Se fogão ligado e temperatura $>$ 140 por 15min, então reduza a temperatura;
- Se fogão ligado e temperatura $<$ 160 por 15min, então aumente a temperatura.

O usuário criou uma regra para controlar a temperatura do fogão, quando já havia outra cadastrada que tinha ação oposta mas que poderia ocorrer ao mesmo tempo. As regras estão, portanto, em conflito, pois não está claro como o sistema deve agir.

A detecção e a resolução de conflitos visam identificar a criação de regras contraditórias e prover uma solução, atendendo ao requisito de confiabilidade. A solução para este tipo de conflito envolve a comparação das expressões de um novo IC com as já existentes (no momento da ligação com o atuador); e, caso encontradas expressões que se intercedam, conferir as ações dos atuadores para ver se são do mesmo tipo mas com diferentes entradas.

3.2.1. Temporização

A temporização em um IC é definida ao se associar um ou mais temporizadores a uma subexpressão da regra (ou a expressão completa). Os temporizadores podem ser tanto relativos (e.g., fogão ligado e pessoa deitada na cama por 15min) quanto absolutos (e.g., ar-condicionado ligado às 18:00).

Durante a avaliação, se a expressão ou uma subexpressão que possuir um temporizador associado chegar em um estado verdadeiro, o temporizador é inicializado. Se, durante uma avaliação, a expressão ou subexpressão se torna falsa, então os temporiza-

dores associados são interrompidos. Se o tempo limite é alcançado e se não há outros temporizadores associados a subexpressões da mesma regra em andamento, então os atuadores são notificados.

3.2.2. Simulação e Gerenciamento

Visando atender o desafio da complexidade das interfaces de usuário, é proposta em [Barreto et al. 2013] uma ferramenta para gerenciamento de aplicações ubíquas/pervasivas e suporte à construção de protótipos, chamada IPGAP (Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas). A IPGAP fornece ao desenvolvedor um ambiente para controle e depuração de aplicações de maneira rápida e com baixo custo.

A Figura 3(a) ilustra a IPGAP executando em um *tablet*. O menu inferior da figura oferece algumas funcionalidades, tais como: criar um novo simulador, carregar no mapa um recurso já registrado, compor uma nova regra de contexto, e outras. O modo compor nova regra está atualmente em desenvolvimento, e tem como função possibilitar a usuários sem experiência técnica a criação ou edição de regras de contexto para o seu dia-a-dia. Além disso possibilita a desenvolvedores criar regras em alto nível e exportá-las para a construção de sua aplicação.

O roteiro para criação de uma regra na IPGAP envolve a criação da expressão da regra e a definição dos atuadores. A expressão da regra é criada por meio da definição de comparações (entre VC e um valor, ou entre duas VCs) e da definição da lógica que inter-relaciona essas comparações (usando os operadores lógicos "*E*", "*OU*" e "*NÃO*", e os parênteses), que ocorre por meio de toques na tela. Uma comparação é realizada por meio da seleção do AR no mapa e da escolha da VC (na Figura 3(a) são exibidas as VCs do fogão clicado), seguida da escolha de um operador de comparação ($=, \neq, <, >, \leq$ e $\geq$) e da definição de um valor, ou de uma outra VC de um AR selecionado. É possível também incluir temporizadores na comparação, na expressão ou em uma subexpressão (expressão entre parênteses). A atuação pode ser definida com a escolha de um ou mais atuadores já registrados no sistema (e.g., Atuador Lembrar Usuário), e/ou com a definição de um novo. A definição de um novo atuador, em semelhança à definição de uma comparação,

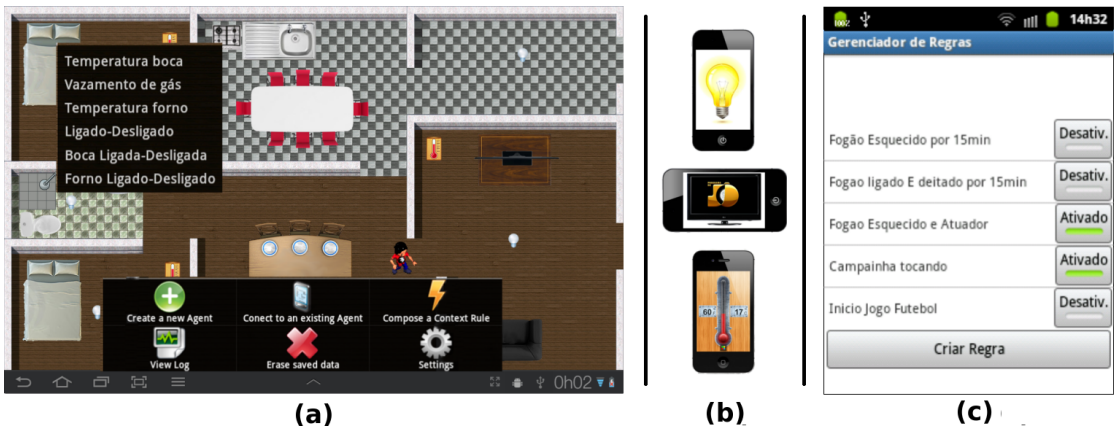

Figura 3. (a) IPGAP (b) Simuladores (c) Gerenciador de Regras

envolve a seleção de um AR no mapa, a escolha da OP e a escrita dos parâmetros (e.g., ar-condicionado: mudar temperatura para 20º C). Ao iniciar, a regra é compilada para o sistema e passa a funcionar.

A Figura 3(b) exibe alguns simuladores desenvolvidos e em funcionamento em *smartphones*. Como a IPGAP se liga aos ARs através do mecanismo de subscrição padrão, ela recebe as atualizações dos simuladores e as apresenta no mapa, assim como recebe as notificações de ARs de recursos reais. A Figura 3(c) mostra uma interface gráfica para gerência das regras de contexto, possibilitando (re)iniciar ou interromper a execução de uma regra. Este ferramental facilita não só a composição mas também o teste de regras de contexto, pois não exige a aquisição de dispositivos reais, sendo, portanto, útil tanto para desenvolvedores como para usuários finais.

4. SmartAndroid

O desenvolvimento deste projeto se consolidou na implementação da plataforma *SmartAndroid*[1]. O projeto contempla a implementação das APIs do *framework*, assim como a interface de prototipagem IPGAP, sobre a plataforma Android.

Algumas aplicações foram construídas para explorar o potencial do *framework* para sensibilidade ao contexto. Dentre estas aplicações podemos citar: o SmartLiC – uma aplicação de controle de iluminação residencial – que faz com que as luzes de um cômodo sejam apagadas conforme os usuários se ausentam deste; o MediaFollowMe, que faz com que um fluxo multimídia (e.g., vídeo, audio, imagens) seja repassado para o dispositivo (capaz de apresentar o fluxo) mais próximo do usuário, como se a mídia o seguisse; aplicações do tipo controle remoto, onde recursos são controlados remotamente por outro recurso (no caso, um *smartphone*); dentre outras aplicações. Um trabalho futuro é instalar no AmbI um sistema de assistência domiciliar a saúde, como em [Carvalho et al. 2010a].

4.1. Exemplo de Aplicação

A Regra do Fogão Esquecido, como abordado em seções anteriores, foi criada para atender a idosos que moram sozinhos e por vezes esquecem o fogão ligado, sendo um motivo de preocupação para a sua própria segurança e a de seus vizinhos. A utilização de elementos do dia-a-dia nesta regra (i.e., fogão, cama, TV), assim como em outras criadas, reflete o compromisso da abordagem em efetivamente tornar os recursos integrados ao AmbI.

Este caso de uso já foi abordado na Seção 3 e é ilustrado na Figura 1. Para sua implementação não foram usados recursos reais, mas criados simuladores para representá-los. Os ARs dos simuladores possuem uma hierarquia de tipos condizente com a de um recurso real, e as operações desencadeiam as reações apropriadas (e.g., ao se invocar a OP do forno "*Set Temperature*", gera uma notificação aos subscritos à VC "*Temperature*").

Algumas modificações foram feitas à regra de contexto original a fim de validar outras propriedades do IC. Na expressão original se fazia: "fogão ligado E pessoa deitada na cama POR 15min", o que permitiu validarmos a ligação entre as VCs e o IC, a temporização do IC, e a ligação com o atuador. A fim de verificarmos o funcionamento do temporizador em diferentes lugares na expressão da regra, criamos o exemplo um pouco mais sofisticado: "fogão ligado E (pessoa deitada na cama OU no sofá POR 15min)".

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

Assim, este IC é indiferente quanto ao tempo em que o fogão está ligado, se preocupando com a quantidade de tempo que ele foi deixado sozinho enquanto a pessoa dormia.

Para tolerar a presença de mais pessoas na casa, sem confundir quem está deitada na cama, bastou substituir, na expressão, a cama pelo sensor de presença da cozinha. A regra ficou assim: "fogão ligado E (cozinha vazia POR 15min)". Essa regra assume que, se de tempos em tempos (menos de 15min) alguém entra na cozinha e o fogão está ligado, então o fogão está sendo vigiado, senão foi esquecido.

A ligação com os atuadores foi feita de duas formas nos exemplos: declarando um atuador no arquivo da expressão da regra, definindo a ação (e.g., desligar fogão, mostrar certa mensagem na TV do quarto); e ligando um atuador existente a um IC também já instanciado (e.g., ligar outro atuador, que envia um SMS para alguém da família).

5. Trabalhos Relacionados

Em [Park et al. 2007] é proposto um sistema que foca na simulação de regras a fim de se descobrir possíveis conflitos, mas não provê suporte para a implantação das regras em ambiente de execução. Em [Dey et al. 2001] é proposto o Context Toolkit que implementa abstrações semelhantes às deste trabalho para sensibilidade ao contexto, incluindo o encapsulamento de recursos, agregação e interpretação de contexto, mas não atende requisitos de personalização do AmbI por usuários finais.

Alguns trabalhos utilizam um motor de regras para avaliar as regras de contexto [Wang 2005, Biegel and Cahill 2004]. Em [Liu and Parashar 2003] as regras são avaliadas em agentes distribuídos, mas controladas por um motor de regras centralizado. Embora o *framework* proposto não imponha a utilização de um motor de regras, e o *SmartAndroid* não faça uso de um, este poderia ser facilmente agregado e se aproveitar da infraestrutura de acesso aos recursos.

6. Conclusão e Trabalhos Futuros

Em ambientes inteligentes, aplicações ubíquas não são pequenas e monolíticas, mas sistemas complexos. Neste artigo foi proposta uma solução para a representação em alto nível de informações contextuais, distribuição de informações por meio da infraestrutura de suporte, e uma solução flexível para a interpretação do contexto. Além disso, uma interface para prototipagem e gerenciamento de aplicações ubíquas, que interage tanto com recursos reais como simulados, foi integrada à proposta a fim de facilitar a manipulação de regras de contexto. Este trabalho procurou, desta forma, atender tanto as necessidades de sensibilidade ao contexto dos desenvolvedores de aplicações ubíquas, como a necessidade de personalização de usuários finais.

A proposta possui potencial para atender a outros desafios importantes na área de computação ubíqua. Pretendemos identificar as preferências do usuário de forma menos intrusiva através de técnicas de mineração de dados aliadas à aprendizagem de máquina. As melhorias nas técnicas de identificação e resolução de conflitos são um problema de maior importância na interpretação de contexto, e este trabalho já está em andamento. A interface de composição e gerenciamento de regras de contexto integrada com o mapa deve também ser mais trabalhada. A fim de tratarmos questões relacionadas à evolução de uma família de aplicações ubíquas, pretendemos utilizar no *framework* uma abordagem como proposta em [Carvalho et al. 2010b].

Agradecimentos. Os autores agradecem ao CNPq, à CAPES e à FAPERJ pelo financiamento parcial deste trabalho.

Referências

Barreto, D., Erthal, M., Mareli, D., and Loques, O. (2013). Uma interface de prototipagem para aplicações pervasivas. *XXXI SBRC*.

Biegel, G. and Cahill, V. (2004). A framework for developing mobile, context-aware applications. In *Proc. PerCom 2004.*, pages 361–365. IEEE.

Brush, A., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., and Dixon, C. (2011). Home automation in the wild: challenges and opportunities. In *Proc. CHI 2011*, pages 2115–2124. ACM.

Carvalho, S., Erthal, M., Mareli, D., Sztajnberg, A., Copetti, A., and Loques, O. (2010a). Monitoramento remoto de pacientes em ambiente domiciliar. *XXVIII SBRC-Salão de Ferramentas, Gramado, RS, Brasil*.

Carvalho, S. T., Loques, O., and Murta, L. (2010b). Dynamic variability management in product lines: An approach based on architectural contracts. In *SBCARS 2010.*, pages 61–69. IEEE.

de Araujo, R. (2003). Computação ubíqua: Princípios, tecnologias e desafios. In *XXI SBRC*, 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.

Dixon, C., Mahajan, R., Agarwal, S., Brush, A., Lee, B., Saroiu, S., and Bahl, V. (2012). An operating system for the home. *Proc. NSDI 2012*.

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

Forgy, C. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37.

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

Mareli, D., Erthal, M., Barreto, D., and Loques, O. (2013). Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes. *XXXI SBRC*.

Park, J., Moon, M., Hwang, S., and Yeom, K. (2007). Cass: a context-aware simulation system for smart home. In *SERA 2007.*, pages 461–467. IEEE.

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

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

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.