Interpretação de Contexto em Ambientes Inteligentes

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

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

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

Abstract.

Resumo. *Aplicações cientes de contexto vêm adquirindo cada vez mais visibilidade no atual cenário de desenvolvimento de aplicações. São tratadas, neste artigo, questões relacionadas à representação das informações de contexto, à sua distribuição em ambientes inteligentes e à interpretação da informação de contexto, identificadas como problemas de primeira necessidade. A criação de regras de contexto possibilita que o ambiente interaja ativamente com pessoas ao prover serviços baseados em suas necessidades ou desejos. As regras são ligadas aos agentes provedores de contexto através de um mecanismo de subscrição, e reagem através do mesmo mecanismo, notificando os atuadores. Na implementação da proposta será também incluída uma interface gráfica que permite a criação de regras de contexto em alto nível, possibilitando que usuário finais definam suas preferências no ambiente de maneira simples.*

1. Introdução

Desde as propostas de Mark Weiser na década de 1990 [**?**], os pesquisadores da área de computação ubíqua/pervasiva vêm propondo mudanças na interação homem-máquina, visando tornar o uso de dispositivos cada vez mais transparente no ambiente. Isso possibilita ao usuário, manter o foco na tarefa a ser realizada e não na ferramenta para realizá-la. A partir dessas ideias surgiu o conceito de ambientes inteligentes (AmbI) [**?**], onde sensores e atuadores interconectados em rede são capazes de fornecer informações relevantes sobre o ambiente para aplicações e usuários, bem como, efetivamente, agir neste ambiente e alterar seu estado.

Estudos, como [**?**], ajudam a explicar a diferença entre as visões de longa data de AmbIs e a realidade, para os usuários. Uma das barreiras é a baixa capacidade de gerenciamento, que traz desafios, como o suporte à personalização, confiabilidade, e a complexidade das interfaces de usuário. Do ponto de vista dos desenvolvedores, a heterogeneidade dos dispositivos envolvidos dificulta a criação de aplicações ubíquas no que concerne à comunicação, assim como a quantidade e a variedade de informações de contexto e serviços é um desafio à interatividade das aplicações; como identificado por [**?**].

Muitos trabalhos têm por objetivo definir *frameworks* voltados à construção e ao gerenciamento de aplicações ubíquas [**?**, **?**, **?**]. Em [**?**] são apontados desafios na aquisição de conhecimentos do ambiente. Em [**?**] 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 [**?**] são propostos serviços

para gerenciar componentes representativos do ambiente no nível de *middleware*. O trabalho de [**?**], em especial, se preocupa em organizar estes componentes de forma a facilitar suas manipulações; a estruturação proposta permite ampliar o escopo de operações de suporte de um sistema ubíquo. Em [**?**]... Estes trabalhos, no entanto, não têm como foco a integração das questões acima identificadas: *suporte à personalização*, *confiabilidade*, e *complexidade das interfaces de usuário*, a fim de se tratar a *baixa capacidade de gerenciamento* das soluções disponíveis atuais.

Este trabalho foca no gerenciamento de AmbIs baseado em regras de contexto. A solução 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 a criação das regras de contexto integrada à Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) [**?**], voltada à visualização e ao teste de aplicações ubíquas, mesclando componentes reais e virtuais.

A proposta foi concretizados sobre uma plataforma denominada *SmartAndroid*[1], desenvolvida no contexto do projeto. Esta implementação tem viabilizado avaliações

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

2. Conceitos Básicos

Este trabalho complementa o *framework*, proposto em [**?**], que provê facilidades e padrões de programação tipicamente requeridos em aplicações sensíveis ao contexto focadas em AmbI, definidas na Seção 2.1. Dentre as facilidades está a capacidade de abstrair detalhes das partes da aplicação que envolve a comunicação, a aquisição de contexto e a localização de recursos. Como abstração aos recursos são utilizados componentes chamados Agentes de Recursos (AR), definidos na Seção 2.2. O gerenciamento dos ARs – incluindo o registro dos mesmos, a consulta, a localização e o gerenciamento do ciclo de vida – é descrito na Seção 2.3.2.

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

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

2.1. Sensibilidade ao Contexto

O contexto exerce um papel de fundamental importância na Computação Ubíqua. Sintetizando propostas anteriores, Dey e Abowd [**?**] definiram que contexto é qualquer informação relevante usada para caracterizar a situação de entidades, especificamente: pessoas, lugares e coisas. No AmbI, "Lugares" são os cômodos, os andares de uma edificação, ou espaços em geral que possibilitam a localização de outras entidades; "Pessoas" são indivíduos que povoam o ambiente e interagem com o mesmo; e "Coisas" são representações virtuais de objetos físicos ou componentes de software.

A sensibilidade ao contexto está em se determinar o que o usuário está tentando realizar a partir da aquisição de contexto. Por exemplo, se uma pessoa sai de casa e deixa a torneira aberta, provavelmente a tenha esquecido nesse estado. Neste caso, um sistema sensível ao contexto faz o que qualquer pessoa faria se detectasse esta situação (i.e., fecharia as torneiras). A aquisição do contexto de forma automatizada contribui para a construção de aplicações para AmbI que, de outra maneira, seriam inviáveis, por exigir a entrada de dados ou comandos diretamente por parte dos usuários. Este trabalho oferecer um suporte para a declaração e consulta das informações de contexto, assim como para a interpretação destas informações, possibilitando a construção de regras de contexto para atuar no AmbI.

2.2. Agentes de Recurso

Sensores, atuadores, dispositivos e eletrodomésticos inteligentes, além de módulos de *software* que forneçam algum serviço para o ambiente, são definidos como *recursos*. Estes são encapsulados em entidades chamadas Agentes de Recurso (AR), que podem ser compreendidos como elementos que expõem informações dos recursos juntamente com sua interface, de forma que outras entidades possam acessá-las de maneira uniforme.

Os ARs escondem detalhes de baixo nível do recurso encapsulado, diminuindo significativamente a complexidade de integração de um recurso no sistema. Por exemplo, os detalhes da coleta de dados de um sensor de temperatura seriam conhecidos apenas pelo seu AR, que se encarrega de fornecer uma interface simples para que os outros componentes do sistema tenham acesso as informações desse sensor.

Proposto inicialmente em [**?**] como uma entidade de coleta de informações de contexto, a estrutura do AR permite resolver o desafio da *heterogeneidade de dispositivos*, anterioremente identificado.

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 do "Middleware" encontram-se os ARs (e.g., Agente da Cama, Agente do Fogao, Agente da 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 (Seção 2.3.2), e os ARs interpretadores e atuadores (Seção 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.

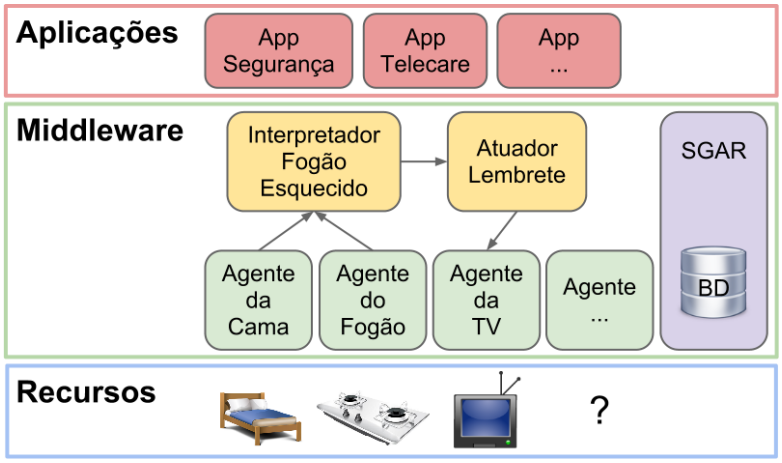

Figura 1. Arquitetura do *Framework*

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. O AR é a unidade básica de modularização do sistema, sendo gerenciados pelo Suporte ao Gerenciamento de Agentes de Recursos (SGAR).

2.3.1. Comunicação

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) e a comunicação por eventos seguindo o paradigma *publish-subscribe* [**?**]. O mecanismo de RPC é utilizado para estabelecer a comunicação síncrona direta entre os componentes do AmbI. Uma entidade invocadora utiliza um *proxy* da entidade invocada, nele estando contida a sua interface de chamadas de procedimentos públicos. A chamada e o retorno do procedimento são enviados através de mensagens serializadas (no *SmartAndroid* através da notação JSON). Este mecanismo pode ser usado para enviar comandos aos dispositivos (e.g, fechar uma torneira, alterar remotamente o *setup* de um ar-condicionado).

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 a WPA.

2.3.2. Suporte ao Gerenciamento de Recursos

O SGAR é o serviço 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 [**?**].

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 Recurso (SRR), o Serviço de Descoberta de Recursos (SDR) e o Serviço de Localização de Recursos (SLR). O SRR é utilizado para registrar (*register*) ou remover (*unregister*) ARs no ambiente, o SDR localiza recursos para permitir o acesso remoto (*search*) e o SLR os localizam através de posições físicas e indica os recursos mais próximos (*searchByProximity*) [**?**].

3. Proposta

A proposta deste trabalho é oferecer uma solução para a representação da informação de contexto e para a sua distribuição através do modelo de contexto apresentado na Seção 3.1. A partir do modelo de contexto é suportada a interpretação de contexto, apresentada na Seção 3.2.

3.1. Modelo de Contexto

O Modelo de Contexto foi elaborado com o objetivo de tratar a questão da *variedade de informações de contexto e serviços* presentes no AmbI. Neste modelo, o *framework* utiliza uma abordagem flexível, onde o mecanismo de comunicação por eventos (visto na Seção 2.3.1) possibilita que ARs divulguem mudanças de contexto. A informação pode estar em qualquer lugar do AmbI, contudo, é provido um nível de abstração tal que a subscrição ou consulta a qualquer informação de contexto é feita de maneira padronizada. O processo de aquisição do contexto envolve a descoberta do AR de interesse através dos serviços da SGAR, com a posterior subscrição do mesmo. A abordagem adotada é flexível de tal maneira que, caso seja conveniente, pode ser criada uma unidade centralizadora para gerenciar as informações de contexto, segundo uma arquitetura de memória compartilhada do tipo *blackboard* (ou quadro-negro) [**?**].

A estrutura geral do AR, possui, dentre outros atributos, uma hierarquia de tipos que é composta por uma sequência de classes, tendo sido inspirada na ontologia descrita em [**?**] para classificar entidades (como os ARs). A hierarquia possibilita a consulta e instanciação de ARs a partir de propriedades associadas a uma classe específica. Na Seção 2.3.2 é apresentado o SGAR que se beneficia desta estrutura do AR.

3.1.1. Variáveis de Contexto e Operações

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

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

Tanto VCs como OPs definem interfaces, ou portas do AR. Diferentes aplicações, instaladas no mesmo ambiente podem usar estas interfaces para interagir com o próprio

ambiente. A Figura 2 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.

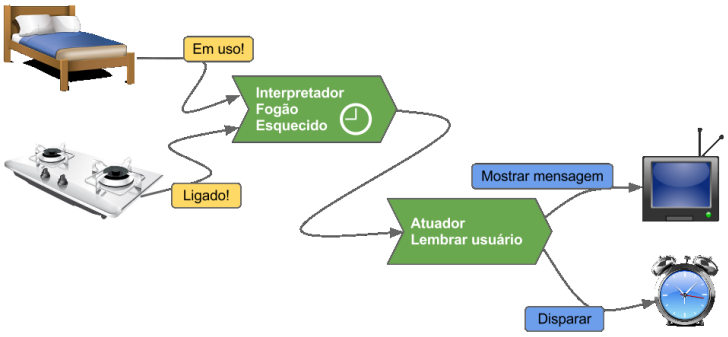

Figura 2. Interpretação de Contexto

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 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, como opção básica, a interpretação do contexto é desempenhada por entidades chamadas Interpretadores de Contexto (IC), que avaliam essas informações e notificam agentes atuadores interessados (ver outras opções na Seção 4.2). Atuadores são quaisquer ARs que se subscrevem em ICs para desempenhar ações dado a ocorrência do evento capturado pelo 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, ao se chamar OPs de outros ARs (e.g., mostrar uma mensagem na televisão, 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.

A interpretação de contexto possibilita que diversas informações de contexto, relacionadas através de uma expressão lógica, sejam manipuladas em uma entidade em separado, provendo separação de interesses. O interessado poderá referenciar apenas uma entidade, ao invés de várias, a fim de se obter a informação interpretada.

Considere a seguinte regra simples: se uma pessoa (que mora sozinha) ligou o fogão, se deitou na cama, e passaram-se 15 minutos, então dispare o alarme do despertador e mostre na televisão a mensagem “O fogão foi esquecido ligado!”. Este exemplo poderia ser implementando no sistema como representado na Figura 2. O IC recebe notificações da TV e do fogão com valores atualizados das VCs “ligado” e “em uso”, respectivamente, e resolve internamente a temporização monitorada destas VCs. Uma vez que o IC tenha avaliado a regra como verdadeira durante o tempo de 15 minutos (previamente declarado), ele notifica o atuador “Lembrar usuário”. O atuador, por sua vez, invoca as OPs “Mostrar mensagem” da TV e “Disparar” do despertador.

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

A a estrutura do IC, como mostrada na figura **??** é composta por um "Parser", que faz a tradução da regra escrita para o modelo usado pelo IC; a Estrutura Lógica

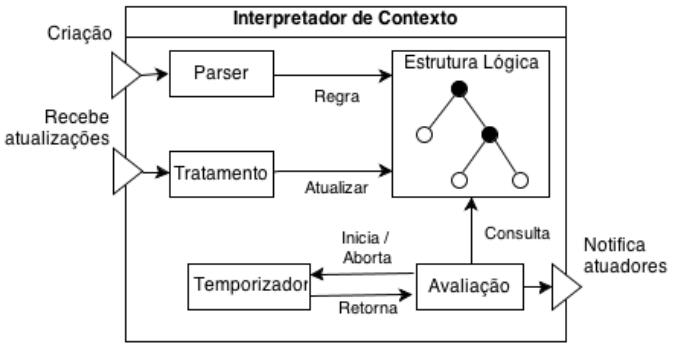

Figura 3. Estrutura do Interpretador de Contexto

3.3. Temporização

A temporização em um IC é definida ao se associar temporizadores ("*timers*") à uma condição expressão da regra (ou subexpressões). 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).

No caso dos temporizadores relativos, durante o tempo indicado o valor ou expressão deve permanecer

3.4. Prototipagem e Simulação

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

A prototipagem de aplicações pervasivas é fundamental para a depuração e teste em um AmbI. A questão de *disponibilidade de recursos* é resolvida através da IPGAP [**?**], um aparato ferramental que permite ao desenvolvedor de aplicações ter acesso a estes

benefícios. Além disso, a IPGAP disponibiliza uma Interface Gráfica de Usuário (*Graphic User Interface* – GUI) que proporciona ao usuário final controlar remotamente recursos do ambiente.

Figura 4. Interface de Prototipagem

Na Figura 4 é ilustrado um *tablet* executando o aplicativo da IPGAP e *smartphones* emulando recursos como TV, lâmpada e termômetro. A IPGAP faz o mapeamento do ambiente e permite visualizar e instanciar cada um destes elementos e representá-los na sua GUI. Outros recursos, além dos representados pelos *smartphones*, podem ser simulados virtualmente por componentes de software. Assim, o desenvolvedor de aplicações não fica limitado pela disponibilidade de recursos físicos podendo simular os elementos indisponíveis.

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

3.5. Conflitos

4. SmartAndroid

Como suporte, a comunicação no *framework* necessita de uma infraestrutura básica de rede. Por conveniência, foi adotado Wi-Fi, que garante o mínimo de segurança contra acesso de agentes externos à rede do ambiente e agrega benefícios de mecanismos de criptografia, como a WPA.

4.1. Regra Fogão Esquecido

4.2. Aplicação de Controle de Iluminação Residencial

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

A Figura 2 representa um caso simples de utilização de interpretadores, onde este contém uma regra apenas e as atuações ocorrem em separado, desempenhadas pelos agentes atuadores. A fim de se prover maior flexibilidade para os desenvolvedores, o *framework* possibilita também que a interpretação de contexto seja desenvolvida no nível da aplicação, contudo, aproveitando o mecanismo de comunicação por eventos e compilações de regras. Note que esta abordagem possibilita a utilização de um motor de regras (e.g., Jess, Drools), se for conveniente.

O Código 2 ilustra uma possível implementação da interpretação de contexto pela aplicação SmartLiC. Na etapa de inicialização, a aplicação utiliza o SLR para localizar os cômodos (Linha 2), e obter a lâmpada e o sensor de presença ("lampAR" e "presAR") de cada um, utilizando a busca do SDR (Linhas 4 e 5). Para evitar a repetição do código da regra, suas referências e a respectiva ação a ser desempenhada (desligar a lâmpada referenciada – "lampAR.turnOff") são adicionadas na lista de configurações ("configList") (Linha 6). A regra é criada em "rule" como: "*lâmpada ligada E o cômodo desocupado POR T_{max} unidades de tempo*", e desempenha uma relação lógica entre as diferentes VCs e a temporização da regra (Linha 7). Em seguida, subscreve-se às VCs dos ARs de interesse, começando o processamento da regra (Linha 8).

Na etapa de avaliação, também descrita no Código 2, é definido um bloco de comando para o tratamento das notificações recebidas dos ARs de interesse (lâmpadas e sensores de presença subscritos). Uma vez recebido um evento com a atualização de uma VC de um dos ARs ("ar_ref") (Linha 11), é selecionada a configuração de "configList" respectiva (Linha 12). Segue-se a avaliação da regra ("*evaluate*"), utilizando a configuração obtida, e considerando os temporizadores (Linha 13). A etapa de avaliação é bloqueante no caso da existência de temporizadores associados, e a invocação das ações (OPs) ocorre na mesma *thread* que a avaliação. A temporização ocorre enquanto a expressão da regra é válida, se eventos chegarem antes do fim da temporização e isto fizer com que a expressão da regra seja invalidada, então o temporizador é finalizado e as ações não são disparadas.

Código 1. Interpretação de Contexto no SmartLiC

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

5. Trabalhos Relacionados

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 de informações contextuais, distribuição das mesmas em ambientes inteligentes, e interpretação do contexto, facilitando o desenvolvimento de aplicações ubíquas. A proposta possibilita que sejam criadas regras de contexto em alto nível a partir de expressões lógicas, que relacionam variáveis de contexto entre si e com temporizadores.