UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS DE SÁ ERTHAL

UMA PROPOSTA PARA A INTERPRETAÇÃO DE CONTEXTO EM AMBIENTES INTELIGENTES

NITERÓI

2014

UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS DE SÁ ERTHAL

UMA PROPOSTA PARA A INTERPRETAÇÃO DE CONTEXTO EM AMBIENTES INTELIGENTES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Orientador:
Orlando Gomes Loques Filho, Ph.D.

Niterói

2014

MATHEUS DE SÁ ERTHAL

UMA PROPOSTA PARA A INTERPRETAÇÃO DE CONTEXTO EM AMBIENTES INTELIGENTES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Aprovada em março de 2014.

BANCA EXAMINADORA

Prof. Orlando Gomes Loques Filho, IC/UFF – Orientador

Prof. Alexandre Sztajnberg, DICC/UERJ

Prof. José Viterbo Filho, IC/UFF

Niterói
2014

À minha família.

Agradecimentos

Agradeço primeiramente à minha família e à Alice, que enfrentaram juntos comigo os percalços desta jornada e que tiveram paciência nos meus momentos de ausência. Sempre com palavras de apoio para os dias de tempestade e compartilhando da alegria dos dias ensolarados.

Agradeço aos ensinamentos do Professor e Orientador Orlando Loques, que me auxiliaram a galgar mais este degrau.

Agradeço aos colegas do Laboratório Tempo e amigos David Barreto e Douglas Mareli, que foram fundamentais para o desenvolvimento deste trabalho. Ao Giulio e seus conselhos sempre oportunos. Ao Sérgio, que mesmo longe nunca deixou de sair de perto. E ao André, Breno, Edhelmira, Gustavo e Tácio pelo apoio e pela força.

Agradeço ao Marcelo Roque e ao Eduardo Régis pela tolerância com o dinamismo da minha agenda. E aos demais amigos e amigas que fazem parte do meu dia-a-dia e não foram mencionados aqui, mas que guardo especial carinho.

Por fim, agradeço à CAPES, pelo incentivo à pesquisa através da bolsa de Mestrado; à FAPERJ, pelo financiamento parcial deste projeto; e ao programa de Pós-Gradução do IC/UFF, que ofereceu toda a infraestrutura necessária para a minha formação e consolidação como Mestre em Computação.

Resumo

A sensibilidade ao contexto possibilita que as aplicações ubíquas/pervasivas não apenas tenham ciência do contexto do ambiente, mas reajam à mudança desse contexto e à passagem do tempo. Com a evolução das tecnologias de comunicação sem fio e computação móvel, surge um cenário favorável para a implementação destas tecnologias, trazendo desafios para os desenvolvedores que devem lidar com uma série de questões características deste tipo de ambiente.

Este trabalho propõe uma solução para a interpretação de informações contexto integrada a um *framework* conceitual que se foca em três desafios pertinentes à implantação de um ambiente inteligente, a saber: a heterogeneidade dos dispositivos, a quantidade e diversidade de informações de contexto e serviços, e a dificuldade no desenvolvimento e teste de aplicações ubíquas. Mais especificamente, este trabalho foca nas questões referentes à sensibilidade ao contexto que permeiam esses desafios.

Os conceitos do *framework* foram consolidados em uma plataforma, o *SmartAndroid*. Para facilitar a construção de regras de contexto sem perda de flexibilidade, foi também implementado um *parser* que lê as regras de arquivos e instancia interpretadores e atuadores que as implementam. Da perspectiva dos usuários finais, foi desenvolvido uma interface de composição de regras, que permite a usuários sem experiência técnica definir suas preferências com simples gestos em uma tela. Essas implementações demonstraram a aplicabilidade da proposta e permitiram o desenvolvimento de um conjunto de aplicações exemplo e regras de contexto para sua validação.

Palavras-chave: Computação Sensível ao Contexto, Computação Ubíqua, Regras de Contexto, Prototipagem Rápida.

Abstract

Context awareness enables ubiquitous/pervasive applications not only to become aware of the environment's context, but also to respond to changes in this context and the passage of time. Due to advances in wireless technology and mobile computing, a favorable scenario for the implementation of these technologies emerges bringing challenges to developers who must deal with a number of issues relevant to these environments.

This work proposes a solution for the interpretation of context information integrated to a conceptual framework that focuses on three relevant challenges to the implementation of an intelligent environment, namely the device heterogeneity, the amount and diversity of context information and services, and the difficulty in the development and testing of ubiquitous applications. More specifically, this work focuses on issues related to context awareness that pervade these challenges.

We implemented the concepts of the framework in a platform named *SmartAndroid*. To facilitate the composition of context rules with no loss of flexibility we also developed a parser that reads rules from files and instantiates the proper interpreters and actuators that implement them. From the end-user point of view, we developed an interface for the composition of context rules that allows users without technical knowledge to set their preferences with simple gestures on a screen. These implementations have demonstrated the applicability of the proposal and allowed the development of a set of example applications and context rules for its validation.

Keywords: Context Aware Computing, Ubiquitous Computing, Context Rules, Rapid Prototyping.

Lista de Figuras

Lista de Listagens

Lista de Tabelas

Lista de Abreviaturas e Siglas

AmbI	:	Ambiente Inteligente;
BD	:	Banco de Dados;
GUI	:	Graphical User Interface;
IC	:	Interpretador de Contexto;
IPGAP	:	Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas;
JSON	:	JavaScript Object Notation;
OP	:	Operação do Agente de Recurso;
PubSub	:	Publish-Subscribe;
RPC	:	Remote Procedure Call;
RR	:	Repositório de Recursos;
SDR	:	Serviço de Descoberta de Recursos;
SGAR	:	Suporte ao Gerenciamento de Agentes de Recursos;
SLR	:	Serviço de Localização de Recursos;
SRR	:	Serviço de Registro de Recursos;
VC	:	Variável de Contexto;

Sumário

Capítulo 1

Introdução

Neste capítulo são apresentadas a motivação, que guiou o desenvolvimento desta dissertação, assim como os objetivos e as principais contribuições. Também é apresentada uma organização do texto, com uma breve descrição dos capítulos.

1.1 Contextualização

Os recentes avanços no desenvolvimento da computação móvel, principalmente em termos de custo, de miniaturização e de consumo de energia, indicam um cenário favorável ao crescimento da aplicação da Computação Ubíqua. Este termo foi criado por Weiser em 1988 [54] para descrever os sistemas que permeiam o ambiente, identificando as necessidades dos usuários e oferecendo serviços. Também chamada de "computação calma", a Computação Ubíqua descreve uma mudança no paradigma de interação entre o usuário e os sistemas computacionais, onde a interação entre os usuários e os computadores ocorre de forma natural, sem ações explícitas.

Uma aplicação ubíqua identifica as necessidades de seus usuários coletando por meio de sensores as informações do seu contexto de execução, e as atende provendo serviços, por meio de atuadores, os quais incluem diversos tipos de interfaces. Esta ideia de tornar a computação disponível às pessoas de maneira não intrusiva em um ambiente, minimizando interações explícitas, define um Ambiente Inteligente (AmbI) [1], que são ambientes onde uma variedade de dispositivos estão disponíveis e conectados em rede, tanto fornecendo informações contextuais relevantes sobre o ambiente para aplicações e usuários, quanto atuando no ambiente (e.g., *smart homes*).

O contexto exerce um papel de fundamental importância na construção de AmbIs.

Diversos autores no início da década de 1990 propuseram definições para "Contexto", embora a maioria seja pouco precisa [2]. Sintetizando propostas anteriores, Dey e Abowd [18] propuseram a seguinte definição para contexto:

> "Qualquer informação que possa ser utilizada para caracterizar a situação de entidades (pessoa, lugar ou objeto) que sejam consideradas relevantes para interação entre um usuário e uma aplicação (incluindo o usuário e a aplicação)."

Esta definição é uma das mais aceitas e utilizadas na área pelos seguintes motivos: por ser ampla o suficiente para atender às demandas de diferentes tipos de aplicações e sistemas; por ser imparcial, no que concerne às variedades e tipos de informações contextuais; por não restringir fontes de informações contextuais; e também por ser precisa, mesmo sem listar tipos e classes específicas de contexto [32].

1.2 Motivação

Diversos desafios surgem derivados da criação e do gerenciamento de aplicações ubíquas por parte dos desenvolvedores; e durante a configuração do ambiente por parte dos usuários. Trabalhos como [8, 21, 30, 45, 48] levantam alguns desses desafios.

Em [8] 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, complexidade das interfaces de usuário, dentre outras, que impactam na aceitação pelos usuários finais.

Do ponto de vista do desenvolvimento, existe uma carência de infraestrutura que permita lidar de forma sistemática com sistemas simples, assim como os de grande complexidade. Como identificado por [17], a (i) *heterogeneidade dos dispositivos* envolvidos dificulta a criação de aplicações ubíquas no que concerne à comunicação. Além disso, a (ii) *quantidade e a variedade de informações de contexto e serviços* trazem um desafio à interatividade das aplicações, e para a execução de testes durante o desenvolvimento existe uma carência na (iii) *disponibilidade de recursos*, incluindo sensores, atuadores, novos dispositivos embarcados e até o próprio espaço físico. O desafio (iii) também está relacionado aos usuários finais, que possuem demandas de personalização do seu ambiente.

1.3 Objetivo

Este trabalho tem como principal objetivo propor uma solução para a construção de aplicações sensíveis ao contexto (ou cientes de contexto) em AmbIs. A sensibilidade ao contexto é adquirida através da interpretação de informações contextuais provenientes de fontes diversas, de tal maneira que uma aplicação possa avaliar se o ambiente alcançou uma condição em específico e desempenhar alguma ação em resposta. Este padrão condição-ação em um AmbI é denominado regra de contexto.

Complementarmente ao suporte conceitual e de implementação apresentado em [33], este trabalho apresenta uma solução para a criação de aplicações que estejam não só cientes do contexto, mas que possam criar regras de contexto que reajam às modificações e sejam capazes, inclusive, de atuar no ambiente, ao interagir com dispositivos físicos por meio de um *middleware*. A fim de validarmos os conceitos propostos, o *middleware* foi consolidado na implementação de referência, o *SmartAndroid*.

Em vista das necessidades de usuários controlarem a execução de regras no ambiente, de acompanhar ações efetuadas, assim como, de criar regras de contexto que representem suas preferências e demandas, implementou-se no *SmartAndroid* uma interface visual e de manipulação que viabiliza a usuários sem experiência técnica desempenhar essas ações. Esta interface foi desenvolvida como parte da Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) [3], que provê uma solução para a prototipagem de aplicações para AmbI, diminuindo seus custos de produção, dentre outros benefícios, e também uma solução para o gerenciamento de recursos do AmbI.

A plataforma *SmartAndroid* desenvolvida em conjunto com esse projeto, tem viabilizado a construção de casos de uso que evidenciam a efetividade da abordagem para prover sensibilidade ao contexto à novas aplicações. Além disso, foram elaboradas questões de competência [26, 49] que avaliam a capacidade do *framework* de atender os desafios apontados previamente.

1.4 Contribuições

Ao longo do desenvolvimento desta dissertação foram produzidos três artigos com alguns dos resultados obtidos, são eles:

1. *Um Framework de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes*, SBRC 2013 [34].

2. *Uma Interface de Prototipagem para Aplicações Pervasivas*, SBRC 2013 [23].

3. *Interpretação de Contexto em Ambientes Inteligentes*, SBCUP 2013 [22].

No primeiro artigo da lista [34] são apresentadas as características gerais do *framework*, incluindo suas abstrações para os recursos do ambiente e serviços para manipulação dos recursos. No segundo artigo [23] há um maior foco no suporte para os usuários finais e para a prototipagem de aplicações por desenvolvedores. Por fim, o terceiro artigo [22] se concentra na solução para a interpretação do contexto.

1.5 Organização

A dissertação está organizada em seis capítulos, e estruturada como a seguir:

Capítulo 1: Introdução (este capítulo)

Apresenta a motivação para o desenvolvimento deste trabalho e descreve brevemente as contribuições

Capítulo 2: Conceitos Básicos

Apresenta os conceitos de suporte que orientam o desenvolvimento deste trabalho, e apresenta o suporte geral do *framework* conceitual

Capítulo 3: Proposta

Apresenta uma proposta para a construção e gerenciamento de aplicações sensíveis ao contexto em AmbI

Capítulo 4: Avaliação

Apresenta uma avaliação para a proposta baseada no desenvolvimento de questões de competência

Capítulo 5: Trabalhos Relacionados

Identifica o estado da arte da área e traz uma comparação do trabalho proposto com outras abordagens

Capítulo 6: Conclusão

Apresenta as considerações finais e trabalhos futuros

Capítulo 2

Conceitos Básicos

Neste capítulo serão abordadas as áreas onde esta dissertação está inserida, que incluem Sistemas Distribuídos (Seção 2.1), Computação Ubíqua/Pervasiva (Seção 2.2), Ambientes Inteligentes (Seção 2.3) e Computação Sensível ao Contexto (Seção 2.4) . Além disso, serão apresentados detalhes sobre a unidade básica de modularização do *framework* (Seção 2.5), os paradigmas de comunicação utilizados (Seção 2.6) e os serviços agregados (Seção 2.7).

Os conceitos utilizados nesta proposta se baseiam no estado da arte, contando com propostas novas e um refinamento das pesquisas realizadas em nosso grupo [5, 9, 15, 41]. Uma implementação de referência derivada destas propostas foi realizada, dando origem ao projeto *SmartAndroid*[1], que contempla o desenvolvimento dos serviços de gerenciamento dos recursos, APIs e a Interface de Prototipagem utilizando a plataforma Android.

2.1 Sistemas Distribuídos

O campo de Sistemas Distribuídos surge da intercessão entre a computação pessoal e as redes locais. Pesquisas realizadas desde meados de 1970 até o início da década de 1990 criaram um *framework* conceitual e uma base algorítimica que provaram ser de valor duradouro em trabalhos envolvendo dois ou mais computadores conectados em rede [45]. Muitas áreas surgiram a partir desta grande área, incluindo a Computação Ubíqua [16].

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

Em Coulouris 2005 [16] Sistemas Distribuídos são definidos como:

> "Aqueles nos quais componentes de hardware ou de software localizados em rede se comunicam e coordenam suas ações somente passando mensagens."

Esta definição cobre a ampla gama de sistemas nos quais computadores em rede podem ser proveitosamente implantados.

2.2 Computação Ubíqua

A Computação Ubíqua representa uma mudança no paradigma de interação entre pessoas e computadores. Pode-se dividir a utilização de computadores em três fases [55], ou eras. A primeira corresponde ao período em que um computador era um recurso escasso e atendia a diversos usuários, caracterizada como a fase dos *mainframes*. A segunda fase está relacionada aos computadores pessoais, ou PCs, onde cada pessoa interage com um computador exclusivamente. A terceira fase, a da Computação Ubíqua, como previsto por Weiser [54], é o período onde cada pessoa interage com diversos computadores, além disso, os computadores variam não só em número, mas também em forma e função [16], e são encontrados nos objetos mais triviais, como canetas, xícaras e etiquetas de roupas.

Os termos "ubíquo" e "pervasivo" são usados frequentemente como sinônimos. O termo "ubíquo", segundo o dicionário Aurélio[2], significa "que está ao mesmo tempo em toda parte, omnipresença", e caracteriza um tipo de sistema onde os computadores, ou dispositivos, estão espalhados no ambiente, "invisíveis". O termo "pervasivo" não existe em português, mas, como é comum na área, está sendo traduzido da palavra em inglês *pervasive*, que significa "disseminado, difuso".

2.3 Ambientes Inteligentes

Um ambiente inteligente é definido como um espaço físico que dispõe de serviços embarcados [16], ou seja, serviços que são providos exclusivamente, ou principalmente, dentro deste espaço físico. Um espaço físico geralmente é imaginado como uma sala de estar ou corredor, mas é possível também introduzir dispositivos computacionais em meio selvagem, onde não existe infraestrutura para executar uma aplicação (e.g., monitoramento ambiental).

[2] www.dicionariodoaurelio.com/

Em [43] é proposto o Gaia, que define um espaço físico como uma região geográfica com fronteiras limitadas e bem definidas contendo objetos físicos, dispositivos de rede heterogêneos e usuários desempenhando uma gama de atividades. Além disso, é definido como espaço ativo (*active space*) um espaço físico coordenado por uma infraestrutura de software sensível ao contexto que aumenta a habilidade de usuários móveis de interagir com e configurar seu ambiente físico e digital facilmente. O conceito de "espaço ativo" do Gaia é equivalente ao de ambientes inteligentes utilizado neste trabalho.

Uma classe de ambientes inteligentes mais comumente encontrada na literatura são as *smart homes*, que focam no ambiente doméstico para a provisão de serviços computacionais para seus residentes [21]. Contudo, estas tecnologias não se limitam às *smart homes*, podendo ser aplicadas por exemplo a hospitais, transportes públicos e fábricas.

2.4 Computação Sensível ao Contexto

A meta dos sistemas de Computação Ubíqua é ser minimamente intrusivos, o que demanda um conhecimento de seu contexto de execução. Ou seja, tais sistemas devem ter conhecimento do estado dos usuários e do meio, e devem modificar seu comportamento baseado nestas informações. O contexto de um usuário pode ser bastante rico, constituído de atributos como sua localização física, estado fisiológico (e.g., temperatura corporal, frequência cardíaca), estado emocional (e.g., irritado, distraído, calmo), histórico pessoal, padrões de comportamento diário, dentre outros. Se um(a) assistente estivesse em posse dessas informações, esta pessoa poderia tomar decisões de maneira pró-ativa, antecipando as necessidades do(a) usuário(a) [45].

A sensibilidade ao contexto é especialmente importante na Computação Ubíqua, embora possa ser utilizada por qualquer outra aplicação computacional. Isto ocorre pois as aplicação ubíquas não levam em consideração apenas os requisitos funcionais (i.e., domínio do problema) que devem prover, mas também o contexto em que se encontram os usuários e, em alguns casos, a infraestrutura de hardware e software [41].

As aplicações sensíveis ao contexto são integradas com o mundo físico e respondem a estímulos do ambiente obtidos através de sensores. Essa é uma caracterização fundamental para sistemas ubíquos/pervasivos. Considere o seguinte cenário: 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, como desligar o fogão, ou acordar o idoso com uma mensagem de aviso, ou contactar

um familiar. Portanto, 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.

Segundo a definição de Dey e Abowd [18], apresentada no Capítulo 1, o contexto é qualquer informação relevante usada para caracterizar a situação de entidades, especificamente: lugares, pessoas e coisas. Em um AmbI, "Lugares" são os cômodos, os andares de uma edificação, ou espaços em geral que possibilitam a localização de outras entidades; "Pessoas" são os indivíduos que povoam o ambiente e interagem com o mesmo; e "Coisas" são representações virtuais de objetos físicos ou componentes de software.

Em comparação com outras definições existentes na literatura [12, 38, 46, 47], Dey e Abowd propuseram uma das mais genéricas [32], considerando que ela deixa a cargo do projetista definir o que é relevante do contexto para atender aos requisitos da sua aplicação. Desta maneira, o contexto pode ser o percentual de uso de uma CPU, a latência de uma rede, os batimentos cardíacos de um paciente ou a localização de uma pessoa.

2.5 Agentes de Recurso

Um *recurso* é uma entidade de hardware ou software que expõe serviços passíveis de serem utilizados por outras entidades e aplicações presentes no ambiente inteligente. Assim, sensores, atuadores e dispositivos inteligentes (e.g. fogão, geladeira, ar-condicionado) se inserem nesta definição. São exemplos de recursos:

- Um sensor de temperatura, que expõe um serviço para obter a temperatura corrente do cômodo onde se encontra
- Um fogão, que expõe a informação de forno ligado ou desligado, bocas acesas, temperatura do forno, dentre outras, além de expor operações para acender as bocas, mudar a temperatura, e outras
- Um módulo de software, que agrega dados de diversos dispositivos para expor informações de consumo de energia do ambiente

A partir da definição de recursos, desenvolveu-se o conceito de *Agente de Recurso* (AR) no *framework*. Os ARs tem por objetivo encapsular os detalhes de implementação

e de comunicação com os recursos, e expor a interface de suas operações de forma padronizada. Assim, aplicações ou mesmo outros ARs podem acessar de maneira uniforme informações de um recurso além de executar suas operações. Isto ocorre através da comunicação síncrona ou assíncrona (ver Seção 2.6). Detalhes de baixo nível do recurso encapsulado são conhecidos apenas pelo AR, diminuindo significativamente a complexidade de integração de um recurso no sistema. Por exemplo, os algoritmos e estruturas de dados utilizados na coleta de dados de um sensor de temperatura seriam conhecidos apenas pela implementação de seu AR, ao passo que as entidades clientes somente precisam conhecer a interface deste sensor. Esta **separação de interesses** é equivalente ao conceito de encapsulamento do paradigma de Orientação à Objetos.

A Figura 2.1 apresenta uma arquitetura em camadas para o *framework*. A organização em camadas facilita a visualização dos diferentes componentes da arquitetura, sendo muito comum sua utilização em propostas de *middewares*, sistemas operacionais, e outras propostas que aumentam o nível de abstração de algo mais elementar.

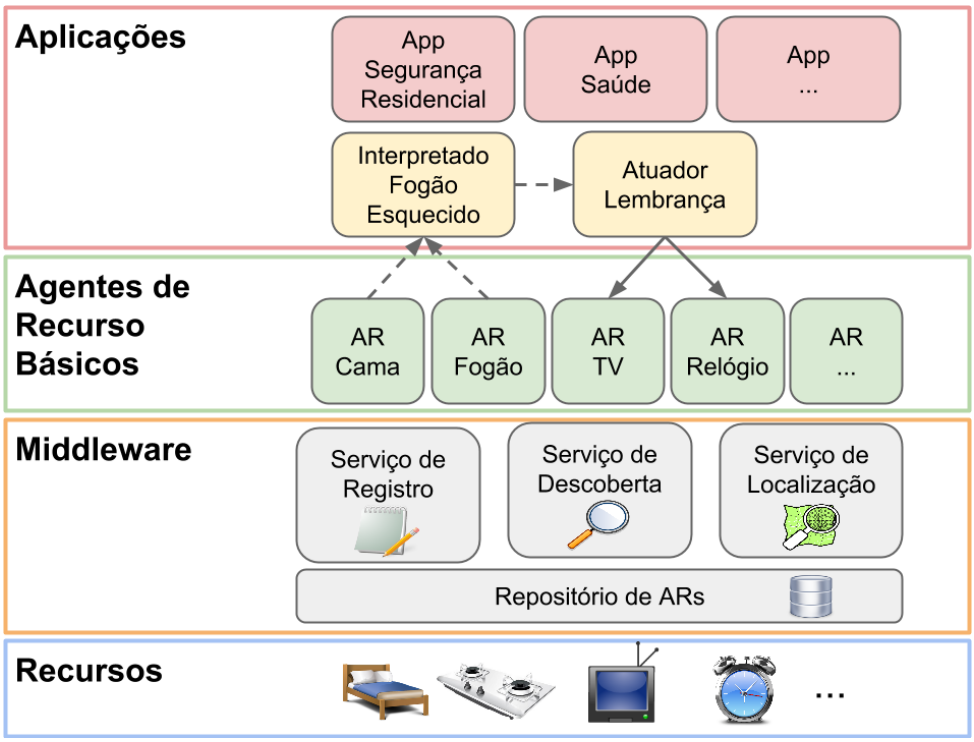


Figura 2.1: Arquitetura em Camadas do *Framework*

Na camada Recursos encontram-se os dispositivos físicos, ou módulos de software, que efetivamente interagem com os usuários (e.g., cama, fogão, TV). Na camada *Middleware* encontram-se os serviços que definem o Suporte ao Gerenciamento de Recursos (SGAR) do *framework*, que será apresentado na Seção 2.7. Na camada Agentes de Recurso Básicos 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. Na camada Aplicações encontram-se aplicações desenvolvidas por terceiros que usufruem de maneira segura das abstrações providas pela camada intermediária, assim como dos serviços. Estão também na camada Aplicações os Interpretadores de Contexto (e.g., Interpretador Fogão Esquecido) e agentes atuadores (e.g., Atuador Lembrança). A partir da implementação de referência, algumas aplicações foram construídas, o que é apresentado no Capítulo 4.

A estrutura geral do AR, como ilustrada na Figura 2.2, é composta por nome único ("tvFamília"), hierarquia de tipos ("\Visual\TV"), localização corrente ("Sala"), classe (classe TV) com variáveis e métodos, e uma lista de interessados em informações de contexto. A hierarquia de tipos, que é composta por uma sequência de classes, foi inspirada na ontologia descrita em [40] para classificar entidades (como os ARs). Esta hierarquia foi organizada para possibilitar a consulta e instanciação de ARs a partir de propriedades associadas a uma classe específica. Na Seção 2.7 é apresentado o SGAR que se beneficia desta estrutura do AR.

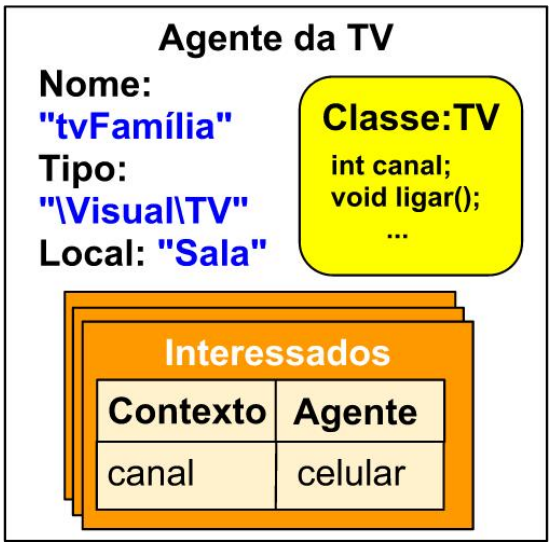


Figura 2.2: Estrutura Interna do AR

O conceito de AR possibilita a padronização dos componentes de interação do ambiente, e dessa forma, resolve o desafio (i) (ver Seção 1.2) relacionado à heterogeneidade dos dispositivos, como abordado em [33]. Sua importância para a interpretação do contexto está no fato de que as regras de contexto podem abstrair a maneira como foi coletada a informação, o que permite padronizar sua construção, além de facilitar a prototipagem de regras visto que a fonte do contexto é desconhecida. O Capítulo 3 detalha estas propriedades do *framework*.

2.5.1 Variáveis de Contexto e Operações

Um AR possui como interfaces para o AmbI 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 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.

O acesso às VCs e às OPs é feito utilizando os mecanismos padrões de comunicação síncrona e assíncrona. A Figura 2.3 representa o acesso às VCs e OPs de diferentes ARs. Na figura, a seta tracejada representa a subscrição à VCs (setas "Em uso" e "Ligado"), a seta comum representa o acesso às OPs dos ARs (setas "Liga", "Dispara alarme" e "Mostra mensagem"), cujos mecanismos de comunicação serão melhor descritos na Seção 2.6. Um ambiente em execução possui diversos ARs se comunicando concomitantemente. Uma VC pode notificar diversos ARs quando seu contexto muda, e os ARs podem estar subscritos a diferentes VCs de diferentes outros ARs, gerando uma malha de comunicação.

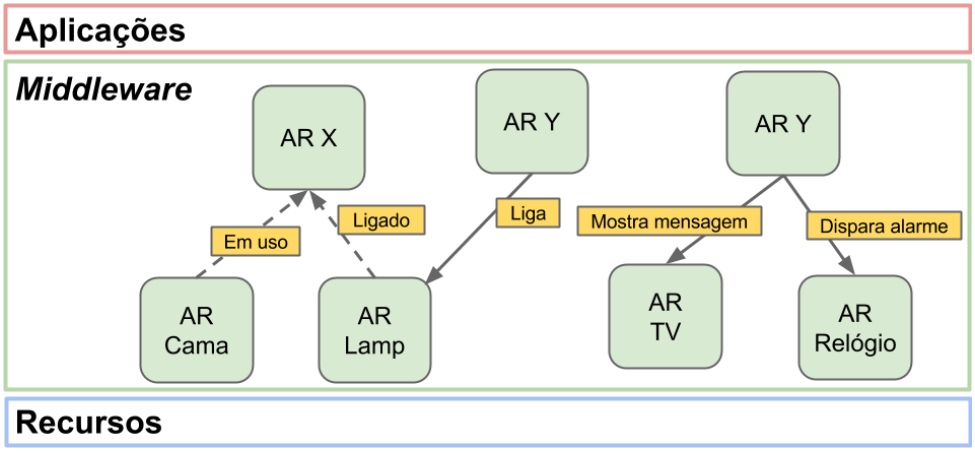


Figura 2.3: Acesso às VCs e OPs

O processo dinâmico de interação entre ARs é melhor apresentado na Figura 2.4. Em um momento inicial, apresentado na Figura 2.4.a, um AR arbitrário, chamado "AR X" está subscrito no "AR Lamp" que atua como um embrulho para a lâmpada real (padrão

wrapper). Em um segundo momento, apresentado na Figura 2.4.b, o "AR Y" invoca a operação "Liga" de lâmpada exposta pelo seu AR, e isso faz com que a lâmpada seja propriamente ligada e os subscritos à VC "Ligado" sejam notificados, no caso, o "AR X".

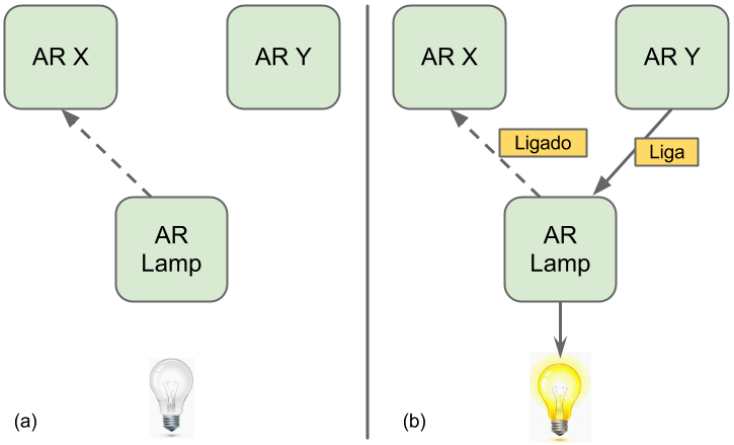


Figura 2.4: (a) Subscrição (b) Cenário de atuação

O acesso às VCs, assim como às OPs, ocorre de maneira transparente para as aplicações e outros ARs. Uma vez obtida a referência para o AR de interesse e com o conhecimento de sua interface, um *stub* é criado para representá-lo localmente. As chamadas às VCs e OPs do AR são efetuadas através de chamadas à métodos do objeto criado, que em seguida são transformadas em mensagens TCP e enviadas pela rede.

O mecanismo de publicação de eventos é utilizado na construção de regras de contexto, possibilitando que uma regra tenha ciência da atualização de uma VC em tempo real, e seja capaz de atuar no ambiente através das OPs. Isto será melhor abordado no Capítulo 3.

2.6 Paradigmas de Comunicação

De modo a atender requisitos típicos de ambientes distribuídos, a proposta utiliza dois mecanismos básicos de comunicação: a comunicação síncrona e a comunicação assíncrona, apresentados a seguir.

2.6.1 Comunicação Síncrona

Na comunicação síncrona uma entidade envia uma mensagem e aguarda uma resposta (corresponde ao protocolo *request/reply*), a fim de obter o contexto de um recurso ou de utilizar um serviço. O principal benefício deste tipo de comunicação é a interação direta

entre as entidades de um AmbI, permitindo que a qualquer momento uma VC possa ser interrogada, para saber seu estado, ou uma OP possa ser invocada.

Na maioria dos *frameworks* atuais esta comunicação ocorre em nível mais abstrato, através da utilização de chamadas de métodos remotos (*Remote Procedure Call* – RPC) [7], como por exemplo o Java RMI (*Remote Method Invocation*) [36], o CORBA [35], a tecnologia de *Web Services* [50] utilizada por diversas plataformas, dentre outras. As chamadas síncronas possibilitam a obtenção do estado de um recurso de maneira direta.

A funcionalidade provida pelo RPC ocorre a partir da definição de uma interface da entidade, que declara os serviços desejados, e é conhecida pela entidade chamadora. Esta interface possuirá duas implementações, sendo a primeira a entidade que executa propriamente a regra de negócio envolvida, e a segunda uma representação, que fica localizada remotamente, e encaminha as chamadas para a primeira, agindo assim como um *proxy*. A implementação da segunda entidade frequentemente é gerada pelo *middleware* de comunicação automaticamente. Assim sendo, o RPC possibilita que a chamada remota seja realizada de maneira transparente para a entidade chamadora, que se comporta como se estivesse efetuando uma chamada de método local, quando na verdade a ocorre uma chamada através da rede.

A segunda entidade, acima descrita, que oferece uma representação da entidade real, é conhecida como *stub*. Como representado na Figura 2.5, quando ocorre uma chamada ao método local do *stub* (Passo 1) esta invocação é, junto com seus parâmetros, serializada (*marshalling*) em um vetor binário ou em um formato de dados semiestruturados (como o XML [51] ou o JSON [29]). A seguir, estes dados são repassados para o suporte do *middleware* para que seja então enviada através da rede (Passo 2). Ao chegar no destinatário a mensagem é desserializada (*unmarshalling*) também pelo suporte do *middleware*, que efetua a seguir a operação na entidade real passando os parâmetros (Passo 3). Ao fim da execução, o *middleware* serializa o valor de retorno, e envia de volta ao *stub* através da rede (Passo 4), que é encarregado de entregá-lo à aplicação. A Figura 2.5 representa todo o processo de encaminhamento da chamada e da resposta.

Considere o exemplo de uma lâmpada inteligente cujo AR possui a operação "*turnOn*", que permite acendê-la. Esta operação pode ser invocada remotamente através de seu *stub* específico. O *middleware* de suporte cuida do encaminhamento da mensagem com seus parâmetros para o entidade representativa da lâmpada, e também efetua a entrega da resposta ao chamador.

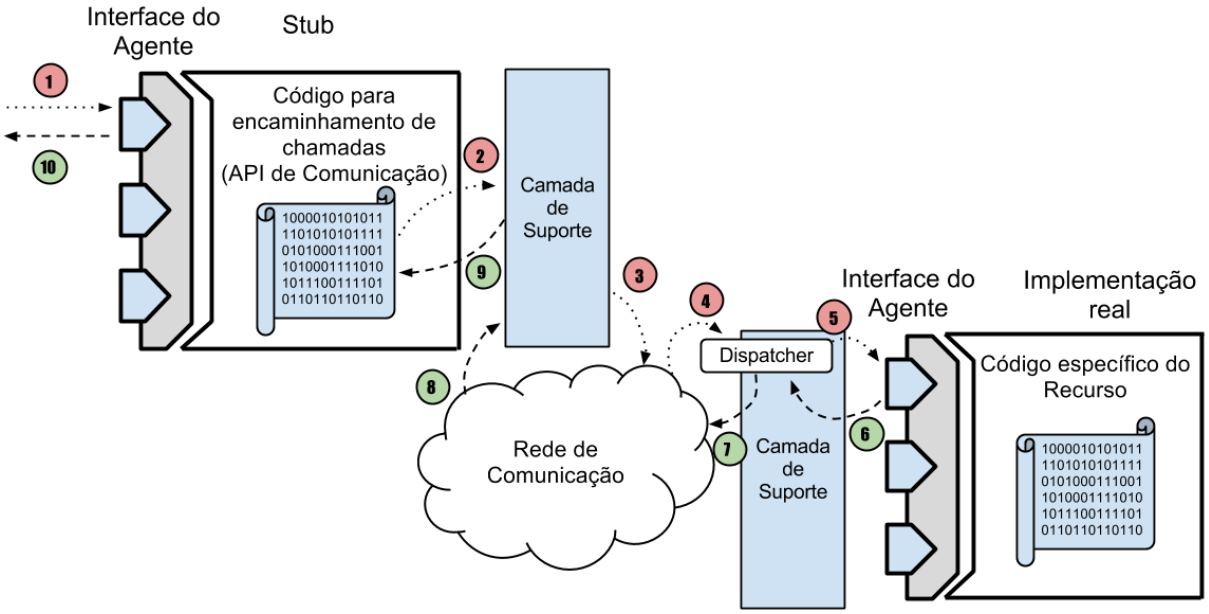


Figura 2.5: Chamada Remota

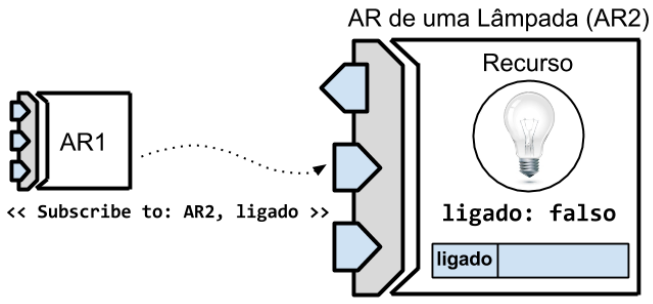
2.6.2 Comunicação Assíncrona

Enquanto na comunicação síncrona a entidade requisitante espera pela resposta da sua chamada, como apresentado na seção anterior, na comunicação assíncrona uma entidade publica seu interesse em uma informação e, eventualmente, será notificada com o valor. A entidade interessada não tem garantias de envio imediato da resposta, e não necessariamente é bloqueada na espera. O benefício da utilização deste padrão no AmbI é a comunicação indireta, que permite que ARs interessados sejam sempre notificados de mudanças nas VCs de interesse.

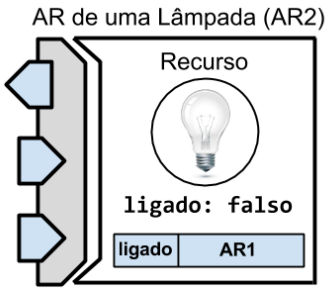
Uma solução para a implementação da comunicação assíncrona em um sistema distribuído é o padrão publica-subscreve (*publish-subsribe*), onde as entidades comunicantes são separadas em publicadoras de eventos e subscritoras de eventos (também chamadas de interessadas, ou *stakeholders*). O publicador recebe pedidos de subscrição de um ou mais interessados em um evento. Quando o evento ocorre, o publicador o propaga para todos os interessados.

A Figura 2.6 apresenta um roteiro de funcionamento do padrão *publish-subscribe*. Considere o agente interessado (AR1), o AR de uma lâmpada (AR2) que possui a informação de interesse "ligado", e um terceiro agente que também interage com a lâmpada (AR3). No início, o AR1 se subscreve ao AR2 (Passo (a)), e, internamente, o AR2 guarda a referência para o interessado (Passo (b)). Passado algum tempo, o AR3 faz com que a lâmpada seja ligada, no caso ao chamar "*turnOn*" (Passo (c)), e modificando assim o

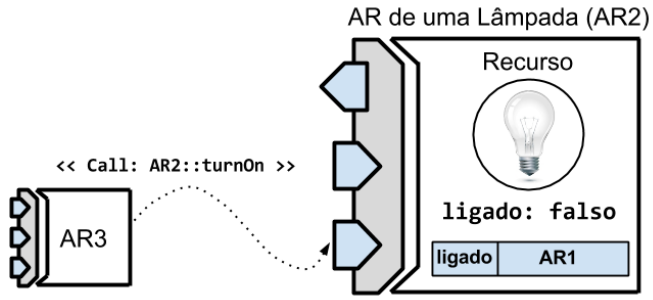
contexto "ligado" (Passo (d)). A seguir, AR da lâmpada notifica todos os interessados da mudança no contexto (Passo (e)).



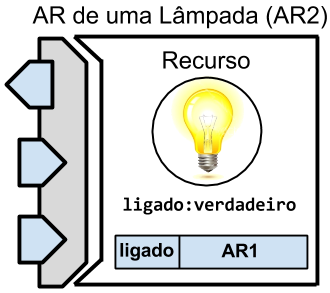
(a) Subscrição do AR1 ao AR2



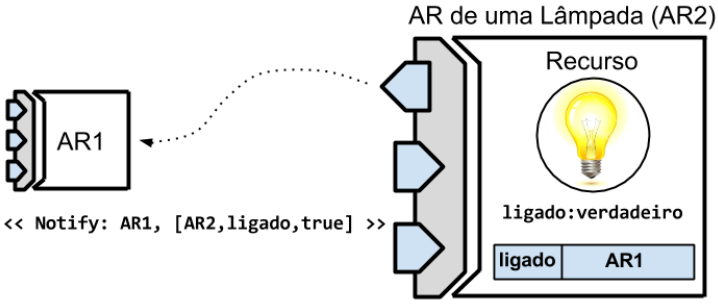
(b) AR2 armazena subscrição do AR1



(c) AR3 invoca operação do AR2



(d) Estado do AR2 modificado



(e) Notificação do AR1

Figura 2.6: Processo de subscrição e notificação de ARs

A mensagem de notificação do evento propagado deve conter informações de quem a enviou, do evento desencadeado e do novo estado da informação de contexto.

Internamente, a comunicação com o AR funciona como apresentado na Figura 2.7. Neste exemplo, dois ARs (a TV e o Alarme) se subscrevem ao AR do fogão (Etapa 1). O AR do fogão então mantém referências para os dois interessados (ou *stakeholders*), e esta lista contém tanto o nome do AR na rede quanto a VC de interesse. Eventualmente, um serviço deste AR será invocado (Etapa 2), que no caso é o "turnOn (burner)" (que liga uma boca do fogão). Neste momento, a chamada é interceptada (Etapa 3) por meio de

técnicas de Programação Orientada à Aspectos (POA), que ativa o código de seleção e notificação de interessados. Na seleção, apenas a TV é escolhida, por estar subscrita na VC "turnOn (burner)" (Etapa 4) e logo em seguida é notificada (Etapa 5).

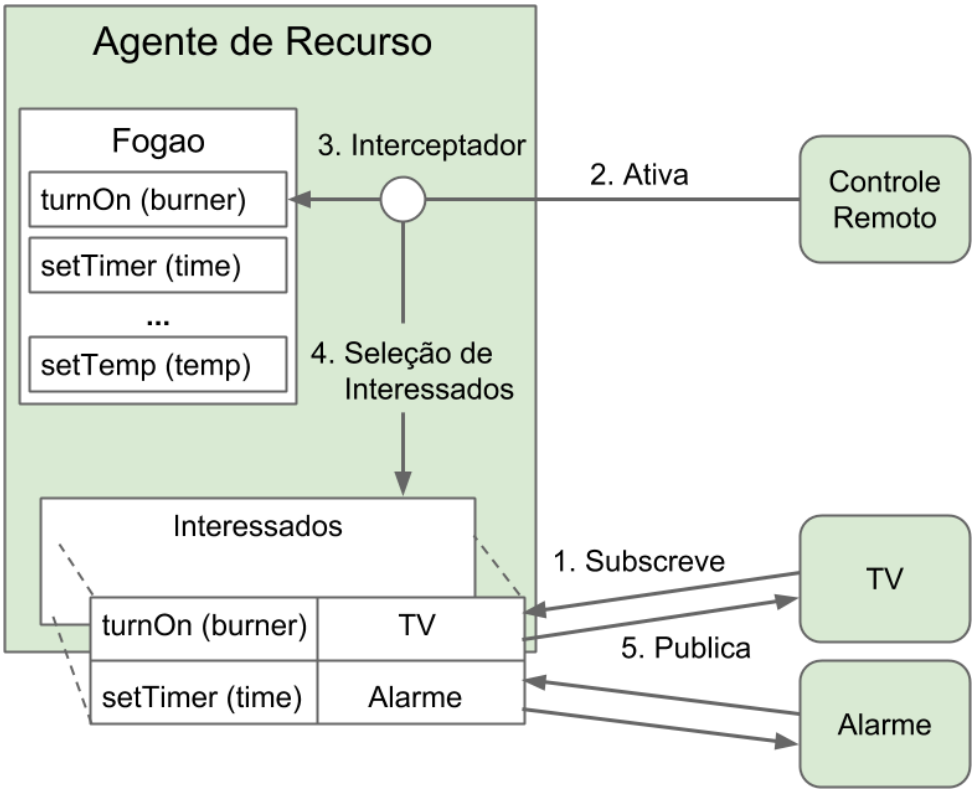


Figura 2.7: Agente de Recurso

2.7 Suporte ao Gerenciamento de Recursos

O Suporte ao Gerenciamento de Recursos (SGAR) é um conjunto de serviços do *framework* responsáveis pelo gerenciamento dos ARs no AmbI, cujas principais funções são o registro dos ARs, busca e manutenção das referências. Na arquitetura em camadas da Figura 2.1, apresentada anteriormente, o SGAR é compreendido pelos serviços localizados na camada intermediária e o Repositório de Recursos (RR).

Três componentes, ou serviços básicos, compõem o SGAR, como evidenciado na Figura 2.8: o Serviço de Registro de Recursos (Subseção 2.7.3), o Serviço de Descoberta de Recursos (Subseção 2.7.4) e o Serviço de Localização de Recursos (Subseção 2.7.5). Estes serviços estão relacionados ao RR (Subseção 2.7.2). As principais operações realizadas no SGAR são as representadas na figura, i.e. registro/remoção dos ARs, busca e localização de ARs, todas desempenhadas a partir de informações presentes no RR.

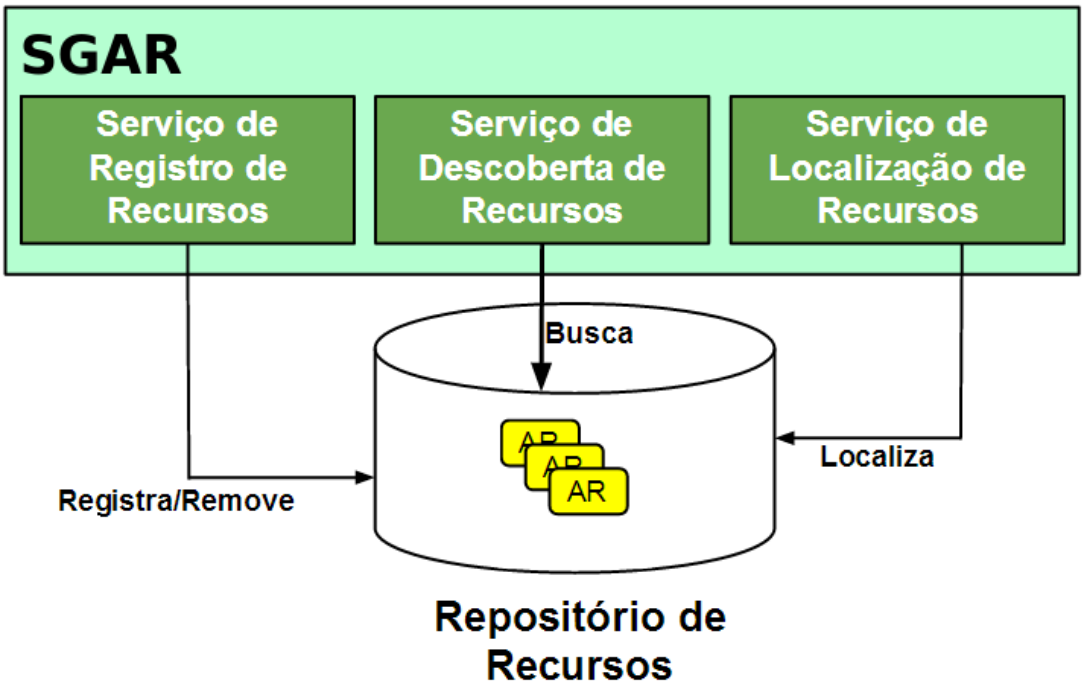


Figura 2.8: Arquitetura do SGAR

2.7.1 Organização Típica do Ambiente

Os componentes do AmbI são organizados de maneira essencialmente distribuída e se comunicam diretamente uns com os outros, como apresenta a Figura 2.9. Os serviços básicos presentes no SGAR executam em máquinas mais robustas, garantindo assim requisitos de qualidade de serviço e segurança a falhas.

A principal forma de comunicação entre os componentes é através de redes sem fio. Isto possibilita uma maior mobilidade dos dispositivos e flexibilidade da rede, aspectos que permitem tornar transparente a interação entre o usuário e o AmbI. A implementação de referência utiliza particularmente o Wi-Fi, que é suportado por uma ampla gama de dispositivos além de prover requisitos de segurança. Contudo, não existem restrições para o uso de outras tecnologias, como Bluetooth, ou Zigbee.

2.7.2 Repositório de Recursos

O Repositório de Recursos desempenha o papel de repositório central na arquitetura como apresentado na Figura 2.10 e atua como um ponto inicial a partir do qual os ARs podem ser encontrados, incluindo os próprios serviços do SGAR.

Internamente, o RR possui o Diretório de Recursos e o Mapa (Figura 2.10), o qual contém representado o conjunto de espaços físicos do AmbI. O Diretório de Recursos

Figura 2.9: Organização das entidades do ambiente inteligente

contém os nomes das instâncias de AR existentes no ambiente, armazenados conforme seus respectivos tipos. No exemplo da Figura 2.10 há o tipo TV com a "tvFamília" e o tipo *tablet* com o "iPad" e o "Galaxy". Os dados consistem do nome, tipo, localização e a referência de acesso. A referência de acesso permite a interação entre instâncias de ARs diferentes, através do RPC (ver Seção 2.6.1).

Cada espaço do Mapa possui uma área e um nome. No caso de uma *smart home* os cômodos da casa representam estes espaços. Cada entrada do Mapa referencia o conjunto de ARs contidos no respectivo espaço. Os ARs representando dispositivos com mobilidade têm suas referências atualizadas dinamicamente.

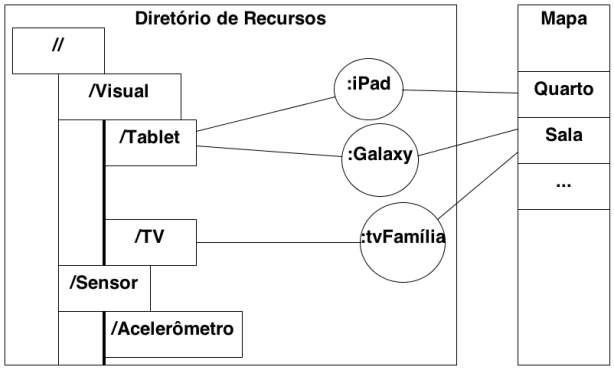


Figura 2.10: Repositório de Recursos

2.7.3 Serviço de Registro

Para possibilitar a descoberta/localização de ARs é necessário que um componente, de interface bem definida, permita o registro de um AR incluindo sua localização e características estáticas. Para este fim, o SGAR contém o Serviço de Registro de Recursos (SRR).

O SRR é responsável por incluir/excluir referências de ARs no RR, onde se encontram todos os recursos ativos no sistema, ou seja, os ARs que podem fornecer seus serviços para outras entidades e aplicações. Sempre que um AR novo for instanciado no sistema, este deve chamar o *Serviço de Registro de Recursos* (SRR), para que sua referência seja incluída no RR e consequentemente fique disponível (visível) para o AmbI.

Para adquirir uma referência ao SRR, o dispositivo deve enviar uma mensagem de *broadcast*, com um parâmetro informando que deseja a referência do SRR daquele ambiente. O SRR ao receber a mensagem, envia para o solicitante sua referência, que a partir desse ponto pode ser utilizada para invocar suas operações. Na Figura 2.11 uma TV envia uma mensagem de *broadcast*, que é recebida por todos os dispositivos ao alcance do sinal. O SRR, presente em um servidor no ambiente também recebe a mensagem, e responde à TV sua identificação. Os outros dispositivos que receberam a mensagem de *broadcast* descartam a mensagem recebida.

Figura 2.11: TV se registrando no Serviço de Registro de Recursos

2.7.4 Serviço de Descoberta

O Serviço de Descoberta de Recursos (SDR) através de suas interfaces de acesso, possibilita que sejam encontrados ARs que estejam devidamente registrados pelo SRR baseados em algum critério. Uma vez obtida a referência para um AR, é possível que uma aplicação, regra ou outro AR obter suas características e informações de contexto. O SDR atua junto do RR, já populado com as referências dos ARs, a fim de realizar a

busca.

A descoberta pode ser realizada através de vários tipos de consulta, que retornam como resultado referências para os ARs que satisfazem os critérios da busca. Uma busca pode ser realizada utilizando o nome de um AR, a identificação, ou ainda o tipo, que é caracterizado através da definição de uma ontologia mínima (como detalhado em [5]).

2.7.5 Serviço de Localização

O Serviço de Localização de Recursos (SLR) semelhantemente ao SDR busca por ARs, contudo, utilizando critérios relacionados à sua localização física. Assim é possível encontrar um AR de duas maneiras. Um AR pode ser encontrado baseado na localização (coordenadas) ou região (ou cômodo), onde poderiam ser feitas consultas como "qual é o AR presente na posição (x,y)?" ou "qual é a TV presente na sala?". A segunda maneira é localizar o AR com posição referente a outro AR, como em "qual a TV mais próxima de determinada pessoa?", ou seja, "dado um AR, qual o AR de determinado tipo mais próximo dele?".

O SLR está associado à uma estrutura de dados que representa o mapa do ambiente, mantida pelo RR. Este mapa é capaz de localizar um recurso pontualmente (em coordenadas cartesianas), assim como em um ambiente (e.g., quarto, sala). Um ambiente pode também estar localizado dentro de outro ambiente, criando uma recursividade e aumentando o poder de expressividade do serviço de localização. Por exemplo, uma sala pode estar dentro de um apartamento, que está em um andar do edifício, etc. Como será visto na Seção 2.8, uma interface gráfica utiliza esta estrutura de mapa para apresentar em alto nível para os usuários o ambiente completo.

2.8 Interface de Prototipagem e Simulação

Em comparação ao avanço no desenvolvimento de aplicações *mobile*, as aplicações ubíquas ainda são escassas no mercado. Podemos citar como causas desse efeito o alto custo de desenvolvimento demandado por elas, devido a questões como a falta de ferramentas adequadas para a criação e integração dessas aplicações, e a dificuldade em depurá-las [53]. Além disso, um ambiente de testes contendo todos os dispositivos e a infraestrutura necessária para realizá-los pode ser inviável financeiramente, ao passo que um ambiente construído em pequena escala pode não ser suficiente para testar os diversos cenários possíveis em um ambiente inteligente.

Visando atender a este desafio, é proposta em Barreto (2013) [3] uma ferramenta para o gerenciamento de aplicações ubíquas/pervasivas e suporte à construção de protótipos, chamada Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP). A IPGAP fornece ao desenvolvedor um ambiente para controle e depuração de aplicações de maneira rápida e com baixo custo. Além de ser também conveniente para os usuários finais, que podem usufruir de uma interface amigável de controle completo do AmbI.

A Figura 2.12 ilustra a IPGAP executando em um *tablet* à esquerda, onde o ambiente representado é uma residência povoada com diversos recursos. O mapa visual é composto a partir da estrutura de dados em mapa contida no RR, dos recursos presentes neste ambiente, que são posicionados conforme suas localizações, e também são representados os usuários dos sistema, no caso, um único. Desta maneira, esta figura evidencia as três principais entidades do *framework*: lugares, pessoas e coisas.

No lado direito da Figura 2.12 são apresentados três simuladores em funcionamento em *smartphones* para: uma lâmpada, uma TV e um termômetro. Como os simuladores são implementados em ARs, suas interfaces são potencialmente as mesmas que as de ARs de recursos reais, desta maneira o AmbI os encara como recursos reais de maneira transparente. As representações da IPGAP estão subscritas nos ARs dos simuladores (ação essa que é executada de maneira simples na própria IPGAP), e quando estes mudam os seus estados, a informação de contexto que mudou é apresentada na tela. No exemplo da Figura 2.12, a lâmpada do quarto é acendida, a TV muda de canal para o "8" e o termômetro passa a marcar 17º C.

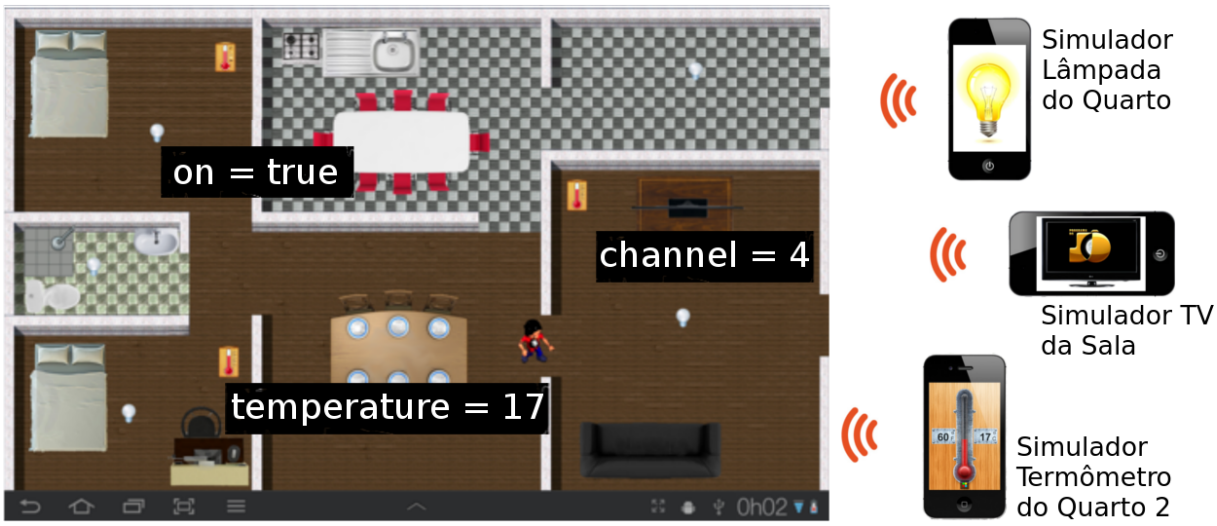


Figura 2.12: IPGAP e Simuladores

2.9 Segurança

Requisitos de segurança, embora não sejam o foco deste *framework*, são inerentes ao problema abordado. Dentre os principais problemas de segurança estão as aplicações maliciosas, os recursos invasores e os usuários com acesso irrestrito.

Aplicações maliciosas poderiam ser construídas para operar no AmbI de maneira indesejada atuando, por exemplo, como *trojans*. Uma solução que é comumente aplicada para resolver este tipo de problema é a atualmente usada em lojas de aplicativos para *smartphones* (e.g., Google Play, App Store). Em primeira instância, as lojas testam as aplicações, procurando por códigos maliciosos (i.e., *malware*), e em uma segunda etapa, os usuários podem observar que serviços a aplicação usa e permitir ou bloquear a instalação. Além disso, é possível também um usuário reportar um problema ocorrido.

Recursos invasores são ARs que se registram no sistema (estando localizados fisicamente perto do AmbI) para ter acesso tanto às informações pessoais dos usuários, como às funcionalidades de outros recursos e aplicações. Para a prevenção contra invasores, a rede interna já oferece um nível satisfatório de proteção. Na implementação de referência, os ARs se comunicam por meio de uma rede Wi-Fi segura, onde os roteadores já possuem mecanismos de criptografia, como o WPA2 (IEEE 802.11i-2004), que garantem segurança na troca de mensagens.

O problema de restrição de acesso de usuários é resolvido com um sistema de permissões, comumente usado em sistemas operacionais. São definidos grupos de usuários (e.g., administrador, usuário comum, convidado, criança), como em [19], e usuários em grupos com menos permissões não podem desempenhar quaisquer tarefas no AmbI. Por exemplo, um usuário do tipo "criança" não poderia ligar o forno de micro-ondas, um usuário do tipo "convidado" não poderia destrancar a porta com sistema biométrico. Esta configuração pode ser realizada em alto nível na IPGAP.

2.10 Conclusões do Capítulo

Este capítulo apresentou inicialmente a definição de conceitos básicos que permeiam esta dissertação e que orientam o seu desenvolvimento. Em seguida, foi apresentado o *framework* desenvolvido em conjunto com este trabalho que trata de questões às quais esta proposta está intimamente relacionada. Dentre as características do *framework* abordadas foram apresentados a abstração dos Agentes de Recurso, assim como as Variáveis de Contexto e as Operações, e os serviços do SGAR, que implementam as funcionalidades mais básicas do sistema distribuído. Por fim, foi apresentada a IPGAP, que permite a desenvolvedores criarem protótipos de novas aplicações e visualizar seu funcionamento, além de possibilitar a configuração do ambiente por desenvolvedores e usuários finais.

No capítulo seguinte serão apresentados outras características do *framework* que complementam esta proposta e favorecem a construção de aplicações sensíveis ao contexto para AmbI.

Capítulo 3

Proposta

Este capítulo apresenta a proposta para a interpretação de contexto em ambientes inteligentes em detalhes. Na Seção 3.1 é apresentada uma observação geral sobre a proposta. Na Seção 3.2 é apresentada a estrutura da regra de contexto e sua motivação, seguida do seu ciclo de vida na Seção 3.3. Na Seção 3.4 é descrita a criação de um Interpretador de Contexto em nível de programação. A Seção 3.5 apresenta o funcionamento da temporização de regras de contexto. A Seção 3.6 oferece mais detalhes sobre o funcionamento do *parser* e na Seção 3.7 é apresentada a implementação do processo completo de criação da regra e de seu funcionamento no *SmartAndroid*. Por fim, na Seção 3.8 é apresentada a solução em nível de usuários finais para a composição de regras de contexto.

3.1 Introdução

Este trabalho complementa outros dois trabalhos anteriores para a proposição de um *framework* conceitual que contempla a construção de aplicações sensíveis ao contexto. Inicialmente este *framework* foi proposto por Mareli (2013) [33], onde há um maior foco no modelo de componentes distribuídos, apresentado nas Seções 2.6, 2.5 e 2.7, e atende ao desafio (ii) relacionado à heterogeneidade das fontes de contexto. Em Barreto (2013) [3] é proposta a IPGAP, que se foca na prototipagem e no gerenciamento de aplicações, atendendo ao desafio (iii), relacionado à disponibilidade dos recursos (ver desafios na Seção 1.2).

A proposta deste trabalho é prover uma abstração para a interpretação de contexto, favorecendo a criação de regras de contexto para o AmbI, e atendendo assim ao desafio (ii), relacionado à variedade e quantidade de informações de contexto. Desta maneira, ao utilizar o *framework*, os desenvolvedores serão capazes de abstrair questões recorrentes

no desenvolvimento de aplicações sensíveis ao contexto (os desafios (i), (ii) e (iii)) para se focar em questões diretamente relacionadas ao domínio de sua aplicação. Do ponto de vista do usuário, este trabalho propõe adicionalmente uma interface para a composição de regras de contexto através da IPGAP. Isto possibilita tanto a prototipagem de regras quanto a implantação de regras no ambiente, por desenvolvedores e por usuários sem experiência técnica, o que atende também ao desafio (iii).

Afim de avaliar os conceitos propostos, foi desenvolvido como implementação de referência o projeto *SmartAndroid*[1] que inclui a solução proposta neste trabalho. O *SmartAndroid* compreende a construção de um *middleware* que implementa as características presentes no *framework* conceitual e a IPGAP, provendo assim condições para o desenvolvimento de aplicações para o AmbI.

O middleware desenvolvido possui uma API que possibilita a utilização dos principais serviços do *framework* apresentados, incluindo a definição de novos ARs pelo desenvolvedor, o acesso ao serviços do SGAR (i.e., SRR, SDR e SLR), a criação de regras de contexto, dentre outros. Além do *middleware*, o *SmartAndroid* inclui a implementação da IPGAP, que auxilia na prototipagem de aplicações ubíquas.

Dentre as tecnologias utilizadas na implementação de referência, foi utilizado para a comunicação entre diferentes dispositivos (i.e., *smartphones*, *tablets*) uma rede Wi-Fi, com WPA2 para a segurança na comunicação. O sistema foi construído sobre a plataforma Android e esta escolha se sustenta na acessibilidade à esta plataforma (que inclusive possui seu código fonte aberto) e na variedade de bibliotecas, o que facilita a implementação de sistemas embarcados. Contudo, é importante ressaltar que não há impedimentos para a implementação dos conceitos do *framework* em outras plataformas distribuídas atuais.

3.2 Regras de Contexto

A motivação de responder à estímulos do ambiente gera a necessidade de se construir regras de contexto. Estas regras têm por objetivo atender às necessidades de usuários e aplicações para atuar no ambiente de forma autônoma, alterando seu estado. O *framework* possibilita que sejam construídas novas aplicações que definem regras para atuar no AmbI, estendendo o alcance de aplicações tradicionais.

O mecanismo de publicação de eventos é a base para a criação de aplicações sensíveis ao contexto. Uma regra de contexto é composta por duas partes. A primeira é a expressão

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

(ou condição), desempenhada no *framework* pelos Interpretadores de Contexto (IC), e a segunda é a atuação, desempenhada pelos agentes atuadores.

A fim de facilitar a criação de regras de contexto, os ICs possibilitam a definição da regra de uma forma padrão através do *middleware* o que facilita também a gerência das regras pelos usuários. O suporte à interpretação de contexto possibilita uma separação de interesses, onde os desenvolvedores abstraem a implementação das regras e focam na lógica da aplicação.

A função do IC é agregar informações contextuais provenientes de diferentes fontes, considerando também a passagem de tempo, e relacioná-las segundo uma expressão lógica em específico. Assim, para que o IC subscreva às VCs, e que outros agentes possam se subscrever ao IC, este estende a classe base dos AR.

A avaliação da expressão é desempenhada pelo próprio IC conforme o contexto muda. A atuação ocorre quando um agente atuador é subscrito a um IC (que implementa a condição), recebendo assim a notificação de que a regra foi avaliada como verdadeira.

A Figura 3.1 apresenta o roteiro do cenário "Fogão Esquecido" utilizando o suporte do *middleware*, que representa a execução da seguinte regra simples:

1. Uma pessoa (que mora sozinha) ligou o fogão
2. A pessoa foi para o quarto e se deitou na cama
3. Passaram-se 15 min e a regra mudou seu estado para verdadeira
4. O alarme do relógio do quarto é disparado
5. É exibido na TV a mensagem "O fogão foi esquecido ligado!"

À esquerda da Figura 3.1 pode-se observar a ligação entre o IC ("Interpretador Fogão Esquecido") e VCs de dois diferentes ARs (i.e., "Em uso" do Agente Cama e "Ligado" do Agente Fogão); pode-se ver ainda a ligação entre o IC e um atuador ("Atuador Lembrete"), e a utilização de um temporizador internamente no IC; e, à direita da figura, é representada a ligação do atuador com duas diferentes OPs (i.e., "Dispara alarme" do Agente Relógio e "Mostra mensagem" do Agente TV).

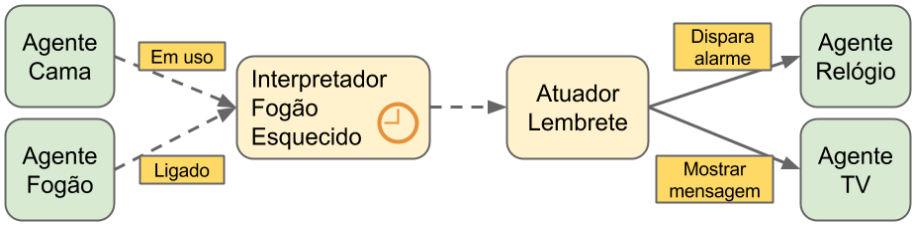


Figura 3.1: Interpretação de Contexto

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 o IC não só à um atuador que esteja criando, mas também a outros que já existem (e.g., no cenário do "Fogão Esquecido" um atuador "Avisar família" poderia ser também subscrito no IC).

Outra propriedade dos ICs é a capacidade de se ligar uns aos outros, dado que o IC é um AR e seu estado válido/inválido é exposto como uma VC. O IC agrega informações de contexto e expõe uma VC com uma informação em nível mais alto, possibilitando que esta informação seja usada não só pelo pelo atuador mas também em outras regras, diminuindo o tráfego de mensagens na rede, a complexidade na construção da regra e aumentando a consistência entre regras, além de facilitar a evolução das regras de contexto, como descrito a seguir.

Tráfego de mensagens O tráfego de mensagens reduz pois se uma regra possui uma subexpressão que é a expressão de outra regra, e esta é composta por um número `N` de VCs, então o tráfego na rede irá aumentar para até um máximo de `N + 1` com a nova regra, e não um máximo de `2N`, uma vez que não é necessário enviar duas vezes as mesmas VCs.

Complexidade da regra A separação de interesses diminui a complexidade da regra por evitar definir repetidamente um conceito (e.g., pessoa dormindo). Se já existe um IC que identifica se uma pessoa está dormindo, por exemplo, basta incluir a VC provida pelo IC anteriormente definido na nova regra.

Consistência entre regras A consistência entre regras aqui diz respeito à utilização de um conceito de forma igual em regras diferentes. É ideal que diferentes regras que utilizem o conceito pessoa dormindo, por exemplo, sigam o mesmo padrão, o que garante uma uniformização no uso da plataforma e um melhor entendimento pelos usuários.

Manutenção das regras Este reaproveitamento de conceitos também facilita a evolução das regras (i.e., quando se deseja mudar a regra seja substituindo VCs, alterando a lógica, ou ambos), por possibilitar a mudança em um lugar apenas.

A Figura 3.2 é uma modificação da Figura 3.1 anteriormente apresentada. Nesta versão da figura, ao invés de o IC definir internamente o que é uma pessoa dormindo (no exemplo anterior o IC era simplesmente ligado ao agente da cama), ele se liga a outro

IC que implementa este conceito. Como pode-se ver na figura, esta ligação é realizada utilizando o mecanismo padrão de subscrição.

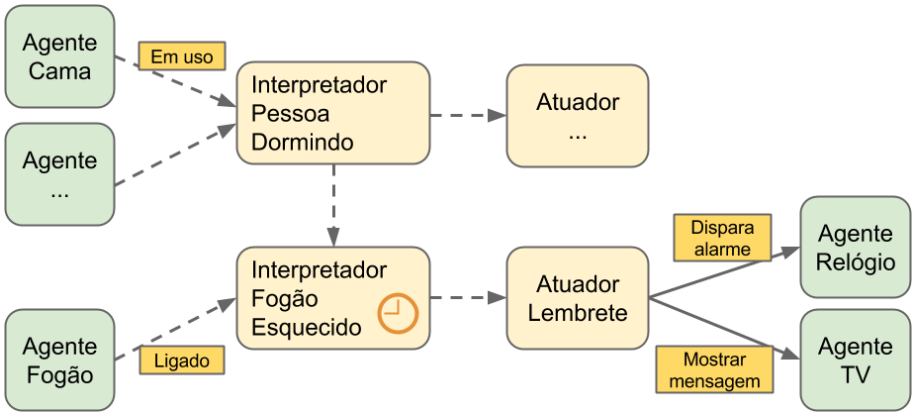


Figura 3.2: Ligação entre ICs

O *framework* é suficientemente flexível para possibilitar a interpretação de contexto a partir dos ICs, assim como de outras formas. Em alguns casos, é interessante a agregação de um motor de regras (e.g., Jess [25], Drools [20], CLIPS [14], outros motores proprietários), por exemplo, quando há muitas regras. Um motor de regras é uma ferramenta geralmente centralizada, que recebe informações de contexto de diversas fontes e processa regras previamente inseridas. É comum usar um algoritmo como o RETE [24], que processa regras construídas apenas com a cláusula "E" eficientemente, tais como a regra do cenário "Fogão Esquecido". Assim sendo, utilizando o *framework*, é também possível que sejam feitas as ligações entre as VCs e o motor de regras, se for de conveniente. Isto pode ser feito encapsulando o motor em um AR e registrando-o no AmbI.

Em geral, motores de regras são ferramentas mais complicadas de serem usadas, que visam uma economia de recursos de processamento e não de recursos de rede. Além disso, regras formadas exclusivamente com a cláusula "E" são menos expressivas do que as regras mais complexas, como as utilizadas na abordagem proposta, possibilitando a usuários e desenvolvedores expressarem suas necessidades em um conjunto menor de regras e com mais significado.

3.3 Ciclo de Vida dos Interpretadores de Contexto

Duas fases importantes na vida dos ICs são a criação e a execução. Eventualmente, enquanto estiver em execução, o IC vai alcançar um estado válido, ou seja, quando sua expressão for avaliada como verdadeira. A expressão (ou parte dela) será reavaliada sempre que uma VC mudar seu estado ou se um temporizador chegar ao final, o que pode

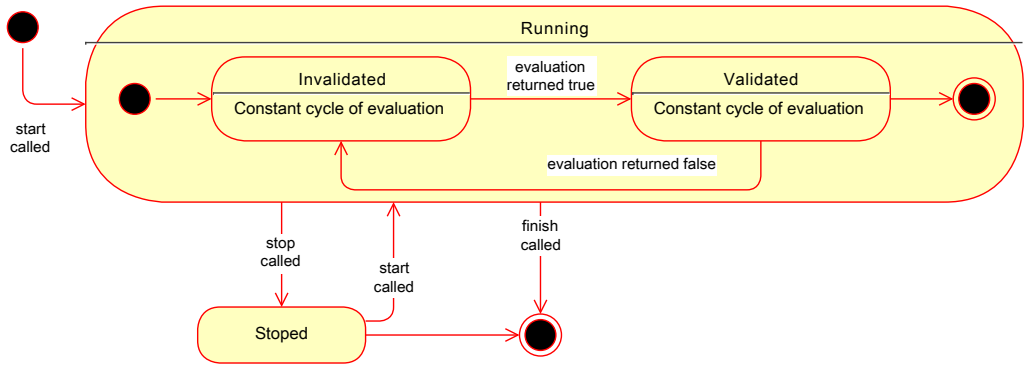


Figura 3.3: Diagrama de Estado: Ciclo de vida do IC

ocasionar a mudança de estado do IC de inválido para válido, ou vice-versa, como bem evidencia o diagrama de estado representado na Figura 3.3, que apresenta os estados de execução do IC. Ainda, por motivo de controle, um IC pode ser parado ou finalizado.

O Diagrama de Sequência da Figura 3.4 apresenta a rotina de criação de um novo IC. Inicialmente um objeto do tipo *ContextInterpreter* é instanciado (Etapa 1), que é a classe base do IC, e este é registrado junto ao SRR. Através de "*setExpression*" a expressão é enviada para o IC (Etapa 2), que ao recebê-la o IC efetua o "*parse*" da expressão, gerando uma representação em memória da regra (Etapa 2.1), i.e. a Estrutura Lógica.

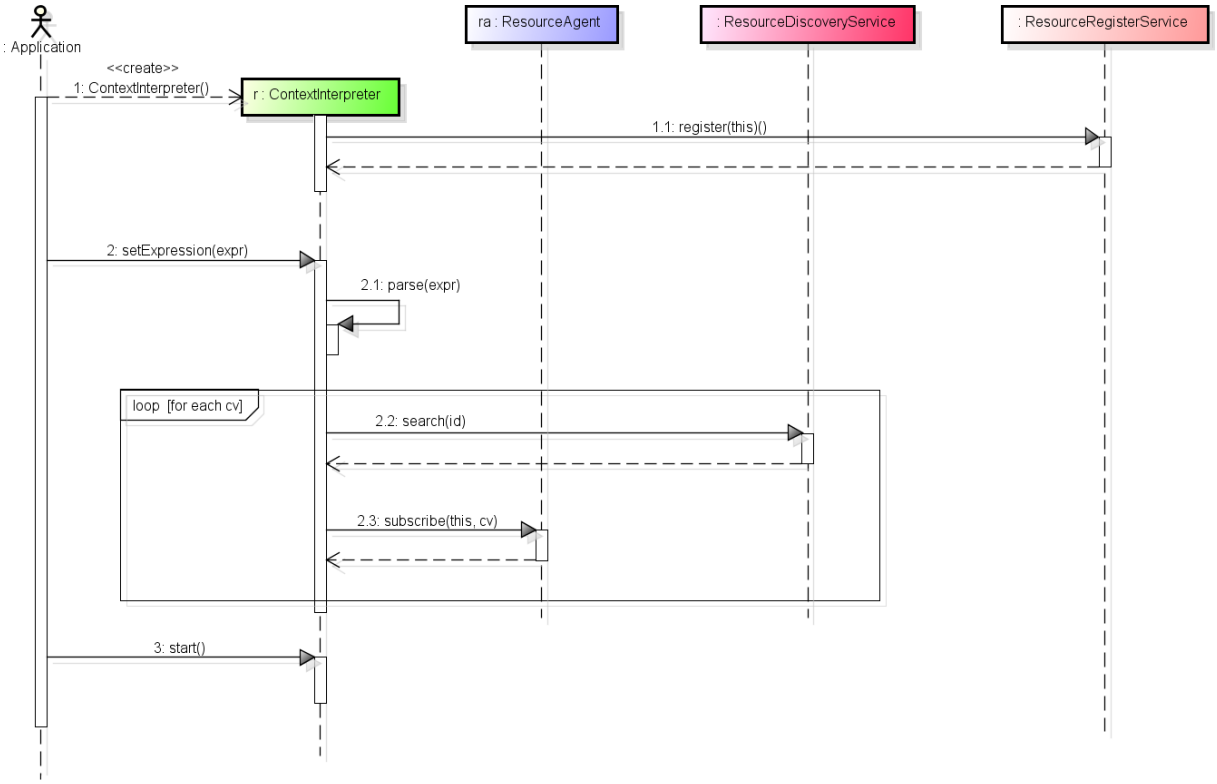


Figura 3.4: Diagrama de Sequência: Criação de IC

São identificados os ARs e as VCs referenciadas na expressão, o IC os procura no SDR (Etapa 2.2) e, com as referências, o IC se subscreve à cada do ARs (Etapa 2.3). Quando a aplicação invoca a operação "start" do IC (Etapa 3), este passa a executar a regra.

Uma vez que o IC esteja em funcionamento, ele apresenta um ciclo de execução como apresentado no diagrama de sequência da Figura 3.5. No início, um AR atuador busca o IC de interesse e se subscreve ao mesmo (Etapas 1 e 2), definindo assim uma regra de contexto. Eventualmente, o IC recebe uma notificação de uma VC referenciada na expressão com a mudança (Etapa 3) e a expressão é avaliada (Etapa 3.1). Durante a avaliação, é avaliado se a subexpressão contém um temporizador, o que pode ocasionar a temporização daquela subexpressão (Etapa 3.2). Ao final da avaliação, se a avaliação retornou verdadeiro, o AR atuador é notificado (Etapa 3.3).

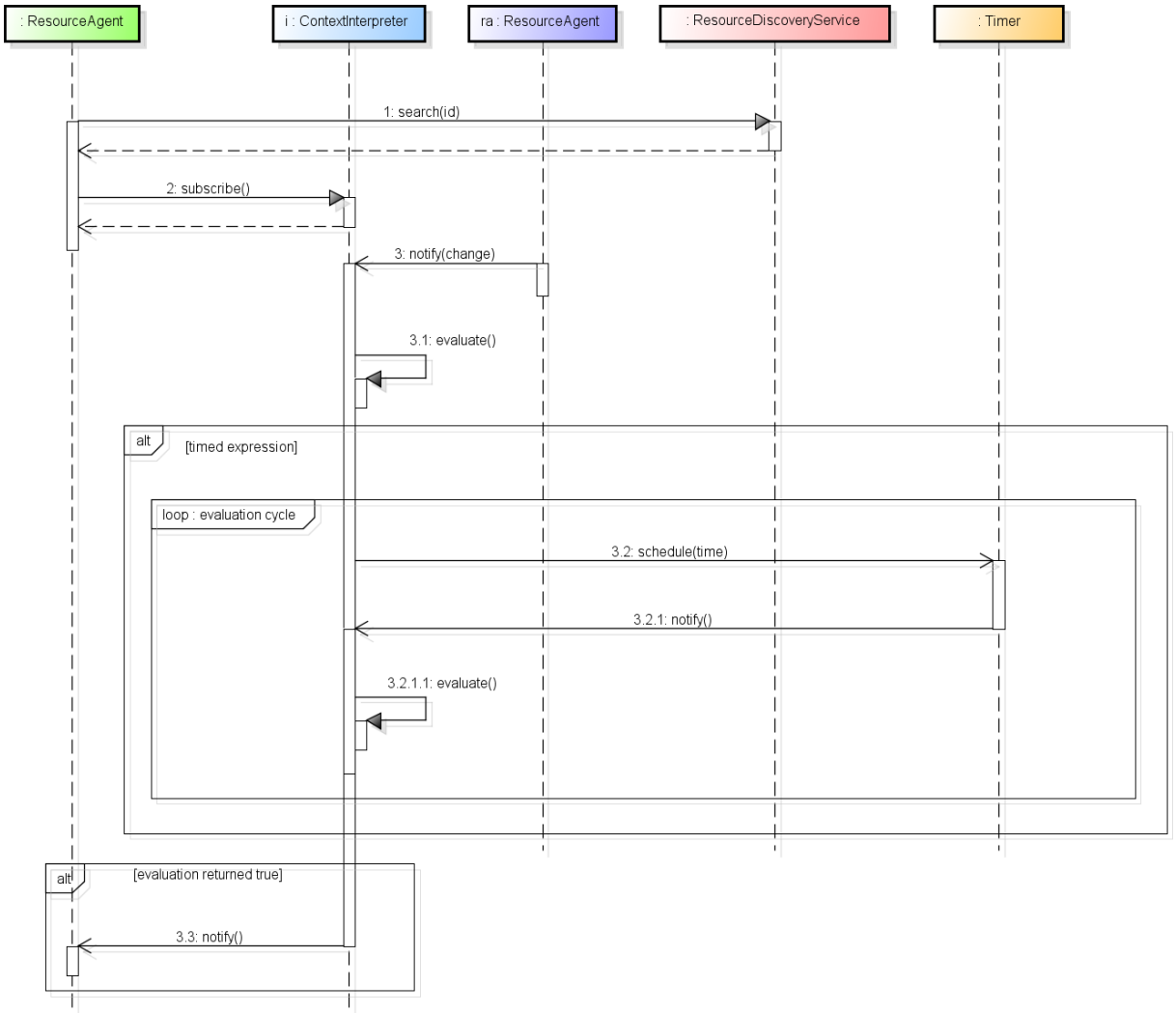


Figura 3.5: Diagrama de Sequência: IC em execução

3.4 Definindo Expressões de Regra

A criação do IC ocorre quando é definida uma expressão lógica no objeto do IC recém instanciado, segundo um padrão próprio adotado em arquivos do tipo JSON. Esta expressão é submetida ao *Parser*, que constrói uma Estrutura Lógica interna, como mostrado na Figura 3.6. 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, mas também os valores atualizados das mesmas. Enquanto em funcionamento, o IC recebe notificações das VCs, tratando-as em seguida para atualizar sua Estrutura Lógica (i.e., identificando a fonte corretamente para atualização da estrutura). O Módulo de Avaliação é notificado de mudanças no contexto e, consultando a Estrutura Lógica avalia se a expressão se tornou verdadeira ou falsa. O Módulo de Avaliação se encarrega de criar temporizadores conforme a parte temporizada da regra é avaliada como verdadeira, e se passou a ser avaliada como falsa, então aborta o temporizador criado. Quando o temporizador chega ao fim, então retorna uma notificação para o Módulo de Avaliação, que continua a avaliar a regra. Se a expressão alcançou um estado verdadeiro, a partir da avaliação e considerando as possíveis temporizações, então o IC notifica os atuadores.

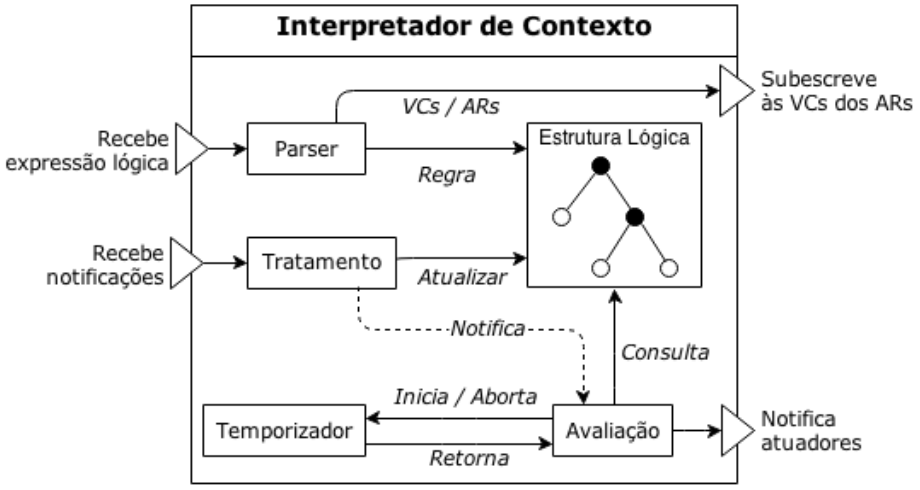


Figura 3.6: Estrutura do Interpretador de Contexto

Como exemplos de regras, a Figura 3.7 apresenta duas variações sobre a regra do "Fogão Esquecido" (ver na Seção 3.2). A primeira, sendo a mais simples, considera apenas se a pessoa está deitada e o fogão ligado durante 15 min. A segunda já utiliza uma subexpressão da regra e a cláusula "OU", e demonstra que regras que utilizam estes mecanismos podem ser mais expressivas. No exemplo da segunda regra considera-se que a pessoa pode estar deitada na cama ou sentada no sofá para fins de validade da regra. No terceiro exemplo de regra, o temporizador foi utilizado mais internamente na regra, ou seja, na

subexpressão. O efeito prático do terceiro exemplo é o de que a regra só será validada se a cozinha estiver vazia durante 15 min e o fogão estiver ligado, assim a pessoa só será avisada (na atuação) se estiver esquecido o fogão ligado e não estiver na cozinha, no caso, a regra assume que a pessoa na cozinha já estará observando o fogão, sem necessitar de aviso.

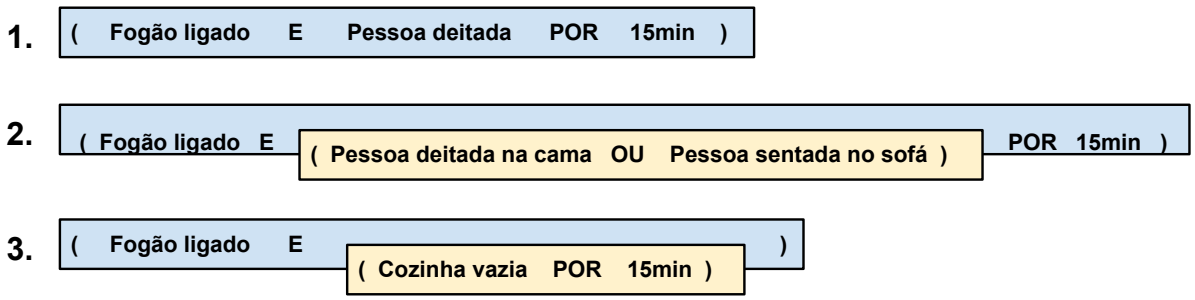


Figura 3.7: Exemplos de expressões de regras

O conjunto de símbolos atualmente aceito na expressão da regra inclui: operadores lógicos “E”, “OU”, “NÃO”; temporizadores; subexpressões, que têm sido aqui representadas entre parênteses; e a definição das condições, que são compreendidas por um par ID-VC (onde o ID é a identificação do AR no AmbI), um operador de comparação (i.e., igual, diferente, maior que, menor que, maior ou igual a, menor ou igual a), e uma constante (texto, número ou booleano) para comparação ou outro par ID-VC (que permite comparar uma VC de um AR com outra do mesmo ou de outro AR).

3.5 Temporização

A temporização em um IC é definida ao associar um ou mais temporizadores a uma subexpressão da regra (ou a expressão completa). Como representado pela Figura 3.7, um temporizador pode ser definido mais internamente na expressão da regra, ou seja, da subexpressão. Os temporizadores podem ser tanto relativos (e.g., fogão ligado e pessoa deitada na cama por 15 min) 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. Durante a avaliação, a expressão ou subexpressão pode se tornar falsa, o que faz com que os temporizadores associados sejam 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.

A temporização pode ser realizada ainda de outra maneira, através da subscrição à um AR especialmente criado para este fim, no caso, um "Agente Temporizador". Este AR possui internamente temporizador implementado, e oferece uma VC referente ao término da temporização ("*timeout*"), além de OPs para inicializá-lo e interrompê-lo. Assim, ao montar uma regra, basta fazer uma operação lógica "E" entre a expressão e esta VC.

3.6 Implementação da Geração da Regra

O JSON é um padrão aberto baseado em texto legível que estende a linguagem de programação JavaScript e possibilita o armazenamento de dados de forma semiestruturada. Sua adoção no *SmartAndroid* facilitou a etapa de análise léxica da regra (*tokenization*), visto que existem bibliotecas para Android que leem uma cadeia de caracteres (*string*) formatada em JSON e constroem objetos JSON em memória. A etapa de construir a árvore (Estrutura Lógica) consiste portanto em primeiramente fazer a análise (*parsing*) do texto para objetos JSON, seguido da geração de um árvore de objetos dos tipos utilizados pelo método avaliador do IC.

A Listagem 3.1 apresenta uma variação da expressão da regra "Fogão Esquecido" escrita em JSON. As IDs "_STOVE_ID_", "_BED_ID_" e "_TV_ID_" são as identificações dos ARs do fogão, da cama e da TV, respectivamente. Neste exemplo há duas condições: a primeira verifica que a VC "*ovenTemperature*" do fogão é maior do que 100º C, e a segunda verifica que a VC "*occupied*" da cama é verdadeira. Além disso é declarado um temporizador de 10 s que garante que esta expressão será válida durante este tempo. Regras mais complexas podem ser criadas aninhando-se expressões dentro de outras. A utilização da VC de temperatura do forno, ao invés de fogão ligado, ocorreu apenas para fins de ilustração do exemplo.

Na parte da atuação, como mostrado na Listagem 3.1, é definido juntamente ao IC um atuador que desempenha duas ações dado a ocorrência de validade da regra, são elas: o desligamento do forno (*"turnOffOven*") e a exibição de uma mensagem na TV do quarto (*"showMessage*").

Listagem 3.1: Regra em JSON

```
{"interpreter": {
    {"name" : "Interpretador Fogão Esquecido"},
    {"expression": [
        {"condition": [
            {"context_elem": {
                "rai": "_STOVE_ID_",
                "cv": "ovenTemperature"
            }},
            {"op": ">"},
            {"val": 100.0}
        ]},
        {"connective": "and"},
        {"condition": [
            {"context_elem": {
                "rai": "_BED_ID_",
                "cv": "occupied"
            }}
        ]},
        {"timer": 10sec}
    ]}
}, "actuator": [
    {"name" : "Atuador Desliga Fogão"},
    {"action": {
        "rai": "_STOVE_ID_",
        "op": "turnOffOven"
    }},
    {"action": {
        "rai": "_TV_ID_",
        "op": "showMessage",
        "params": "Fogao foi esquecido ligado"
    }}
]}
```

O *parser* implementado executa uma rotina recursiva que navega pelos objetos do tipo JSON segundo uma abordagem descendente (*top-down*) criando os objetos referentes aos

nós da árvore. A Listagem 3.2 apresenta este método de forma simplificada em linguagem aproximada do Java, que foi a linguagem utilizada na implementação real. Primeiramente é obtida a cadeia de caracteres contendo a expressão da regra. O objeto JSON é criado a partir desta cadeia de caracteres. Em seguida é criado o nó raiz da árvore ("*root*") e a rotina "*buildTree*" é chamada, recebendo como argumento a expressão da regra em JSON e o ponteiro para o nó raiz.

O que a rotina "*buildTree*" faz é percorrer os pares chave-valor do objeto JSON recebido identificando qual o tipo da chave e criando o equivalente para o IC. São nós terminais da árvore os seguintes:

- "condition" define uma condição que pode ser formada por:
  - Uma VC, um operador de comparação (i.e., $=, \neq, <, >, \leq$ e $\geq$) e outra VC
  - Uma VC, um operador de comparação e um valor constante (i.e., inteiro, booleano ou texto)
  - Uma VC apenas, que possui valor booleano
- "connective" define qual conectivo liga as condições, subexpressões, ou condição e subexpressão, podendo ser "E" ou "OU"
- "timer" define que há um temporizador na expressão, ou na subexpressão onde está localizado, e qual é o tempo, com unidade de medida em segundos, minutos e horas

Os nós não terminais definem subexpressões na regra permitindo que a expressão lógica relacione não apenas condições ("*condition*"), mas também uma condição com uma subexpressão ou duas subexpressões. São nós não terminais na árvore os seguintes:

- "not" define uma negação de uma subexpressão localizada internamente
- "expression" define uma subexpressão na regra

O resultado final é uma estrutura de dados de árvore que utiliza a Orientação à Objetos para definir todos os diferentes tipos de nós como implementações da classe abstrata "*Node*". Cada um dos nós não terminais aponta para os que estão abaixo (nós filho), o que mantém a estrutura.

Listagem 3.2: Construção da Estrutura de Dados

```
// Aponta para o nó raiz da árvore
String expression_str = "...";
JSON json = getJSONObject(expression_str);
Node root = new ExpressionNode();
buildTree(json, root);

// Constrói a árvore
private void buildTree(JSONObject json, Node expression) {
    Object value;
    // Para cada chave no objeto JSON, criar o nó
    foreach (string key in json.getkeys()) {
        value = json.getKey(key);
        // São terminais os nós "condition", "timer" e "connective"
        // Os nós "not" e "expression" são não-terminais
        switch (key) {
            case "condition" :  { expression.attach(getCondition(value)); }
            case "timer" :      { expression.attach(getTimer(value)); }
            case "connective" : { expression.attach(getConnective()); }
            case "not" : {
                Node node = new NotNode();
                expression.attach(node);
                buildTree(value, node);
            }
            case "expression" : {
                Node node = new ExpressionNode()
                expression.attach(node);
                buildTree(value, node);
            }
        }
    }
}
```

3.7 Implementação da Avaliação da Regra

A implementação atual do IC pode ser observada no diagrama de estados da Figura 3.8 que apresenta a criação e avaliação da regra incluindo os mecanismos que relacionam a temporização à regra propriamente dita, de maneira detalhada. A seguir são descritas as diferentes etapas do processo.

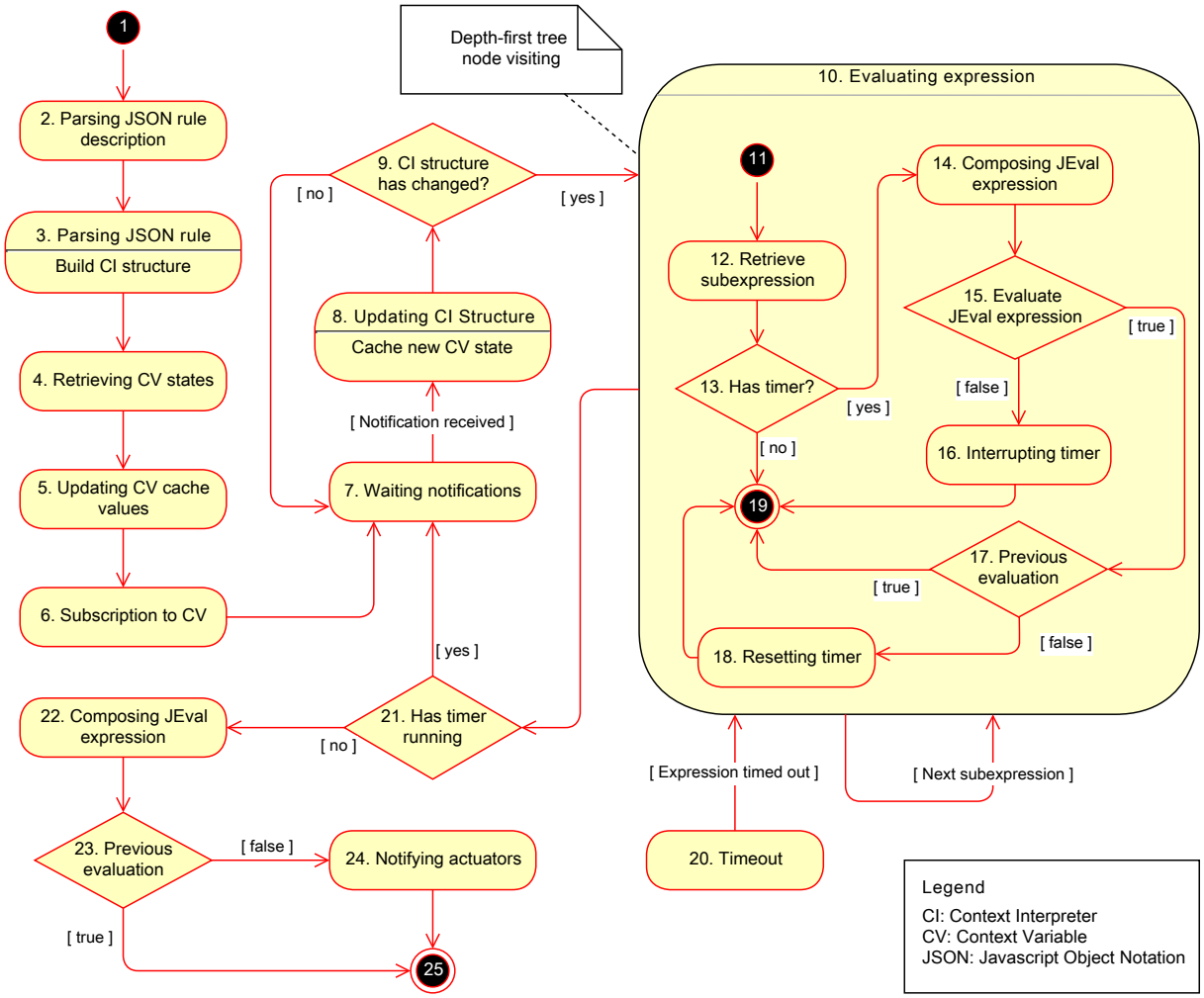


Figura 3.8: Diagrama de Estado – Criação e Avaliação de ICs

Criação No diagrama da Figura 3.8 é mostrado um IC que é criado a partir de uma regra em JSON. Do Estado 1 ao 6 ocorre como apresentado no diagrama de sequência da figura com a inicialização do IC. Onde inicialmente é lido o cabeçalho da regra (Estado 2), para fazer o *parse* em seguida (Estado 3). Os valores atuais das VCs são recuperados (Estado 4), guardados em cache (Estado 5) e o IC se subscreve nas VCs (Estado 6).

Monitoramento Os estados 7, 8 e 9 correspondem ao recebimento de atualizações de VCs, tratamento e atualização da Estrutura Lógica apresentado na Figura 3.6. Se houve mudança na estrutura, então se inicia o estado 10 de avaliação, senão volta para o estado 8 de espera de notificações.

Avaliação Este diagrama detalha melhor a etapa da avaliação como ocorre na implementação.

Estrutura Lógica Como a estrutura de dados que armazena a regra assim como

o valor atualizado das VCs é uma árvore, para avaliar a regra foi adotado um algoritmo de busca em profundidade (*depth-first search* – DFS), que explora tanto quanto possível cada um dos ramos da árvore, antes de retroceder (*backtracking*).

Iteração na árvore Utilizando o DFS podemos avaliar primeiramente as subexpressões mais profundas da árvore. A avaliação de uma subexpressão no diagrama, considerando a passagem de tempo, ocorre no estado 10 e através da transição "Next subexpression" a próxima subexpressão é avaliada até que a regra esteja inteiramente avaliada.

Avaliação de subexpressão Dentro do estado 10, o diagrama se aprofunda na avaliação da subexpressão. Isto consiste da recuperação da subexpressão (Estado 12) e de etapas subsequentes que consideram se há um temporizador associado, da avaliação propriamente dita da expressão lógica (Estados 14 e 15), e verificação de mudança de estado de validade na subexpressão – i.e., se era verdadeira e se tornou falsa, se era falsa e se tornou verdadeira, se era falsa e continuou falsa, e se era verdadeira e se tornou falsa – (Estado 17). Isto garante que os atuadores não serão novamente notificados caso a expressão tenha mudado de um estado válido para um novo estado válido e que temporizadores sejam interrompidos (Estados 16 e 18) caso a expressão tenha se tornado falsa, economizando recursos da máquina.

Avaliador JEVAL A avaliação propriamente dita da subexpressão ocorre através da conversão da expressão para o formato usado pelo JEVAL. O JEVAL é uma API utilizada para avaliar expressões matemáticas, de cadeias de caracteres (*strings*), booleanas e funcionais [28], onde são as expressões boolenas que nos interessam. Nas expressões que são convertidas para o formato JEVAL as VCs já são devidamente substituídas por constantes, que são o valor atual VC (em cache). Se o temporizador da expressão ou subexpressão ainda não terminou, isto é o equivalente a fazer uma operação lógica "E" com o valor "falso", e se já terminou então é equivalente à uma operação "E" com o valor "verdadeiro".

Timeout O fim da temporização gera também a avaliação da subexpressão como observa-se no estado 20 do diagrama.

Avaliação da expressão raiz Uma vez que todos ramos da árvore tenham sido devidamente avaliados, há uma transição para o Estado 21, onde é verificado se ainda há algum temporizador em funcionamento. Em caso positivo, então se transita no-

vamente para o estado de espera (Estado 7), caso contrário, a expressão é avaliada completamente utilizando o JEVAL (Estado 22). Verifica-se se a regra mudou de um estado falso para verdadeiro (Estado 23), e, em caso positivo, então os atuadores são notificados (Estado 24).

3.8 Regras de Contexto na IPGAP

Além da motivação de facilitar a construção de aplicações ubíquas pelos desenvolvedores, visamos neste trabalho também prover uma interface na qual usuários sem experiência técnica sejam capazes de construir regras de contexto, assim como gerenciá-las. Esta motivação está relacionada ao desafio (iii), que se refere à disponibilidade de recursos no ambiente. Ao utilizar a IPGAP (apresentada inicialmente na Seção 2.8) para compor regras, os usuários e desenvolvedores poderão criar e prototipar regras antes de adquirir os recursos reais, trabalhando inicialmente com simuladores, diminuindo custos.

Além da disponibilidade dos recursos, outras questões têm atrasado a adoção da automação em ambientes residenciais, e também em outros AmbIs, pelo ponto de vista dos usuários finais, como identificado por Brush 2011 [8]. Dentre elas, estão a *capacidade de personalização do ambiente* e a *complexidade das interfaces de usuário*.

Como será visto nesta seção, para atender a questão da *capacidade de personalização do ambiente* propomos a interface de composição e gerenciamento de regras de contexto integrada com a IPGAP, que visa atender a demanda dos usuários de construir regras para o seu AmbI e gerenciá-las, assim como gerenciar regras construídas pelas aplicações. Utilizando esta interface, regras como a do "Fogão Esquecido" (ver na Seção 3.2) podem ser facilmente criadas pelos usuários, que também podem controlar seu estado de execução.

Embora este trabalho não esteja inserido na área de Interação Humano-Computador (IHC) onde costuma-se medir a usabilidade de uma interface pelos usuários, visamos atender à questão da *complexidade das interfaces de usuário* ao tornar a criação das regras de contexto integrada com o mapa do ambiente. A interface foi construída de tal maneira que, através de uma tela sensível ao toque, um usuário possa criar a regra selecionando VCs de recursos do ambiente e definindo as operações lógicas através de um menu.

O roteiro para criação de uma regra na IPGAP envolve a criação da expressão da regra e a definição dos atuadores. A expressão é criada por meio da definição de comparações (entre uma VC e um valor, ou entre duas VCs) e da lógica que inter-relaciona essas

comparações (usando os operadores lógicos “E”, “OU” e “NÃO”, e parênteses), que ocorre por meio de toques na tela. Uma comparação é realizada em três etapas:

1. Seleção do AR no mapa e escolha da VC
2. Escolha de um operador de comparação ($=, \neq, <, >, \leq$ e $\geq$)
3. Definição de um valor, ou de uma outra VC de um AR selecionado
4. Incluir um temporizador, seja na comparação, ou na expressão completa

A Figura 3.9 apresenta uma interface da IPGAP sendo exibida em um *tablet* com tela sensível ao toque (Galaxy Tab 10.1), onde o mapa do ambiente está ao fundo e o menu de ferramentas está aberto na parte inferior da figura. Estão representadas no mapa ainda cinco recursos, são eles: duas lâmpadas (uma no quarto e a outra na sala), um fogão (na cozinha), uma televisão de tela plana (na sala) e uma cama (no segundo quarto).

Figura 3.9: Inicialização da composição de regras

Quando, em operação normal, o usuário clica em um recurso, é aberto um controlador (ou um simulador) daquele recurso. Nesta figura o usuário clicou no menu de ferramentas no botão *Compose a Context Rule*, que fez com que a interface mudasse seu modo de operação para o modo de composição de expressão de regra. Isto fez com que o menu lateral esquerdo fosse aberto para apresentar a regra em construção, e mudou o efeito do

clique em um recurso do ambiente, que agora, ao invés de abrir o controlador do recurso, apresenta um menu com a lista de VCs do AR do recurso, como mostrado na Figura 3.10. Nesta figura, o usuário clicou no fogão, o que fez com que fosse exibida a lista de suas VCs. Conforme o usuário monta a regra, ela é exibida no menu lateral esquerdo, como também pode-se ver nesta figura.

Figura 3.10: Seleção de Variável de Contexto

Além da escolha dos ARs e de suas VCs, é necessário indicar o operador de comparação e escolher o objeto de comparação, que pode ser uma constante (a ser digitada) ou outra VC (a ser escolhida no mapa). Para isso, após o usuário escolher uma VC na tela, é apresentado o menu representado na Figura 3.11 no qual o usuário escolhe o tipo de comparação e o tipo de objeto a ser comparado. Além de criar comparações, este menu apresenta ferramentas para escolher as operações lógicas para relacionar essas com-

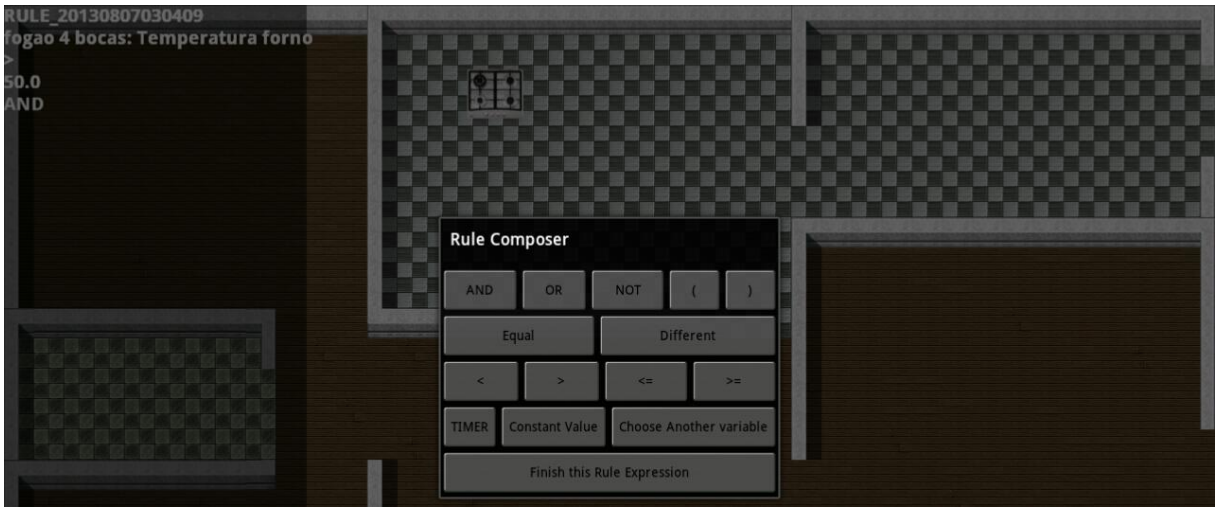


Figura 3.11: Menu de composição de regras

parações, criar subexpressões (selecionando parênteses) e acrescentar temporizadores à expressão. Estas escolhas são feitas sequencialmente durante a composição da regra.

Uma vez que a expressão da regra esteja completa, o usuário deve clicar em *Finish this Rule Expression* no menu, o que faz com que um IC seja instanciado com a expressão criada e inicializado. Um arquivo JSON também é gerado, contendo a definição da regra, para persistência. Isto possibilita que o IC seja novamente instanciado caso o sistema reinicie. Em seguida, a interface muda para o modo de composição do atuador.

Na Figura 3.12 a IPGAP está no modo de composição do atuador, onde o menu lateral é usado para listar as ações escolhidas e um menu central aparece sempre que o usuário desejar escolher uma nova ação para o atuador. A escolha de ações ocorre de modo análogo à definição de uma comparação, i.e., o usuário deve selecionar um AR no mapa, escolher uma OP e definir os parâmetros da OP (e.g., ar-condicionado: mudar temperatura para 20º C). Após terminar a escolha das ações, um atuador é instanciado para desempenhar estas ações no ambiente, e é subscrito ao IC anteriormente criado. Além disso, a definição do atuador é escrita no mesmo JSON do IC, possibilitando que, se o sistema reiniciar, não só o IC seja instanciado, mas também o seja o atuador, e que o segundo seja subscrito no primeiro.

Figura 3.12: Menu de seleção de ações

A atual arquitetura possibilita que um ou mais atuadores sejam ligados ao IC ou desligados, utilizando o mecanismo de subscrição. Assim é possível que, através da IPGAP, o usuário selecione um atuador já existente para ligar ao IC recém definido. Esta alternativa busca unir a possibilidade da definição da regra em alto nível por usuários finais, com o poder da criação de atuadores no nível de desenvolvimento.

Não apenas as aplicações, mas também as regras de contexto devem ser passíveis de ser controladas pelos usuários. Na Seção 3.3 foi apresentado o ciclo de vida dos ICs, que conta basicamente com os estados: em criação, em execução (avaliação da regra em face do contexto), parado e finalizado (parada definitiva). Esta seção já apresentou o método utilizado, integrado à IPGAP, para a criação de regras em nível de usuário. Os outros estados também podem ser controlados pelos usuários, pois o IC define OPs para o seu controle, i.e., para parar sua execução, finalizar e reiniciar. Para dar este suporte no nível do usuário, foi implementada uma interface que permite a interrupção ou reinicialização da regra, assim como começar a criação de uma regra nova, apresentada na Figura 3.13.

O objetivo desta interface é capacitar os usuários para personalizar o ambiente não apenas criando novas regras de contexto, mas também gerenciando as já existentes. Esta ferramenta possibilita que o usuário lide com conflitos de interesses, i.e., se uma regra foi criada para prover uma facilidade mas está gerando um distúrbio, neste caso ela pode ser simplesmente desativada.

Figura 3.13: Controle de Regras de Contexto

3.9 Conclusões do Capítulo

Este capítulo detalhou a proposta da dissertação, que corresponde aos recursos do *framework* para se utilizar de sensibilidade ao contexto na construção de novas aplicações. Em especial se focou na definição dos Interpretadores de Contexto, e como é possível se criar regras de contexto de maneira flexível. Foi também apresentado o recurso de criação de regras de contexto através do *SmartAndroid*, que permite uma separação de interesses entre o conjunto de regras e a lógica da aplicação, e foi viabilizado através da implementação de um *parser* para gerar os agentes necessários para implantar a regra.

Do ponto de vista dos usuários finais, foi apresentada a interface de composição e gerenciamento de regras de contexto que funciona integrada ao mapa do ambiente provido pela IPGAP.

No próximo capítulo, é feita uma avaliação sobre a proposta, considerando pontos centrais do projeto e desenvolvimento.

Capítulo 4

Avaliação

Este capítulo apresenta uma avaliação do *framework* conceitual e de sua implementação de referência, no que concerne ao tema central desta dissertação. São abordadas algumas das principais características da proposta, assim como implementações que as atendem.

A avaliação consiste na formulação de questões de competência diretamente relacionadas à proposta. Esta técnica tem sido bastante utilizada no desenvolvimento de ontologias, tendo sido primeiramente sugerida por Grüninger e Fox [26]. Ela consiste da criação de perguntas que delimitam um problema que se deseja estudar e prover uma solução.

4.1 Questões de Competência

No contexto desta dissertação, questões de competência foram formuladas e as aplicações exemplo são utilizadas para demonstrar sua conformidade com os requisitos evidenciados pelas questões. Estas questões foram elaboradas com foco nos requisitos relacionados à sensibilidade ao contexto do *framework* e do suporte do *SmartAndroid*.

Dois grupos foram criados para abrigar as questões de competência, como podem ser vistos a seguir:

1. Questões relacionadas à interpretação do contexto
   (a) Como utilizar o *parser* para se gerar regras de contexto?
   (b) Como definir temporizadores em uma regra de contexto?
   (c) Como é possível criar um Interpretador de Contexto?

(d) Como ligar Interpretadores de Contexto a Atuadores?

(e) Como utilizar a abstração de um Interpretador de Contexto em outro?

2. Questões de suporte aos usuários

(a) Como é possível prototipar regras de contexto no ambiente?

(b) Como usuários podem definir suas preferências no ambiente?

(c) Como é possível gerenciar as regras de contexto em execução?

O Grupo 1 um contém os recursos do *framework* que possibilitam a construção de aplicações sensíveis ao contexto com maior foco nos desenvolvedores. Estas questões avaliam a capacidade para interpretação do contexto e do suporte provido para este fim. O Grupo 2 possui maior foco nos usuários finais, e avaliam o suporte aos usuários para interação e configuração do AmbI.

4.2 SmartLiC

A aplicação ubíqua de controle de iluminação inteligente (Smart Light Control – SmartLiC) visa otimizar o uso de iluminação em residências a fim de diminuir o consumo de energia elétrica, visto que esta é uma grande fonte de gasto energético. A aplicação usa sensores de presença para identificar a localização de pessoas em cada cômodo, e decide quais lâmpadas devem ser ligadas ou desligadas. Assim, se previne o esquecimento de lâmpadas acesas, e pode também acender lâmpadas se uma pessoa entra em um cômodo à noite e se ninguém está dormindo naquele cômodo. A aplicação também leva em consideração se é dia ou noite, acendendo lâmpadas somente no segundo caso, além de oferecer um mecanismo para bloquear lâmpadas de acenderem em um cômodo em particular, evitando atuar de maneira indesejada.

Para avaliar a aplicação em funcionamento, foram criados simuladores para os recursos do ambiente, assim como a simulação de uma pessoa para representar o usuário. Estes simuladores implementam interfaces equivalentes às que um recurso real implementaria. Por exemplo, o AR do sensor de presença está localizado em um cômodo e possui uma VC que indica se uma pessoa entrou no mesmo cômodo. Um sensor real dispararia uma notificação para esta VC caso efetivamente detectasse uma pessoa, o que é feito através da IPGAP no caso do simulador. O mesmo é valido para as lâmpadas e o sensor de luz do dia simulados.

4.2.1 Criação de Regras de Contexto

Para a aplicação foram criadas dois tipos de regras de contexto. A primeira tem por objetivo verificar se alguém entrou no cômodo (i.e. se ele foi ocupado), e se é dia. Neste caso, a regra acende a lâmpada do cômodo, como mostra a Listagem 4.1.

Listagem 4.1: Regra Acender Lâmpada

```
1  IF (ComodoX.ocupado AND NOT (SensorLuminosidade.dia))
2  THEN
3  LampadaX.ligar();
```

A segunda regra avalia se o cômodo ficou vazio e a luz foi deixada acesa, durante um tempo T (na implementação foi usado 10 s, para facilitar os testes). Como consequência, esta regra apaga a luz do cômodo, como mostra a Listagem 4.2.

Listagem 4.2: Regra Apagar Lâmpada

```
1  IF (NOT (ComodoX.ocupado) AND LampadaX.acesa FOR Tsec)
2  THEN
3  LampadaX.desligar();
```

Na implementação de cada uma das regras foi utilizado um esqueleto de regra em JSON que não define o cômodo e a lâmpada em específico, deixando a cargo da aplicação definir. Isto evita a repetição de código e o conhecimento prévio da quantidade de cômodos e lâmpadas. O SensorLuminosidade, que detecta se é dia, não está presente em cada cômodo e só é necessário que haja um, localizado fora da casa.

Durante a inicialização da aplicação, os arquivos JSON são lidos, os sensores de presença (que indicam se um cômodo foi ocupado) e as lâmpadas são recuperados através de uma busca usando o SDR. O esqueleto, em formato de texto, é então preenchido com as identificações dos ARs e submetido ao *parser*. Como resultado são instanciados ICs para cada uma das instâncias de regras. Os atuadores criados são subscritos aos ICs e desempenham a ação respectiva no cômodo.

A solução para a criação de regras de contexto em JSON atende à **Questão 1a**, relacionada ao uso do *parser*. Nesta implementação as referências para os ARs foram incluídas no esqueleto da regra em JSON, mas as referências também poderiam ter sido diretamente incluídas nos arquivos.

4.2.2 Declaração de Temporizadores

A declaração do temporizador é feita durante a criação da regra de contexto em JSON, apenas informando a quantidade de tempo de validade da expressão de regra para que seja avaliada como verdadeira. O temporizador é ligado à expressão e o *parser* o considera como uma operação lógica "E" com uma subexpressão que retorna "falso". Contudo, esta subexpressão é condicionada à expressão que está ligada ao temporizador e se a expressão continuar verdadeira até que o temporizador chegue ao final, a subexpressão do temporizador será tornada verdadeira, mas se a expressão mudar frequentemente à sua avaliação para falso, o temporizador será reiniciado frequentemente antes de chegar ao seu estado "verdadeiro". Além deste exemplo para o uso de temporizadores, foram criadas variações do exemplo "Fogão Esquecido" usando temporizadores no Apêndice A.

Os temporizadores podem ser resolvidos internamente no IC, como foi o caso desta implementação, mas também pode ser adicionado diretamente como uma VC à regra. Neste caso, basta referenciar um AR que implementa um temporizador, na VC de *timeout*. Quando esta VC disparar, a regra será avaliada normalmente, o que gera o mesmo efeito final da primeira abordagem. Desta maneira, é respondida à **Questão 1b**, que concerne à adição de temporizadores às regras de contexto durante o desenvolvimento.

4.2.3 Interpretador de Contexto

Uma evolução do SmartLiC é possibilitar a identificação de uma pessoa dormindo em um cômodo à noite, o que evitaria que a lâmpada fosse acesa. Uma solução para isto é a construção de um IC que identifique se há uma pessoa dormindo no cômodo. Isto poderia ser feito utilizando diferentes tipos de sensores e o IC, como um agregador, para prover esta informação em nível mais alto.

Considere um cenário onde uma pessoa possui em sua residência uma cama inteligente (que informa se alguém a está utilizando) e um acelerômetro costurado em sua roupa (para identificar se a pessoa está deitada). Um IC poderia identificar se a pessoa está deitada e utilizando a cama concomitantemente durante um tempo T, para afirmar que a mesma está dormindo com maior confiabilidade. Se houverem outros sensores disponíveis, a informação pode inclusive ser mais confiável, como câmeras (que podem ter suas imagens tratadas) e sensores de frequência cardíaca (que geralmente é mais baixa quando a pessoa está dormindo).

Para criar este IC adota-se o mesmo passo-a-passo utilizado pelo SmartLiC para ins-

tanciar cada um de seus ICs que monitoram a movimentação das pessoas pelos cômodos. Para isto é necessário: a instanciação de um objeto da classe *ContextInterpreter*, que aqui pode ser chamado por exemplo de *InterpretadorPessoaDormindo*; a definição da expressão de regra, o que é realizado ao se invocar o método *setExpression* passando como parâmetro a expressão de interesse; e a inicialização do IC. Isto responde a **Questão 1c**, sobre como o IC é criado. A inicialização do IC faz com que ele se registre no AmbI usando o SRR e se subscreva aos ARs referenciados na expressão.

Um simples atuador para apagar a luz do quarto quando a pessoa for dormir pode ser definido e ligado ao IC *InterpretadorPessoaDormindo*. Para obter a referências para o AR da lâmpada basta fazer uma consulta por tipo usando SDR e escolher a lâmpada localizada no cômodo de interesse. Com esta referência, o atuador gera um *stub* para invocar a OP de apagar luz da lâmpada, que é acionada quando houver o retorno do IC. Para ligar o atuador no IC, como já discutido, basta fazer com que o primeiro se subscreva na VC do segundo que, para o IC, apenas informa se ele está em um estado válido ou inválido. Desta maneira é respondida a **Questão 1d** referente ao processo de ligação de ICs e atuadores.

Uma vez que é possível descobrir se há uma pessoa dormindo no cômodo, faz-se necessário adaptar a regra Acender Lâmpada para considerar esta informação. Para esta implementação, o IC da regra Acender Lâmpada deve referenciar o outro IC que identifica pessoas dormindo (i.e. *InterpretadorPessoaDormindo*). Como o IC possui um estado (i.e., válido ou inválido) e uma VC para publicar a mudança deste estado, então ele também pode ser referenciado na regra de outro IC utilização o mecanismo de subscrição padrão. Isto responde a **Questão 1e** sobre como ligar diferentes ICs.

4.2.4 Prototipagem e Teste

A prototipagem e teste da aplicação foi auxiliada com o suporte da IPGAP e do uso de simuladores que implementam as mesmas interfaces que possíveis recursos reais. Os simuladores permitiram o teste da aplicação sem a necessidade de compra dos dispositivos reais, o que diminuiu os custos; e a IPGAP favore a observação da aplicação em funcionamento, ou seja, vemos a mudança no contexto conforme ações acontecem no ambiente, sem que seja necessário acompanhar a execução da aplicação em *logs*.

No caso do SmartLiC, a principal ação que ocorre no ambiente é a movimentação da pessoa. Desta movimentação decorrem as ações de acender e apagar lâmpadas, o que poderia inclusive disparar a execução de outras regras de contexto em funcionamento no

AmbI.

Além do mapa, são representados na interface a pessoa e as lâmpadas. A mudança nas VCs destes ARs gera mensagens na tela abaixo do símbolo do recurso. No caso das lâmpadas as mensagens podem ser: "`isOn = true`" e "`isOn = false`", respectivamente: ligada e desligada. Para a pessoa, a única VC que muda no exemplo é sua localização, e a mensagem é "`position = (x=<valor de x>, y=<valor de y>)`", onde "`<valor de x>`" e "`<valor de y>`" são as coordenadas cartesianas do recurso no mapa.

O SLR do *SmartAndroid* permite a definição da localização de um recurso baseado tanto na posição em coordenadas cartesianas como na presença em um cômodo. Isto possibilita descobrir se um AR muda de lugar em um mesmo cômodo, se muda não só de posição mas também de cômodo, se está no mesmo cômodo que outro AR, se tem algum AR de um determinado tipo em um cômodo, além de possibilitar o mapeamento do ambiente pela IPGAP.

Considere o cenário de uso da SmartLiC que se inicia na Figura 4.1. Após a inicialização do ambiente, incluindo a criação de todos os cômodos, que só existem virtualmente, é inicializada a SmartLiC. Inicialmente são instanciados os simuladores de lâmpada e sensor de presença para todos os cômodos, e um simulador de sensor de luz do dia. Além disso, foi criado um avatar para o usuário. A Figura 4.1 apresenta as lâmpadas localizadas nos diversos cômodos e a pessoa localizada na sala de estar com a lâmpada ligada. Os demais componentes do ambiente, como móveis, e outros recursos, não foram incluídos neste exemplo. Foram também instanciados os ICs e Atuadores como apresentado na Subseção 4.2.1.

O avatar do usuário é controlado por uma interface da IPGAP que funciona separadamente, chamada Interface de Emulação de Movimento. Esta interface permite emular a movimentação dos recursos e avatares no AmbI, ao simplesmente se arrastar um símbolo e soltar no lugar desejado, ou também ao definir uma rota para o recurso. A Figura 4.2 apresenta a definição para uma possível movimentação do avatar.

No exemplo da SmartLiC, o avatar foi controlado em um dispositivo em separado. Um *smartphone* foi utilizado para movimentar o avatar, enquanto a IPGAP era atualizada com sua nova posição, além dos outros eventos que estavam ocorrendo no AmbI. A Figura 4.3 apresenta o caminho que a pessoa fez, ao sair da sala de estar, para a sala de jantar. Imediatamente, a luz da sala de jantar acendeu, ao considerar não só a entrada da pessoa, mas também o fato de ser noite, como representado na figura.

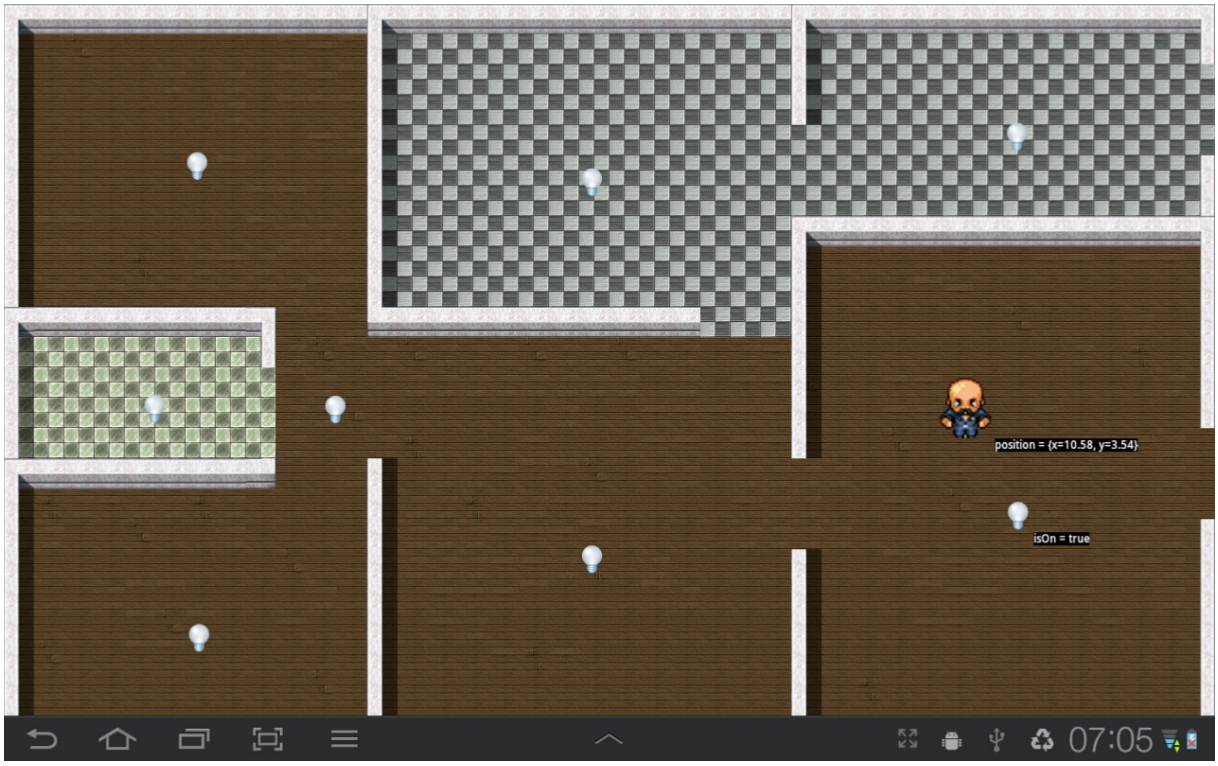


Figura 4.1: SmartLiC - Pessoa entrando em casa

Seguindo o efeito da segunda regra de contexto, a luz da sala de estar deve desligar após o sensor não detectar presença durante o tempo estipulado. A Figura 4.4 demonstra esta ação ocorrendo.

A solução apresentada para o uso de simuladores aliada ao uso da IPGAP atende à **Questão 2a**, referente ao requisito de prototipagem rápida. Foi possível observar como, com o suporte deste ferramental, é possível de construir aplicações e regras de contexto para o AmbI com o mínimo de recursos. A prototipagem de regras de contexto é possível quando estas são executadas em um ambiente controlado, e com o acompanhamento não apenas do *log* de execução, mas também com a visualização do ambiente em funcionamento integrado. Esta questão está relacionada tanto aos desenvolvedores, no decorrer do desenvolvimento da aplicação, como aos usuários finais, que podem criar regras de contexto em alto nível e precisam testá-las.

Como futura implementação para o SmartLiC, um modo de segurança da residência será incluído. Esta função aprenderá sobre o uso da iluminação na residência e no momento em que estiver vazia, e a função estiver ativada, as luzes serão acesas seguindo uma sincronia realista para enganar ladrões em potencial. Esta é uma função muito básica, mas útil em lugares onde é comum que pessoas mal intencionadas observem o fluxo da residência a fim de cometer um furto.

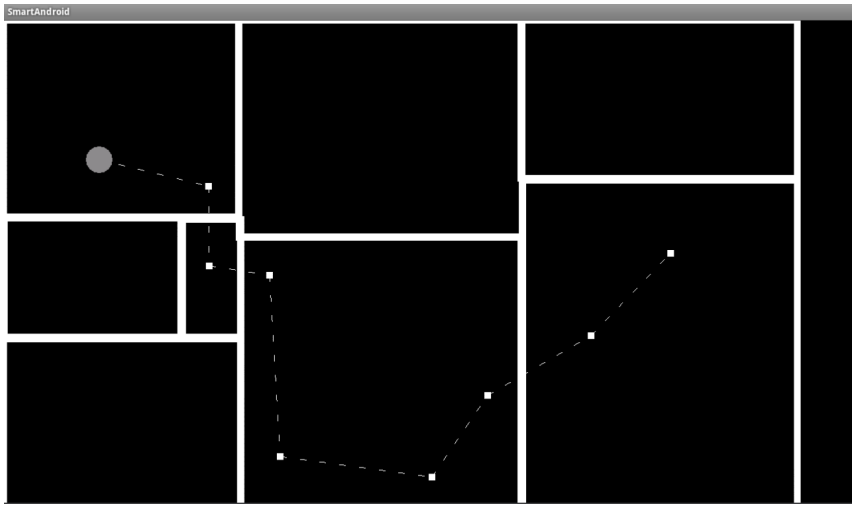


Figura 4.2: Interface de Emulação de Movimento

Figura 4.3: SmartLiC - Pessoa mudando de cômodo

4.3 Regras de Contexto Criadas por Usuários

Um AmbI é construído para suportar as aplicações ubíquas e as regras de contexto, mas seu objetivo último é atender as demandas das pessoas. Para que usuários sem experiência técnica possam desenvolver suas próprias regras de contexto no ambiente, foi desenvolvido na IPGAP uma interface de composição de regras de contexto (ver Seção 3.8).

4.3.1 Simuladores e Controles Remotos

Uma regra criada através da IPGAP foi a regra do "Fogão Esquecido" (ver Seção 3.2). Para que isto ocorresse, foi necessário a disponibilidade dos recursos, cuja resolução foi equivalente a apresentada na Subseção 4.2.4, ou seja, foram criados simuladores para os recursos. Contudo, estes simuladores foram instanciados pelos próprios usuários da IP-

Figura 4.4: SmartLiC - Desligamento da lâmpada

GAP, que podem criá-los ao simplesmente selecioná-los em um menu inicial. A Figura 4.5 apresenta os simuladores para a cama e para o fogão respectivamente.

(a) Simulador de Cama

(b) Simulador do Fogão

Figura 4.5: Simuladores

O controle sobre os recursos ocorre de maneira similar a um controle remoto. Estes controles permitem que o usuário interaja com os recursos em específico. No caso da Figura 4.5(a) o usuário indica que alguém deitou na cama ao clicar sobre ela nesta interface. A regra "Fogão Esquecido", que considera se a pessoa deixou o fogão ligado e foi se deitar, recebe essa notificação, e é atualizada. O mesmo é válido para o controle do fogão (Figura 4.5(b)) que permite ligar e desligar o forno, assim como ligar e desligar quaisquer uma de suas bocas. Os controles também apresentam mudanças nos estados dos ARs, como pode ser visto nas mensagens apresentadas nas figuras. Os controles remotos apresentados na Figura 4.5 também atendem a **Questão 2a** referente à prototipagem rápida, assim como a solução apresentada na Subseção 4.2.4.

Além dos simuladores, os controles remotos também servem para controlar recursos reais. Atualmente o nosso grupo está realizando testes com uma placa de prototipagem chamada BeagleBone [4], que permitirá a construção de protótipos para uma série de dispositivos. Um dos exemplos em construção é o controle remoto de uma lâmpada, representado na Figura 4.6. A placa, de um lado, é capaz de ligar e desligar lâmpadas (OPs) e, de outro lado implementa um AR para a lâmpada controlada, o qual é registrado no AmbI. Uma interface gráfica para o controle da lâmpada é capaz de chamar suas OPs remotamente (i.e., liga e desliga), e também é subscrita ao AR Lâmpada, o que permite ao controle remoto estar ciente do seu contexto.

Figura 4.6: Controle Remoto

4.3.2 Criação e Gerenciamento de Regras de Contexto

Uma vez instanciados os simuladores necessários, e utilizando seus controles remotos, pode-se começar a construir novas regras de contexto. Para isto, os usuários devem mudar o modo de uso da IPGAP. No modo de uso padrão, ao se clicar em um AR, é aberto um possível controle remoto para o mesmo. Por exemplo, ao se clicar na lâmpada, é aberto um controle remoto como o da Figura 4.6.

Ao se abrir o menu de contexto da IPGAP (na região inferior da Figura 3.9), é escolhido a opção "*Compose a Context Rule*". Isto faz com que a IPGAP mude seu modo de uso para o de composição de regras. Um menu lateral esquerdo é aberto, como mostra a figura, e ao se clicar em um AR, ao invés do controle remoto, são exibidas as VCs deste AR (ver Figura 3.10), o que a IPGAP obtém através do SDR. Conforme uma ação é completada (e.g., a escolha de uma VC) é aberto um menu (ver Figura 3.11) que possibilita fazer comparações da VC com valores constantes ou outras VCs. A seguir, estes passos se sucedem, ou seja, o usuário continua definindo comparações para as VCs, inter-relacionamentos das VCs por meio de uma expressão lógica, e definindo temporizadores,

até que clique em finalizar. Como resultado, é instanciado um IC que implementa esta regra.

Em uma segunda etapa, após a criação do IC, o usuário define o que deseja que ocorra quando a regra é avaliada como verdadeira. No caso da regra do "Fogão Esquecido", o objetivo é acordar a pessoa disparando o despertador do quarto e mostrando uma mensagem explicativa na TV. Uma vez que a primeira etapa esteja concluída, a IPGAP muda seu modo de uso mais uma vez, e agora, ao se clicar em um AR são exibidos suas OPs. Desta maneira, o atuador da regra "Fogão Esquecido" é definido ao se escolher a OP "disparar" do despertador, e "mostrar mensagem" da TV. Ao finalizar, um AR atuador é instanciado e subscrito ao IC recém criado, e a interface volta para o seu modo padrão de uso.

Esta solução para a criação de regras em alto nível, possibilita a definição das preferências dos usuários, atendendo a **Questão 2b**.

Como os ICs e atuadores são também ARs e podem ser encontrados através do SDR, é possível controlar o funcionamento dos mesmos por meio de suas OPs, ou seja, os ICs possuem OPs para parar seu funcionamento, retomá-lo ou interrompê-lo definitivamente, como apresentado na Seção 3.8. Para possibilitar este controle em alto nível foi desenvolvida uma interface que permite aos usuários manipular as regras de contexto em execução, como apresentado anteriormente na Figura 3.13. A interface lista os ICs registrados no sistema possibilitando aos usuários desempenhar essas ações de controle ou criar uma nova regra.

Os controles remotos, a interface de composição de regras de contexto, e a interface de controle e regras são três componentes do ferramental que os usuários detêm para gerenciamento das regras no ambiente, o que atende à **Questão 2c**.

4.4 Conclusões do Capítulo

Este capítulo procurou avaliar a proposta do *framework* conceitual e de sua implementação de referência baseado em questões de competência definidas com o propósito de limitar o problema da sensibilidade ao contexto em estudo. Foram elaborados dois conjuntos de questões, onde o primeiro contém como principal alvo os desenvolvedores, e o segundo contém como principal alvo os usuários finais.

As questões definidas em alto nível, compreendem pontos centrais da proposta, que avaliam a flexibilidade para a construção de aplicações ubíquas, os recursos para a construção de regras de contexto, a temporização de regras de contexto, o suporte para prototipagem rápida e testes, e o mecanismo de controle e definição de preferências dos usuários no AmbI. Verificou-se que a proposta atende a estes pontos e como isto é feito.

No próximo capítulo serão estudados trabalhos relacionados e comparados à proposta deste trabalho.

Capítulo 5

Trabalhos Relacionados

Os benefícios para usuários e desenvolvedores que a Computação Ubíqua prevê ultrapassam o horizonte de muitos pesquisadores criativos. As possibilidades que dela advém rompem os limites estabelecidos por interfaces de dispositivos comuns, como, por exemplo, o mouse, o teclado e telas sensíveis ao toque. No entanto, construir tais aplicações adaptáveis requer um grande esforço de desenvolvedores, que se encontram imersos em um universo de especificações de dispositivos e tecnologias de comunicação, além do próprio problema que a aplicações deve resolver. Portanto, muitos pesquisadores focam em diminuir estes obstáculos ao prover abstrações, *middlewares*, serviços, ferramentas e outras técnicas de suporte, não apenas para o desenvolvimento mas também para a prototipagem de aplicações ubíquas.

Neste capítulo são apresentados trabalhos e propostas relevantes de outros autores que lidam com desafios também enfrentados por este trabalho. Dentre os trabalhos selecionados encontram-se *frameworks*, *middlewares* e sistemas para o desenvolvimento de sistemas sensíveis ao contexto e a definição das preferências dos usuários no ambiente.

5.1 Context Toolkit

O *framework* de Dey e Abowd [18], implementado pelo Context Toolkit, é um trabalho seminal para o suporte à prototipagem rápida de aplicações sensíveis ao contexto e provê um conjunto de abstrações para a composição de componentes reusáveis que focam na aquisição de contexto e interpretação. Neste trabalho é proposta uma plataforma de alto nível, que visa abstrair os mecanismo de aquisição de contexto, lidando assim com o desafio da heterogeneidade dos dispositivos através de um componente denominado *widget* de contexto que, em semelhante ao AR, tem como objetivo prover uma separação

de interesses e a reusabilidade do componente. Um componente interpretador de contexto transforma dados brutos em informações de nível mais alto. E um componente agregador reúne informações de diversos *widgets* em mesmo local, facilitando o acesso às informações.

Embora esta abordagem ofereça abstrações para a subscrição às informações de contexto e interpretação das mesmas, não detêm preocupações no nível dos usuários finais, a fim de facilitar a definição de suas preferências. Além disso, não oferece recursos para a prototipagem rápida de aplicações, incluindo o gerenciamento e testes de aplicações em nível de usuário.

5.2 Gaia

Além do Context Toolkit, outro trabalho pioneiro no campo, o *middleware* Gaia [39] foi projetado para facilitar a construção de aplicações para AmbI. Este *middleware* estende conceitos típicos de sistemas operacionais para incluir sensibilidade ao contexto, cujo foco é suportar o desenvolvimento e execução de aplicações para AmbI.

O *middleware* consiste de um conjunto de serviços núcleo e um *framework* para a construção de aplicações sensíveis ao contexto distribuídas. O Gaia abrange muitos desafios, incluindo a aquisição de contexto, a manutenção de descrições de software e hardware, mecanismos para a procura por recursos e para a composição de regras de contexto, dentre outras. Assim como o Context Toolkit, o Gaia se foca no desenvolvimento, não provendo soluções no nível do usuário.

5.3 Gator Tech Smart House

O Gator Tech Smart House [27] foi criado a partir da implementação de uma arquitetura de referência para AmbI, visando a construção de espaços programáveis orientados a serviços, onde profissionais específicos de domínios (e.g., profissionais da saúde) poderão desenvolver e implantar aplicações poderosas para usuários. A proposta contempla um *middleware* onde os dispositivos são manipulados como *bundles* OSGi; uma camada de serviços implementa mecanismos de registro e descoberta, e permite compor novos serviços a partir de outros existentes; uma camada de conhecimento contém uma ontologia dos serviços oferecidos, aplicações e dispositivos, utilizada no registro e descoberta de serviços; e uma camada de gerenciamento de contexto, que permite às aplicações criar e registrar contextos de interesse, e que possui um motor de contexto responsável interpretar

as informações contextuais.

Contudo, não fica claro como ocorre a sensibilidade ao contexto, que envolve não só a interpretação do contexto mas também atuação no ambiente. Além disso, subscrição às informações de contexto também não é um tema abordado, e como desenvolvedores e usuários podem compor novas aplicações.

5.4 MoCA

O MoCA (Mobile Collaboration Architecture) [44] é um *middleware* para desenvolvimento e implantação de aplicações colaborativas sensíveis ao contexto para usuários móveis. Sua arquitetura possui como serviços núcleo: o *monitor*, coleta dados referentes ao dispositivo no qual está executando e informações sobre a rede; o *context information service* que mantém informações compartilhadas pelos *monitors* e implementa um mecanismo de eventos para notificação de mudanças; o *discovery service*, que armazena informações sobre aplicações e serviços; o *configuration service* que armazena e gerencia informações de configurações para todos os dispositivos móveis, possibilitando o uso dos serviços núcleo; e o *location inference service*, que infere a localização aproximada de dispositivos.

Sua implementação engloba um conjunto de APIs para acessar estes serviços, no entanto, não provê suporte para a interpretação de contexto, cabendo tão somente às aplicações definir como isto é feito. É possível se criar regras na forma de "*interest expression*", aos quais a aplicação se subscreve, contudo não é possível a declaração de temporizadores. Também não é provido suporte em nível de usuários finais.

5.5 DIOS++

O DIOS++ [31] é uma infraestrutura para possibilitar adaptação autonômica baseada em regras e controle de aplicações científicas distribuídas. O *framework* proposto define abstrações para aumentar aplicações existentes com sensores e atuadores para questionamento e controle em tempo de execução. Uma rede de controle conecta e gerencia sensores e atuadores distribuídos, e possibilita a descoberta externa, o questionamento, o monitoramento e a manipulação destes objetos em tempo de execução. A partir de um motor de regras distribuído, o DIOS++ possibilita a definição, implantação e execução de regras para gerenciamento autonômico de aplicações em tempo de execução. As regras são

avaliadas em agentes distribuídos, mas controladas por um motor de regras centralizado.

No entanto, não é tratada a questão da temporização das regras de contexto, e a infraestrutura não provê um suporte para a subscrição às informações contextuais. Considerando o suporte aos usuários finais, não é provido um ferramental para criação e gerenciamento de regras em alto nível, e nem um suporte para a prototipagem de regras.

5.6 CASS

Em [37] é proposto um sistema que foca na prototipagem de regras e possui um mecanismo simples que visa descobrir possíveis conflitos entre regras que são compostas utilizando-se exclusivamente cláusulas "E". Sua arquitetura é composta de uma interface gráfica para composição de regras em nível de usuário, e gerenciamento das regras e do ambiente; um gerenciador de regras, capaz de exportar regras em formato XML, importar, e verificar a ocorrência de conflitos; um gerenciador de sensores, que simula de maneira virtual a aquisição de informações contextuais; e um gerenciador de atuadores, que implementa um dispositivo virtual para desempenhar as ações simuladas.

Embora este sistema auxilie na composição e gerenciamento de regras para manipular as informações de contexto, ele não provê suporte para a implantação de regras em um ambiente de execução por não oferecer suporte a dispositivos reais. Em relação à sensibilidade ao contexto, o sistema não permite a construção de regras mais complexas, e também não incorpora abstrações para a interpretação do contexto.

5.7 VisualRDK

Em [53] é proposto uma ferramenta para a prototipagem rápida de aplicações pervasivas, o Visual Robotic Development Kit (VisualRDK). Desenvolvedores criam programas visualmente ao arrastar e soltar componentes e operadores lógicos. Os componentes representam elementos de hardware ou software que possuem uma localização física, e introduzem um conjunto de eventos e comandos (equivalente às VCs e OPs). Novos componentes podem ser ligados através de plugins cujo desenvolvimento é auxiliado pela ferramenta.

O público alvo do VisualRDK consiste de desenvolvedores profissionais assim como aprendizes, contudo não há um foco em usuários finais para a criação de regras ou aplicações, ou gerenciamento das mesmas.

5.8 Motores de Regras

Alguns trabalhos utilizam um motor de regras para avaliar as regras de contexto [6, 13, 52], como por exemplo, o Jess, o Drools, o CLIPS, ou um motor proprietário. Este trabalho não determina que um motor de regras em particular seja utilizando para fins de interpretação de contexto, e o *SmartAndroid* não faz uso de um. Contudo, se for de interesse, um motor poderia ser facilmente agregado e aproveitar a infraestrutura de acesso aos recursos do AmbI.

5.9 Discussão e Conclusão

Neste capítulo foram apresentados trabalhos no estado da arte para sensibilidade ao contexto em AmbI. Foram apresentadas algumas das principais características e arquitetura das propostas, e comparadas com este trabalho. Estas comparações foram feitas com foco nas seguintes características:

1. **Subscrição à Informações de Contexto** O *SmartAndroid* implementa um mecanismo definido no *framework* para subscrição e notificação de informações de contexto. Esta característica possibilita a definição dos ICs e, portanto, facilita a criação de aplicações sensíveis ao contexto.

2. **Abstração Interpretadores de Contexto** Os ICs são entidades que permitem a agregação de informações de contexto em uma entidade única, possibilitando a interpretação desdes dados. Esta abstração permite uma flexibilização na composição de regras de contexto por parte das aplicações.

3. **Regras de Contexto Complexas** Devido ao fato de muitas abordagens utilizarem regras formadas somente com cláusulas "E", esta característica avalia se a abordagem possibilita a construção de regras mais complexas, ou seja, utilizando outros operadores lógicos e parênteses. Isso permite não só aumentar a expressividade da regra, como também a diminuição de regras no ambiente, favorecendo o controle no nível dos usuários e aplicações.

4. **Regras de Contexto Temporizadas** A temporização de regras de contexto é um recurso fundamental na construção das mesmas, visto que a dimensão tempo possibilita a construção de regras mais sofisticadas. Enquanto nossa proposta possibilita não só a definição de um temporizador na regras, mas também de outros, poucas

propostas sequer abordam este tema, e muitas deixam este tema para ser resolvido no nível da aplicação.

5. **Desacoplamento entre Interpretadores e Atuadores** A separação entre a interpretação e a atuação possibilita uma reutilização dos componentes, por exemplo, se foi construído um atuador complexo para desempenhar uma tarefa, este poderia ser ligado a outro interpretador, assim como um interpretador construído pode ser futuramente ligado a outro atuador que não foi concebido durante a concepção do primeiro. Isto difere de outras propostas que definem a atuação de maneira intrínseca à interpretação.

6. **Prototipagem de Regras de Contexto** A prototipagem de regras de contexto está relacionada à criação de regras de contexto que funcionam em um ambiente simulado, ou seja, que referencia recursos simulados. Esta característica favorece a prototipagem de novas regras de contexto de maneira rápida e atende tanto a desenvolvedores como a usuários.

7. **Criação de Regras em Nível de Usuário** Esta característica diz respeito a capacidade da solução de prover mecanismos para a criação de regras em alto nível, possibilitando que usuários sem experiência técnica sejam capazes de criar novas regras de contexto e que desenvolvedores possam prototipar regras facilmente para em seguida utilizá-las na aplicação em desenvolvimento.

8. **Gerenciamento de Regras de Contexto** O gerenciamento de regras de contexto possibilita o controle em nível de usuário das regras de contexto em execução, possibilitando a interrupção de seu funcionamento e a retomada. Isto aumenta o poder de controle dos usuários, possibilitando que regras indesejadas sejam interrompidas.

Estas características foram consideradas na comparação com os trabalhos relacionados e são diretamente mapeadas para a Tabela 5.1 onde os trabalhos relacionados foram classificados com com três possibilidades para a presença da característica:

1. **S** – Indica que o trabalho possui esta característica.

2. **N** – Indica que o trabalho não possui esta característica, ou não deixa claro que possui.

3. **D** – Indica que o trabalho discute o uso dessa característica, mas não traz maiores detalhes de projeto ou não deixa claro como a utiliza, como a implementa ou implementaria.

Na Tabela 5.1, a proposta conceitual e sua implementação foram incluídos no item *SmartAndroid* a título de simplificação. Como pode ser visualizado na tabela, as diferentes abordagens proveem soluções para diferentes características dentre as listadas, que foram selecionadas com base nos principais aspectos da proposta. Estas características poderiam ser separadas em dois grupos, onde no primeiro estão as relacionadas ao *framework* e sua implementação (itens um até o cinco), e no segundo as relacionadas à IPGAP e ao suporte ao usuário (itens seis até o oito). Dentre as propostas estudadas foram excluídos os *frameworks* e *middlewares* que lidam com a heterogeneidade dos recursos (atendendo ao desafio (i)) mas que não oferecem um suporte a sensibilidade ao contexto (ou não deixavam isto claro), assim como os trabalhos de suporte ao usuário (que atendem parcialmente ao desafio (iii)) mas que também não possuem soluções neste sentido.

As comparações apresentadas demonstraram que as propostas são pouco abrangentes, não provendo um suporte tanto no nível dos desenvolvedores como dos usuários. Portanto, foi questionada a versatilidade das propostas em prover não só abstrações para sensibilidade ao contexto, mas também recursos para a prototipagem rápida de novas regras de contexto e um controle do ambiente em alto nível.

Tabela 5.1: Resumo entre os trabalhos relacionados

Solução	Características							
	Subscrição Contexto	**Abstração IC**	**Regras complexas**	**Regras temporizadas**	**Desacopl. IC-Atuador**	**Prototipagem de regras**	**Criação de regras**	**Gerencia de regras**
Context Toolkit	S	S	S	N	S	N	N	N
Gaia	S	S	S	N	S	N	N	N
Gator Tech	D	N	N	N	N	N	N	N
MOCA	S	N	S	N	N	N	N	N
DIOS++	N	N	S	N	N	N	N	N
CASS	N	N	N	N	N	S	S	S
VisualRDK	S	N	S	S	N	S	N	N
SmartAndroid	S	S	S	S	S	S	S	S

Capítulo 6

Conclusão

Esta dissertação teve início a partir de ideias relacionadas ao projeto SCIADS, desenvolvido no Laboratório Tempo[1]. O SCIADS utiliza conceitos de Computação Ubíqua para prover serviços de saúde a pacientes em ambiente domiciliar [42]. Como resultado foi desenvolvido um sistema sensível ao contexto que coleta informações fisiológicas do paciente (e.g., pressão arterial, frequência cardíaca, peso, atividade física) e detecta sua situação de saúde (i.e., normal, alerta ou emergência), estas informações são então enviadas à Central de Saúde Remota que as apresenta aos profissionais de saúde, e que também emite notificações em caso de situações de emergência, possibilitando que sejam tomadas providências em relação ao paciente [11]. Dando sequência a este projeto, identificamos a necessidade de um suporte para a construção de aplicações ubíquas, não apenas para se promover serviços de saúde, mas serviços diversos como segurança, entretenimento, acessibilidade, economia e outros.

Para atender a este objetivo foi concebido um *framework* que utiliza conceitos bem estabelecidos de Sistemas Distribuídos com o intuito de propor abstrações para a aquisição de contexto e atuação no ambiente. A definição dos Agentes de Recurso e suas Variáveis de Contexto e Operações desempenham este papel, e um conjunto de serviços básicos permite o registro, descoberta e localização física destes Agentes de Recurso. Isso possibilitou atender ao primeiro desafio, assumido por Mareli (2013) [33], que corresponde a *heterogeneidade dos recursos* no ambiente.

O segundo desafio é um dos focos desta dissertação e está relacionado à quantidade e variedade das informações de contexto presente em um ambiente. Este desafio foi atendido com a definição dos Interpretadores de Contexto, que utilizam o suporte a eventos das Variáveis de Contexto para criar regras de contexto personalizadas. Os Interpretadores

[1] www.tempo.uff.br

de Contexto são responsáveis por agregar informações contextuais, provendo uma informação em nível mais alto. Desta maneira, é possível, através do *framework*, se construir aplicações sensíveis ao contexto diminuindo sua complexidade.

A partir do *framework* conceitual foi criada uma plataforma, o *SmartAndroid*, com o objetivo de validar os conceitos propostos e viabilizar a implementação de aplicações exemplo. O *SmartAndroid* contempla as principais características do *framework*, e disponibiliza para os desenvolvedores uma API e um conjunto de serviços básicos, que possibilitam a construção de um ambiente inteligente e de aplicações para este ambiente. O suporte provido pelo *SmartAndroid* facilita a criação de regras de contexto, que podem ser definidas separadamente em arquivos JSON e um *parser* se encarrega de, a partir destas definições, instanciar Interpretadores de Contexto e Atuadores que implementem a regra de contexto definida.

Sobre o *SmartAndroid* foi proposta por Barreto (2013) [3] uma interface que facilita a prototipagem e o gerenciamento de aplicações ubíquas, e interage tanto com recursos reais quanto virtuais, a IPGAP. Foi agregado à IPGAP uma interface para a criação e o gerenciamento de regras de contexto, possibilitando a personalização e o controle do ambiente por usuários sem experiência técnica. Esta característica corresponde ao segundo foco desta dissertação e atende ao terceiro desafio, relacionado à *disponibilidade de recursos* para a criação de novas aplicações e composição de regras.

6.1 Trabalhos Futuros

O *framework* proposto possui potencial para atender a outros desafios relevantes na área de Computação Ubíqua. Como trabalhos futuros, incluem-se a implementação completa do *framework* proposto, a construção de outras aplicações que explorem a capacidade de sensibilidade ao contexto utilizando inclusive de recursos de aprendizagem, dentre outras.

A utilização de mecanismo de inteligência artificial, como a mineração de dados em históricos do uso (*log*) e aprendizagem de máquina, permitirá expandir o uso de sensibilidade ao contexto do *framework* ao possibilitar a adaptação de aplicações diante de um contexto inferido. Atualmente a criação de regras de contexto ocorre através da declaração explícita por aplicações ou pessoas. Futuramente será possível, através de técnicas de aprendizagem de máquina, se inferir contextos de mais alto nível, tais como o humor do usuário. No contexto da aplicação ubíqua *SmartLiC* (Seção 4.2) um dos trabalhos

futuros é implementar um módulo de segurança domiciliar que aprende sobre o padrão de movimentação das pessoas para, posteriormente, simular a presença de pessoas na casa e evitar assim possíveis furtos. Além disso, é possível também que uma aplicação sugira novas regras de contexto que favoreçam o uso dos usuários ou a economia de recursos.

Em relação ao suporte para a definição de regras de contexto do *SmartAndroid*, visamos incrementar o *parser* para interpretar outros tipos de operações. Isto possibilitará o aumento da expressividade das regras de contexto. Dentre estes operandos, consideramos o "existe" ($\exists$) e o "para todo" ($\forall$), encontrados na lógica de segunda ordem.

A multiplicação de regras de contexto no ambiente inteligente, criadas por diferentes usuários e aplicações, pode levar a casos de conflitos entre regras. Este conflito pode ser tanto em relação ao interesse do usuário, que ocorre quando uma regra age de maneira inversa ao que o usuário espera (e.g., ligar a lâmpada do quarto, trocar o canal da televisão), quanto em relação a regras contraditórias, por exemplo:

- Se fogão ligado e temperatura $> 140^{o}$ C por 15 min, então reduza a temperatura
- Se fogão ligado e temperatura $< 160^{o}$ C por 15 min, então aumente a temperatura

Então, considerando que essas regras aconteceram ao mesmo tempo, um conflito poderia ser detectado, pois não está claro como o sistema deve agir. Contudo, se uma regra ocorre logo depois da outra (e.g., um segundo depois), há um outro tipo de problema, pois neste caso há um intervalo de tempo a se considerar, que pode inclusive variar dependendo de interesses das pessoas ou do tipo de dispositivo, e isto leva a um outro nível de dificuldade para se lidar com a detecção e resolução de conflitos.

A interface de criação de regras de contexto é também um importante ponto de evolução do trabalho, visto que através desta interface usuários sem experiência técnica são capazes de não só criar novas regras de maneira simples, mas também de controlar o ambiente ao desativar a modificar regras em execução. Esta funcionalidade diminui os possíveis conflitos de interesses entre os usuários e as aplicações ou regras em funcionamento. Em uma implementação futura pretendemos utilizar menos texto e mais símbolos na composição da regra (para escolha dos operandos, parênteses e os outros recursos). É necessário que haja não só o aprimoramento da interface, mas também um estudo de usabilidade que conduza ao desenvolvimento centrado no usuário de uma interface mais amigável e intuitiva. Além disso, um outro tópico de interesse é tornar esta interface acessível a usuários que apresentem deficiências físicas, como por exemplo a cegueira (parcial ou integral).

Outra proposta para a interface de criação de regras é identificar um padrão em uma regra sendo criada que é utilizado em outro IC e oferecer ao usuário a ligação ao IC, ao invés da ligação à todas as VCs. Com isso, o usuário se beneficiará da não repetição de regras, o que além de diminuir o tráfego de mensagens também diminui a complexidade da regra e a manutenção.

Dentre as melhorias gerais a serem implementadas no *framework* inclui-se a definição de um modelo de Linhas de Produto. Este modelo deve contemplar mudanças arquiteturais que permitam especificar as variabilidades do produto (i.e., as diferenças entre produtos de uma mesma linha), e definir assim uma família de aplicações, o que torna possível a agregação de novos módulos à aplicação gerando um novo produto. O metamodelo proposto em [10] possibilita descrever as configurações das características de cada aplicação a partir de contratos arquiteturais e, com o modelo estabelecido, estas configurações podem ser implantadas no ambiente gerando uma nova aplicação ou produto.

Um domínio da Computação Ubíqua de particular interesse é a área de saúde. Neste contexto, um dos trabalhos futuros é adaptar o sistema construído no projeto SCIADS [11] para utilizar o *SmartAndroid*. Uma das vantagens de fazer esta portabilidade é a facilidade de agregar novos dispositivos médicos e módulos de avaliação ao sistema. Além disso, o suporte à regras de contexto pode ser usado na definição do plano de cuidados do paciente, que corresponde às atividades prescritas por profissionais de saúde que o paciente deve realizar (e.g., tomar medicamento, fazer exercício, aferir a pressão arterial). O plano de cuidados, em geral, requer que o paciente informe se realizou a tarefa. Em um ambiente inteligente é possível descobrir se o paciente realizou a tarefa de forma autônoma, evitando que o paciente esqueça ou que faça duas vezes (como é comum em pacientes com Alzheimer, por exemplo), além de aproximar a família do tratamento.

Referências

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

[2] Baldauf, M.; Dustdar, S.; Rosenberg, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing 2*, 4 (2007), 263–277.

[3] Barreto, D. Interface de prototipagem e gerenciamento de aplicações pervasivas. Dissertação de mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, 2013.

[4] BeagleBoard. Open Hardware Physical Computing on ARM and Linux. http://beagleboard.org/, acessado em fevereiro de 2014.

[5] Bezerra, L. N. *Uso de ontologia em serviço de contexto e descoberta de recursos para autoadaptação de sistemas.* Dissertação de mestrado, Programa de Pós Graduação em Engenharia Eletrônica (PEL) – Universidade do Estado do Rio de Janeiro, 2011.

[6] Biegel, G.; Cahill, V. A framework for developing mobile, context-aware applications. In *Pervasive Computing and Communications.* (2004), IEEE, pp. 361–365.

[7] Birrell, A.; Nelson, B. Implementing remote procedure calls. *ACM Trans. Comput. Syst. 2*, 1 (1984), 39–59.

[8] Brush, A.; Lee, B.; Mahajan, R.; Agarwal, S.; Saroiu, S.; Dixon, C. Home automation in the wild: challenges and opportunities. In *Human factors in computing systems* (2011), ACM, pp. 2115–2124.

[9] Cardoso, L. Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software. Dissertação de mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, 2006.

[10] Carvalho, S. T. *Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas.* Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2013.

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

[12] Chen, G.; Kotz, D., et al. A survey of context-aware mobile computing research. Tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[13] Chen, Y.S. and Chen, I.C. and Chang, W. Context-aware services based on OSGi for smart homes. *Ubi-media Computing (U-Media) 11* (2010), 392.

[14] CLIPS. A Tool for Building Expert Systems. clipsrules.sourceforge.net/, acessado em fevereiro de 2014.

[15] Copetti, A. *Monitoramento Inteligente e Sensível ao Contexto na Assistência Domiciliar Telemonitorada.* Tese de doutorado, Instituto de Computação – Universidade Federal Fluminense, 2010.

[16] Coulouris, G.; Dollimore, J.; Kindberg, T. *Distributed systems: concepts and design.* Addison-Wesley Longman, 2005.

[17] de Araujo, R. Computação ubíqua: Princípios, tecnologias e desafios. In *XXI Simpósio Brasileiro de Redes de Computadores* (2003), vol. 8, pp. 11–13.

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

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

[20] Drools. The Business Logic integration Platform. www.jboss.org/drools/, acessado em fevereiro de 2014.

[21] Edwards, W. K.; Grinter, R. E. At home with ubiquitous computing: seven challenges. In *Ubicomp 2001: Ubiquitous Computing* (2001), Springer, pp. 256–272.

[22] Erthal, M.; Mareli, D.; Ferreira, D. B.; Loques, O. Interpretação de Contexto em Ambientes Inteligentes. In *SBCUP 2013* (julho 2013).

[23] Ferreira, D. B.; Erthal, M.; Mareli, D.; Loques, O. Uma interface de prototipagem para aplicações pervasivas. In *SBRC 2013* (maio 2013).

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

[25] Friedman-Hill, E., et al. Jess, the rule engine for the java platform, 2003.

[26] Grüninger, M.; Fox, M. S. Methodology for the design and evaluation of ontologies.

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

[28] JEVAL. http://jeval.sourceforge.net/, acessado em fevereiro de 2014.

[29] JSON. The Java Script Object Notation. http://www.json.org/, acessado em fevereiro de 2014.

[30] Kindberg, T.; Fox, A. System software for ubiquitous computing. *Pervasive Computing, IEEE 1*, 1 (2002), 70–81.

[31] Liu, H.; Parashar, M. DIOS++: A framework for rule-based autonomic management of distributed scientific applications. *Euro-Par 2003 Parallel Processing* (2003), 66–73.

[32] Loureiro, A. A. F.; Oliveira, R. A. R.; Silva, T.; Júnior, W. R. P.; Oliveira, L.; Moreira, R.; Siqueira, R.; Rocha, B.; Ruiz, L. Computação ubíqua ciente de contexto: Desafios e tendências. *Simpósio Brasileiro De Redes De Computadores E Sistemas Distribuídos, Recife 27* (2009), 99–149.

[33] Mareli, D. Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes. Dissertação de mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, 2013.

[34] Mareli, D.; Erthal, M.; Ferreira, D. B.; Loques, O. Um framework de desenvolvimento de aplicações ubíquas em ambientes inteligentes. In *SBRC 2013* (maio 2013).

[35] OMG – The Object Management Group. CORBA – Common Object Request Broker Architecture. http://www.omg.org/spec/, acessado em fevereiro de 2014.

[36] Oracle. Java Remote Method Invocation. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, acessado em fevereiro de 2014.

[37] Park, J.; Moon, M.; Hwang, S.; Yeom, K. CASS: A Context-Aware Simulation System for Smart Home. In *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)* (2007), IEEE, pp. 461–467.

[38] Pascoe, J. Adding generic contextual capabilities to wearable computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on* (1998), IEEE, pp. 92–99.

[39] Ranganathan, A.; Campbell, R. A middleware for context-aware agents in ubiquitous computing environments. In *International Conference on Middleware* (2003), Springer-Verlag New York., pp. 143–161.

[40] Ranganathan, A.; Chetan, S.; Al-Muhtadi, J.; Campbell, R.; Mickunas, M. Olympus: A high-level programming model for pervasive computing environments. In *Pervasive Computing and Communications, 2005.* (2005), IEEE, pp. 7–16.

[41] Rodrigues, A. L. B. *Uma infra-estrutura para monitoramento de sistemas cientes do contexto.* Dissertação de mestrado, Programa de Pós Graduação em Engenharia Eletrônica (PEL) – Universidade do Estado do Rio de Janeiro, 2009.

[42] Rodrigues, A. L. B.; Gomes, I. C.; Bezerra, L. N.; Sztajnberg, A.; Carvalho, S. T.; Copetti, A.; Loques, O. Using Discovery and Monitoring Services to Support Context-Aware Remote Assisted Living Applications. In *2009 International Conference on Computational Science and Engineering* (2009), IEEE, pp. 1092–1097.

[43] Román, M.; Hess, C.; Cerqueira, R.; Ranganathan, A.; Campbell, R. H.; Nahrstedt, K. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE 1*, 4 (2002), 74–83.

[44] Sacramento, V.; Endler, M.; Rubinsztejn, H. K.; Lima, L. S.; Goncalves, K.; Nascimento, F. N.; Bueno, G. A. Moca: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online, IEEE 5*, 10 (2004), 2–2.

[45] Satyanarayanan, M. Pervasive computing: Vision and challenges. *Personal Communications, IEEE 8*, 4 (2001), 10–17.

[46] Schilit, B.; Adams, N.; Want, R. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994.* (1994), IEEE, pp. 85–90.

[47] Schmidt, A.; Aidoo, K. A.; Takaluoma, A.; Tuomela, U.; Van Laerhoven, K.; Van de Velde, W. Advanced interaction in context. In *Handheld and ubiquitous computing* (1999), Springer, pp. 89–101.

[48] Tang, L.; Yu, Z.; Zhou, X.; Wang, H.; Becker, C. Supporting rapid design and evaluation of pervasive applications: challenges and solutions. *Personal and Ubiquitous Computing 15*, 3 (2010), 253–269.

[49] Uschold, M.; Gruninger, M., et al. Ontologies: Principles, methods and applications. *Knowledge engineering review 11*, 2 (1996), 93–136.

[50] W3C – World Wide Web Consortium. Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis, acessado em fevereiro de 2014.

[51] W3C – World Wide Web Consortium. XML Essentials – W3C. http://www.w3.org/standards/xml/core, acessado em fevereiro de 2014.

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

[53] Weis, T.; Knoll, M.; Ulbrich, A.; Muhl, G.; Brandle, A. Rapid prototyping for pervasive applications. *Pervasive Computing 6*, 2 (2007), 76–84.

[54] Weiser, M. The Computer for the 21st Century. *Scientific American 3* (1991), 94–104.

[55] Weiser, M.; Brown, J. S. The coming age of calm technology. In *Beyond calculation*. Springer, 1997, pp. 75–85.

APÊNDICE A – VARIAÇÕES SOBRE EXEMPLO DE REGRA

Neste apêndice são apresentadas algumas possibilidades de composição de regras utilizando os recursos para definição de ICs do *SmartAndroid*. As regras 1 à 9 são variações da regra Fogão Esquecido evidenciando diferentes possibilidades de composição de regras de contexto através do *framework*.

1. SE (fogão ligado E alguém está deitado na cama) POR 20 min
   ENTÃO desligar fogão E mostrar mensagem na TV da sala
   Comentário: esta é uma regra simples e que usa temporizador, e atua desligando o fogão e mostrando uma mensagem na TV da sala

2. SE (fogão ligado E (alguém está deitado na cama de casal OU alguém está deitado na cama de solteiro) POR 20 min
   ENTÃO desligar fogão E mostrar mensagem na TV da sala
   Comentário: esta é uma regra mais complexa do que a primeira por utilizar uma subexpressão e também usa um temporizador

3. SE (fogão ligado E Tania está deitada) POR 20 min
   ENTÃO desligar fogão E disparar despertador
   Comentário: esta regra consegue referenciar pessoas especificamente

4. SE (fogão ligado E não tem ninguém na cozinha) POR 20 min
   ENTÃO desligar fogão E avisar pessoa com mensagem na TV
   Comentário: esta regra consegue afirmar que, para qualquer pessoa, nenhuma está na cozinha

5. SE (fogao ligado E não tem ninguém na cozinha) POR 20 min
   ENTÃO desligar fogão E avisar pessoa com mensagem na TV
   Comentário: regra que consegue afirmar que, para qualquer pessoa, nenhuma está na cozinha

6. SE (fogão ligado POR 40 min) E (Tania está deitada POR 20 min)
   ENTÃO desligar fogão E disparar despertador
   Comentário: objetivo de testar a utilização de temporizadores mais internamente na regra
   Comentário: "quero ser lembrado que a comida já cozinhou se eu estiver dormindo", onde dormindo aqui significa estar deitado por 20 min ou mais

7. SE (fogão ligado E Tania está deitada) POR 20 min
   ENTÃO desligar fogão E disparar despertador E mostrar mensagem na TV do cômodo onde Tania está deitada
   Comentário: a complexidade desta regra está na atuação, pois a obtenção do AR da TV é dinâmico, dependendo, no caso, de onde Tania deitou

8. SE (fogão ligado E Tania está deitada no quarto) POR 20 min
   ENTÃO desligar fogão E disparar despertador
   Comentário: esta regra está exigindo 2 informações no mesmo predicado, que é "Tania está deitada no quarto". Esta sententença poderia ser quebrada em "Tania está deitada E Tania está no quarto"

9. (SE alguém saiu de um cômodo E a luz deste cômodo está acesa) POR 5 min
   ENTÃO: desligar a luz deste cômodo
   Comentário: esta regra pode ser implementada como a seguir: "SE o cômodo C está vazio E a luz do cômodo C está acesa
   ENTÃO desligar a luz do cômodo C"
   esta regra é instanciada para cada cômodo da casa