UFF

DOUGLAS FARIA MOREIRA MARELI

UM FRAMEWORK DE DESENVOLVIMENTO DE APLICAÇÕES UBÍQUAS EM AMBIENTES INTELIGENTES

NITERÓI

2013

UFF

DOUGLAS FARIA MOREIRA MARELI

UM FRAMEWORK DE DESENVOLVIMENTO DE APLICAÇÕES UBÍQUAS 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

NITERÓI

2013

DOUGLAS FARIA MOREIRA MARELI

UM FRAMEWORK DE DESENVOLVIMENTO DE APLICAÇÕES UBÍQUAS 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 Agosto de 2013.

BANCA EXAMINADORA

Prof. Orlando Gomes Loques Filho - Orientador, IC-UFF

Prof. Alexandre Sztajnberg, DICC-UERJ

Prof. José Viterbo Filho, IC-UFF

Niterói

2013

À minha família e amigos, grandes apoiadores.

Agradecimentos

Agradeço aos Professores da UFF, que me forneceram o conhecimento necessário e me motivaram durante o mestrado. Principalmente ao Professor e Orientador Orlando Loques que com muita sabedoria traçou os rumos de um projeto ambicioso e promissor.

Aos colegas do Laboratório Tempo que me ajudaram quando as dificuldades apareceram. Especialmente aos colegas Matheus Erthal e David Barreto que foram parceiros de projeto.

À CAPES por incentivar meus estudos e pesquisas através da bolsa de Mestrado.

Ao Jorge Luiz dos Passos Mareli, meu pai, o qual me inspira a ser um homem melhor.

À Cassia Faria Moreira, minha mãe, a qual se empenhou em me garantir uma educação qualidade nas melhores instituições possíveis.

Aos meus amigos que sempre confiaram e torceram pelo meu sucesso.

Resumo

A computação móvel e as tecnologias de comunicação sem fio vêm avançando em direção a propostas práticas e simples na área de computação ubíqua e pervasiva. Contemplando este cenário, o trabalho propõe um *framework* para prototipagem e execução em um ambiente permeado por inteligência computacional. Agregam-se a esta proposta abstrações que possibilitam o desenvolvimento de sistemas ubíquos por desenvolvedores com conhecimentos básicos de programação. Visando demonstrar a viabilidade da proposta, os conceitos foram concretizados em uma plataforma denominada *SmartAndroid*. Sobre esta plataforma foram desenvolvidas aplicações sofisticadas que envolvem diversidade de recursos, sensibilidade ao contexto, questões de segurança e mobilidade no espaço físico.

O *framework* inclui um modelo de componentes distribuídos e mecanismos de comunicação concebidos para facilitar a emissão de comandos e a consulta de status de entidades do ambiente. Em particular, a comunicação baseada em eventos é utilizada para manter entidades atualizadas sobre mudanças em informações de contexto pelas quais estão interessadas. As entidades e suas respectivas localizações são mapeadas dinamicamente em uma interface gráfica, representativa do ambiente (Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas). Além da prototipagem de aplicações, essa interface permite o gerenciamento e a configuração de serviços no conjunto de entidades virtuais e reais nela mapeadas.

A proposta tem como diferencial possibilitar que novas aplicações ubíquas possam ser concebidas incrementalmente e gerenciadas de forma dinâmica e adaptativa, disponibilizando conceitos e mecanismos essenciais para se configurar Ambientes Inteligentes personalizados em alto nível. Como benefício adicional, incentiva a separação de interesses, que distribui a carga de codificação de requisitos típicos (*e.g.*, comunicação, segurança, adaptação) dessa classe de aplicações.

Palavras-chave: computação ubíqua, *middleware* para computação ubíqua, ambientes inteligentes, sensibilidade ao contexto.

Abstract

Mobile computing and wireless communication technologies are advancing toward the simple and practical proposals in the area of pervasive and ubiquitous computing. Contemplating this scenario, the paper proposes a framework for prototyping and implementation in an environment permeated by computational intelligence. Additionaly to this proposed abstractions that enable the development of ubiquitous systems for developers with basic programming knowledge. In order to demonstrate the feasibility of the proposal, the concepts were implemented in a platform called SmartAndroid. About this platform were developed sophisticated applications involving resource diversity, context awareness, safety and mobility in physical space.

The framework includes a model of distributed components and communication mechanisms designed to facilitate the issuance of commands and status query entities environment. In particular, event-based communication is used to keep entities updated about changes in context information for which they are interested. Entities and their respective locations are dynamically mapped into a graphical interface, representing the environment (Interface Management and Prototyping pervasive applications). Besides prototyping applications, this interface allows configuration and management services in the set of virtual entities and real it mapped.

The proposal is to allow new differential ubiquitous applications can be designed and managed incrementally and dynamically adaptive, providing essential concepts and mechanisms to configure custom Intelligent Environments at a high level. As an additional benefit, promotes separation of concerns, which distributes the load encoding typical requirements (eg, communication, security, adaptation) of this class of applications.

Keywords: ubiquitous computing, middleware, smart spaces, context awareness.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AC	:	Atuador de Contexto;
AmbI	:	Ambiente Inteligente;
API	:	Application Programming Interface;
AR	:	Agente de Recurso;
AUbi	:	Aplicação Ubíqua;
BSN	:	*Body Sensor Network*;
CAR	:	Classe de Agentes de Recursos;
DAO	:	*Data Access Object*;
DNS	:	Domain Name System;
ECA	:	Evento-Condição-Ação;
GUI	:	*Graphical User Interface*;
IAR	:	Instância de Agente de Recurso;
IC	:	Interpretador de Contexto;
IPGAP	:	Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas;
JSON	:	JavaScript Object Notation;
MCoD	:	Modelo de Componentes Distribuídos;
MER	:	Modelo de Entidade e Relacionamento;
ODAR	:	Objeto de Dado do Agente de Recurso;
OP	:	OPeração de contexto;
POO	:	Programação Orientada a Objetos;
QoS	:	*Quality of Service*;
RANS	:	Resource Agent Name System;
RPC	:	*Remote Procedure Call*;

SBCUP	:	Simpósio Brasileiro de Computação Ubíqua;
SBRC	:	Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos;
SCIADS	:	Sistema Computacional Inteligente de Assistência Domiciliar à Saúde;
SDR	:	Serviço de Descoberta de Recursos;
SegRes	:	aplicação de Segurança Residencial;
SGAR	:	Suporte ao Gerenciamento de Agentes de Recursos;
SGBD	:	Sistema Gerenciador de Banco de Dados;
SGS	:	Serviço de Gerenciamento de Segurança;
SLR	:	Serviço de Localização de Recursos;
SmartLiC	:	*Smart Light Controler*;
SoA	:	*Service oriented Architecture*;
SRR	:	Serviço de Registro de Recursos;
VC	:	Variável de Contexto;

Capítulo 1

Introdução

Na década de 90, Weiser [33] propôs o conceito de Computação Ubíqua (ou Pervasiva, como é chamada equivalentemente em outras publicações), a qual descrevia uma mudança no paradigma de interação entre o usuário e os sistemas computacionais. Weiser previu o surgimento da computação invisível, onde a interação entre os usuários e os computadores ocorre de forma natural, sem a percepção explícita da atuação de sistemas. Os sistemas inicialmente lidavam com a relação de vários usuários para uma máquina (e.g, mainframe); com a popularização dos PCs, a relação passou a ser de um para um; e ultimamente, com *smartphones* e *tablets* e uma grande variedade de dispositivos com capacidade de comunicação ganhando espaço, percebe-se uma relação de vários dispositivos computacionais servindo a um usuário ou um pequeno grupo. Ao longo do tempo, houve um aumento na quantidade e diversidade de dispositivos. Este cenário de disponibilidade crescente motivou avanços tecnológicos que viabilizam propostas concretas na área de computação ubíqua.

Um ambiente inteligente [3] é caracterizado por esta alta disponibilidade de recursos computacionais com capacidade de comunicação, ideal para a construção de aplicações que utilizam conceitos de computação ubíqua. Uma aplicação ubíqua utiliza estes recursos para identificar e atender as necessidades de seus usuários. Para as identificar, coleta as informações do seu contexto de execução por meio de sensores; e para as atender, provê serviços por meio de atuadores, os quais incluem diversos tipos de interfaces.

A construção e a manipulação de aplicações ubíquas representam grandes desafios tanto pra desenvolvedores quanto para usuários. Para desenvolvedores, é especialmente importante o conhecimento técnico especializado e ter a disposição de dispositivos reais durante o desenvolvimento da aplicação. Esta necessidade de conhecimento e recursos ocasiona o encarecimento do desenvolvimento. Para usuários, é necessário ter o acesso

facilitado aos dispositivos do ambiente, e a capacidade de configurar e aplicar suas preferências. Esta demanda deve ser atendida dinamicamente, o que dificulta o estabelecimento de um comportamento padronizado no sistema ubíquo. Alguns desses desafios podem ser assim destacados: (i) há dificuldades em se estabelecer um protocolo comum de comunicação entre os componentes do sistema distribuído, por conta da *heterogeneidade dos dispositivos envolvidos*; (ii) a interatividade das aplicações ubíquas é dificultada dependendo da quantidade e da *variedade de informações de contexto e serviços* disponíveis no ambiente; (iii) o desenvolvimento e o teste de aplicações exigem uma alta *disponibilidade de recursos*, como por exemplo, sensores (*e.g.*, presença, iluminação, temperatura), atuadores (*e.g.*, chaves, alarmes, *smart-tvs*), incluindo novos dispositivos embarcados, ou ainda de espaços físicos, tais como uma casa para aplicações do tipo *smart home*.

Esta dissertação aborda o desafio (i), se concentrando no suporte a nível de *middleware*. Este trabalho faz parte de um projeto, realizado pela equipe do Laboratório Tempo [1], que tem por objetivo estabelecer um *framework* conceitual de construção, depuração, teste e implantação de sistemas ubíquos para ambientes inteligentes resolvendo os três desafios principais apresentados. O trabalho de [14], que aborda o desafio (ii), explora a sensibilidade ao contexto provida pelo *framework*. O trabalho de [16], trata a questão de prototipagem de aplicações ubíquas, que procura solucionar o desafio exposto em (iii).

Em (i) é abordada uma questão que envolve principalmente a parte de *middleware*, camada intermediária entre a camada física (compreendida pelos sensores e atuadores) e a camada de aplicação (onde se encontram o ambiente de desenvolvimento e as aplicações ubíquas). Nesta camada lida-se com alguns dos principais desafios relacionados com a computação ubíqua, como pode ser verificado em [11], dentre estes se destacam: a diversidade de dispositivos, a descentralização dos serviços e questões de conectividade. Lidar com a diversidade de tipos de dispositivos é um dos desafios centrais desta proposta, onde a resolução da questão é obtida por meio da descentralização dos serviços, utilizando a arquitetura orientada a serviços (SoA). O uso de SoA permite o relacionamento entre diferentes entidades, onde quanto maior a descentralização mais flexível é a conectividade, já que a dependência de um serviço central se minimiza. Logo, problemas em um componente do sistema são suavizados e podem ser absorvidos por mecanismos de tolerância a falhas e persistência. Outros desafio abordado é a definição de suporte ao serviço de sensibilidade ao contexto proposto em [14] e à prototipagem de aplicações ubíquas descrita em [16].

[1] www.tempo.uff.br

Muitos trabalhos focados no *middleware* buscam atingir o objetivo de definir *frameworks* voltados à construção e ao gerenciamento de aplicações ubíquas [19, 8, 27, 26]. Em [19], propõe-se um *framework* estruturado em camadas, onde são definidas, paralelamente ao *middleware*, as camadas de conhecimento e de contexto. A primeira concentra soluções de inteligência artificial e a segunda envolve serviços de sensibilidade ao contexto. Em [8] são propostos serviços para gerenciar componentes representativos do ambiente. Nosso trabalho aborda o conceito de Ambientes Inteligentes (AmbI), onde uma variedade de dispositivos está disponível, como por exemplo, em casas inteligentes (ou *smart homes*), com televisores, termômetros, *smartphones*, e outros, os quais podem ser descobertos e configurados de acordo com suas especificidades. Os trabalhos [27, 26] se inserem no âmbito do projeto Gaia, o qual se preocupa em organizar esses componentes de forma a facilitar suas manipulações; a estruturação proposta permite prover flexibilidade no conjunto de operações de suporte de um sistema ubíquo, isto sem exigir a adoção de uma ontologia especifica. Os trabalhos mencionados, no entanto, não têm como foco a integração das questões acima identificadas: *heterogeneidade dos dispositivos*, *variedade de informações de contexto e serviços*, e *disponibilidade de recursos*.

Outras propostas como [7, 2, 30, 29, 24] focam em características específicas encontradas em AmbIs. Trabalhos como [7] focam no teste de aplicações ubíquas em ambiente simulado. Em [2] são tratadas questões de localização, a proposta consiste em um *framework* para promover uma interação entre objetos e pessoas de um espaço. Em [30], preocupa-se com a descentralização dos serviços e propõe-se um *framework* para o desenvolvimento de serviços com objetos distribuídos. Em [29], lida-se com a variedade de informações em um sistema de aplicativos Android. Com esse fim se propõe um sistema de tratamento de eventos, os quais são gerados por mudanças de estado. E em [24], com foco na segurança, propõe-se uma etapa de intervenção na comunicação entre processos no sistema Android, que é quando se verifica autenticidade e atendimento de políticas configuradas pelo desenvolvedor.

É apresentado neste trabalho, uma proposta de *framework* para o desenvolvimento de aplicações ubíquas em AmbI. O objetivo é fornecer suporte à programação, teste e execução de aplicações, permitindo lidar de forma integrada com sistemas de grande complexidade. Este *framework* destaca-se por tratar dos desafios já identificados na Computação Ubíqua [11]. A *heterogeneidade de dispositivos* é tratada através da definição de um Modelo de Componentes Distribuídos, no qual o componente básico tem uma estrutura uniforme definida como um Agente de Recurso (AR), proposto inicialmente em [8] como uma entidade de coleta de informações de contexto. Neste trabalho, o AR, além de manter

informações de contexto, age como um componente que encapsula o código do dispositivo a ele associado, incluindo os aspectos de interação com os componentes da aplicação e atuação sobre a interface física, a qual interage diretamente com o ambiente. Para a questão da *variedade de informações de contexto e serviços*, é proposto um Modelo de Contexto [13] que possibilita definir as variáveis de interesse, bem como o armazenamento e a distribuição de tais informações. Uma ferramenta complementar possibilita a criação de regras de contexto e a geração do código de suporte necessário a interpretação dessas regras em tempo de execução. Finalmente, em relação à questão sobre a *disponibilidade de recursos* e sua acessibilidade pelos usuários finais, o *framework* inclui uma aplicação de Interface de Prototipagem e Gerenciamento de Aplicações Pervasivas (IPGAP) [5], voltada à visualização e ao teste de aplicações ubíquas, mesclando componentes reais e virtuais.

Os ARs foram concebidos para representar módulos do sistema de aplicações ubíquas. Desenvolver uma arquitetura para suporte de ARs na camada de *middleware* é o foco principal deste trabalho. Esta arquitetura abrange questões como segurança interna, a modularidade do sistema, a organização semântica dos módulos e a interação destes com o ambiente inteligente aplicado.

A proposta do *framework* é colocada no nível conceitual, seus conceitos foram definidos objetivando a implementação em qualquer plataforma adequada para dispositivos móveis. Para efetivar esta ideia foi o *framework* foi desenvolvido em uma plataforma denominada *SmartAndroid*[2]. Este projeto viabiliza a demonstração prática do *framework* concretizando o processo de construção de aplicações ubíquas. A escolha pela plataforma Android é apoiada por sua acessibilidade, por ser um sistema de código aberto, e por sua disponibilidade, dado que diversos dispositivos baseados no Android, simples e complexos, estão sendo lançados no mercado. Com a integração dos dispositivos aos recursos reais do ambiente, a implantação das aplicações concebidas pelo *framework* se torna mais viável.

Para avaliar a proposta, foram definidas questões de competências para o *framework* nos quesitos de desenvolvimento de aplicações ubíquas. As questões são sobre aspectos de suporte, subscrição a eventos, localização e segurança. Exemplos de cenários e aplicações são abordados para resolver as questões apresentadas.

Um jogo da velha ubíquo (ver Seção 4.1) implementa a interação entre jogadores através de agentes distribuídos, o que envolve questões relacionadas a suporte e subscrição de eventos. Uma aplicação de controle de iluminação inteligente (ver Seção 4.2.1) atua

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

na economia de consumo de energia baseado na localização de ocupantes, o que resolve questões de localização. Esta aplicação serve também como base para a implementação de uma simulação de atividade no ambiente, por meio da ativação e desativação de dispositivos (e.g., lâmpadas e TVs), a qual é orientada por um cronograma gerado pelo histórico de movimentação e atividades de moradores. Finalmente, para resolver as questões de segurança interna de um AmbI é proposta uma aplicação de segurança residencial (ver Seção 4.3), cuja implementação consiste em utilizar regras de contexto envolvendo grupos de usuários e domínios de segurança privados.

Através do *framework* proposto, conclui-se que há uma capacidade de construção, depuração e controle em AmbIs que permite um desenvolvimento ágil de soluções. Com sua utilização, um programador seria capaz de criar aplicações ubíquas com uma curva de aprendizagem minimizada em relação ao que é disponibilizado pelas plataformas básicas dos dispositivos. Também, um usuário comum seria capaz de utilizá-las e controlá-las, através da IPGAP, com relativa facilidade uma vez tendo adquirido familiaridade com a interface gráfica do usuário (GUI) disponibilizada.

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta os conceitos básicos utilizados ao longo do texto, o que inclui a apresentação do Modelo de contexto e da IPGAP. O Capítulo 3 apresenta a arquitetura geral, em nível conceitual, do *framework* descrevendo as partes do suporte e da segurança. No Capítulo 4 é descrito o projeto de implementação denominado *SmartAndroid*. O quinto capítulo apresenta exemplos de aplicações que são avaliadas demonstrando a viabilidade e eficiência do *framework* através de questões de competências. Após a avaliação, o Capítulo 6 apresenta uma comparação com trabalhos relacionados. E para finalizar, o Capítulo 7 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos Básicos

Os conceitos da computação ubíqua são utilizados para conceber um *framework* adequado para ambientes com recursos computacionais avançados e com serviços integrados. Em complemento, a Inteligência Ambiental [3] é um conceito importante para a definição do espaço onde executam-se as aplicações ubíquas, as quais manipulam os recursos e serviços disponíveis. A Computação Sensível ao Contexto [12] é utilizada como ferramenta para a automatização de serviços em reação ao comportamento encontrado no ambiente e, por fim, a Prototipagem de Aplicações Ubíquas [32] permite ao desenvolvedor ter uma avaliação rápida destas aplicações, ao usuário provê a a capacidade de visualização configuração dos serviços no ambiente.

Neste Capítulo são apresentados os conceitos básicos utilizados ao longo da proposta da dissertação. Na Seção 2.1 é apresentada a Inteligência Ambiental contendo a definição de Aplicação Ubíqua, termo que é amplamente utilizado para se referir as aplicações abordadas neste trabalho. Na Seção 2.2 é definido o conceito de Agente de Recurso, módulo básico de sistemas ubíquos. Na Seção 2.3 são apresentados serviços de sensibilidade ao contexto e do modelo de contexto que permite desenvolvê-los. Na Seção 2.4 é apresentada a aplicação de Prototipagem de Aplicações Ubíquas utilizada neste trabalho. Finalmente, na Seção 2.5 é descrita a forma de comunicação utilizada pelos componentes das aplicações definidas através do *framework*.

2.1 Ambientes Inteligentes

A Inteligência Ambiental caracteriza o espaço manipulado pelas aplicações, denominado Ambiente Inteligente (AmbI). Um AmbI é geralmente baseado em uma arquitetura em camadas (ver Figura 3.2). A camada mais profunda é definida pelo espaço físico com

seus ocupantes (*e.g.*, moradores em uma casa, inquilinos em um prédio). Logo acima está a camada dos sensores e atuadores, o primeiro coleta informações de contexto e o outro provê serviços para atender as necessidades dos ocupantes. A camada intermediária, conhecida como *middleware*, é definida entre a camada de tomadas de decisão (camada de aplicações) e a camada de sensores e atuadores. O *middleware* inclui conceitos e mecanismos para a construção e execução de softwares com funcionalidades que usufruem de ambas das camadas intermediadas. Na camada de aplicações, as decisões podem ser tomadas por um ocupante ou através de mecanismos de inteligência artificial. Há propostas que envolvem tipos específicos de AmbIs, a "*smart home*" (casa inteligente) é um dos exemplos mais conhecidos, o qual é explorado em [19, 3, 27].

O controle de iluminação inteligente, descrito na Seção 4.2.1, é uma aplicação que envolve *smart homes*. Nela é explorada a localização dos ocupantes para o controle da iluminação nos diversos cômodos de uma casa. A próxima subseção formaliza a definição deste tipo de aplicação, denominadas aplicações ubíquas.

2.1.1 Aplicações Ubíquas

As aplicações que usufruem dos recursos disponíveis em AmbIs para prover serviços de forma ubíqua são denominadas Aplicações Ubíquas (AUbi). Uma AUbi é caracterizada por interagir com o *middleware* para manipular sensores e atuadores. Os sensores são utilizados para captar as informações dos recursos do ambiente (pessoas, objetos e espaços), para as quais a aplicação utiliza técnicas de sensibilidade ao contexto para tratamento e fornecimento de serviços adequados através de atuadores.

Desta forma uma AUbi tem por objetivo atender aos princípios da computação ubíqua lidando com: (i) a diversidade de dispositivos; (ii) a descentralização dos serviços e (iii) as questões de conectividade [11]. Em (i) costuma-se aplicar a definição de interfaces padronizadas para abstrair as especificidades de cada dispositivo, de forma a facilitar o acesso e a manipulação dos recursos. Para resolver esta questão foi proposta a definição de Agente de Recurso (AR) (ver Seção 2.2). Atendendo ao princípio (ii), utiliza-se um suporte ao gerenciamento de recursos (ver Seção 3.1.2) para acessar e manipular os recursos espalhados no ambiente a partir de qualquer dispositivo. Quanto ao princípio (iii), deve-se avaliar as condições do ambiente para configurar características de baixo nível relatadas a conectividade: uma casa inteligente pode ser bem atendida por uma rede Wi-Fi; já em um estádio de futebol, devido sua grande extensão, pode se adequar melhor a uma rede de bandas 3G/4G.

No Capítulo 4 são apresentadas aplicações que buscam explorar as principais características de AUbis. Ao longo do trabalho, exemplos são ilustrados para uma melhor compreensão dos conceitos apresentados.

2.2 Agentes de Recurso

Um AmbI é caracterizado por possuir uma diversidade de tipos de recursos inteligentes, o que torna difícil o seu gerenciamento. Há sensores de temperatura, presença, de movimento, medidores de frequência cardíaca, glicose, controladores de televisores, sistema de refrigeração, de tranca eletrônica, além de outros módulos de software e hardware que forneçam algum serviço para o ambiente. A nível de *middleware*, estes componentes são encapsulados em estruturas de dados predefinidas, isto permite que sejam manipulados por aplicações interessadas em suas funcionalidades. Estas entidades encapsuladoras possuem a denominação comum de Agente de Recurso (AR), compreendido como elemento que expõe, em forma organizada, uma interface com informações do recurso representado. Assim, ao expor serviços seguindo o estilo SoA, outros componentes podem acessá-lo de maneira uniforme diminuindo a complexidade de integração. Por exemplo, os detalhes da coleta de dados de um sensor de temperatura são conhecidos internamente pelo AR, o qual se encarrega de fornecer uma interface simples para outros componentes do sistema, como coletar a temperatura em Celsius (C) ou em Fahrenheit (F) que internamente, pelo sensor representado, está captado em Kelvin (K).

No *framework*, um AR é definido como um módulo básico que encapsula o código do recurso representado. Atributos importantes deste módulo como sua identificação, localização e classe (ver Seção 2.2.1) são armazenados no *middleware*. Este processo permite que aplicações o descubram a partir destes atributos. Por exemplo, é possível obter o AR de TV especificando o cômodo, basta fazer uma busca local de ARs ($ARsInPlace = search(PLACE,"room")$), e em cima do resultado buscar pela classe TV (($ARsInPlace = search(TYPE,TV.class)$) (ver detalhes na Seção 3.1.2.4). As classes definem interfaces que podem ser utilizadas pelas aplicações para realizar operações e acessar funcionalidades do recurso representado. Uma definição semelhante sobre AR pode ser encontrada em [8], porém sendo apresentado apenas como uma entidade de coleta de informações de contexto.

2.2.1 Classes de Agentes de Recursos

As AUbis lidam com a questão da *diversidade e heterogeneidade de recursos disponíveis* através da definição de Classes de Agentes de Recursos (CAR). Em [27] é descrita uma hierarquia de tipos para entidades características em AmbIs, complementarmente é apresentada uma ontologia guiada pelas propriedades características de cada tipo. Nosso trabalho define CARs baseadas nessa estruturação em hierarquia, mas deixa em aberto a utilização ou não de uma ontologia de propriedades, permitindo assim uma a maior flexibilidade para a implantação de acordo com as particularidades de cada tipo de AmbI. Em analogia com a Programação Orientada a Objetos (POO), define-se CAR como a representação de um tipo de recurso com uma interface específica dentro do padrão de AR. Toda CAR é filha de AR, similarmente em Java, toda classe é filha de Object.

Com o uso de POO, diferentes instâncias de AR (IAR) podem ser controladas pelo mesmo tipo de interface. Além disso, diferentes CARs podem herdar funcionalidades de super CARs, e seguindo a ideia de polimorfismo, instâncias de classes com códigos diferentes podem ser manipuladas por uma mesma interface de sua superclasse comum. Toda essa flexibilidade fornecida pela POO é utilizada pelas AUbis, facilitando a manipulação dos recursos. Por exemplo: televisores, celulares e *tablets* possuem serviços em comum como ligar/desligar, mostrar vídeo, tocar música etc., logo podem ser controlados por uma mesma superclasse denominada audiovisual.

A definição de CAR e IAR é utilizada pelo gerenciamento de recursos apresentado na Seção 3.1.2 e no Modelo de Contexto apresentado na Seção 2.3. Isto permite que, através da especificação da classe, instâncias de recursos sejam descobertas e tenham suas informações de contexto coletadas. Isto permite, por exemplo, que uma IAR encontre um dispositivo audiovisual qualquer e colete a sua imagem independentemente de ser uma TV, celular ou *tablet*. Na Seção 3.1.1, a estrutura das classes são descritas detalhadamente, no contexto da proposta.

2.3 Contexto

O contexto é fundamental para capacitar AUbis a manipularem informações de elementos do AmbI. Sintetizando propostas anteriores, Dey e Abowd [12] definiram contexto como qualquer informação relevante que caracterize a situação de entidades, especificamente: pessoas, lugares e coisas. No AmbI, “Lugares” são os cômodos, os andares de uma edificação, ou espaços em geral que possibilitam a localização de outras entidades; “Pessoas”

são indivíduos que povoam o ambiente e interagem com o mesmo; e "Coisas" são representações virtuais de objetos físicos, geralmente associados a componentes de software e com capacidade de comunicação.

Como suporte, pode ser predefinido CARs no nível do *middleware* para cada entidade básica de contexto. O CAR de "Coisas" provê informação de localização. Para "Lugares", necessita de um formato padrão (mapa ou planta do ambiente) para a área ocupada e operações básicas de localização. Para "Pessoas" é especificado um suporte à mobilidade e definida uma estrutura de agregação de "Coisas" utilizadas por esta entidade, por exemplo, para estabelecer uma rede de sensores corporais (*Body Sensor Network* - BSN).

2.3.1 Sensibilidade ao contexto

A sensibilidade ao contexto é focada em determinar o comportamento de um sistema de acordo com as informações de contexto coletadas. Em [4], são apresentadas propostas de *frameworks* concebidos seguindo esse conceito. Duas se destacam [26, 25] por influenciarem propostas mais atuais na área. O trabalho exposto em [26] destaca-se pela definição de agentes sensíveis ao contexto, os quais determinam a modularização de serviços característicos deste conceito em componentes do sistema. Estes agentes, quando compostos, são capazes de formar sistemas adaptáveis e mais completos, tais como blocos em uma construção civil. O trabalho exposto em [25] possui um aspecto mais prático, promovendo adicionalmente o uso de simulação, através de arquivos de configuração, para definir o comportamento dos agentes. Com essa capacidade, é possível explorar a resolução de conflitos entre regras de contexto. Em [25] foi definido um método, que recebe como entrada duas regras (no esquema evento-condição-ação (ECA)), e verifica se as condições geram o mesmo resultado para pelo menos uma entrada de valores, se isso ocorre e as ações (ou operações) especificadas são diferentes, então o conflito é detectado.

Exemplificando a sensibilidade ao contexto em uma casa inteligente, tem-se a seguinte situação: se uma pessoa sai de casa e esquece uma lâmpada acesa, dois contextos podem ser inferidos: ou foi um esquecimento, portanto um sistema desliga a lâmpada; ou foi proposital (*e.g.*, para simular uma presença), portanto nada é realizado. Entretanto, o sistema não possui informação suficiente para tomar uma decisão segura, então a detecção de morador na casa e o histórico indicando uso escasso da lâmpada são utilizados como complemento para indicar o desligamento da lâmpada. Desta forma, o sistema agregou informação suficiente para realizar um serviço que seria feito por uma pessoa. Quanto mais confiáveis são as informações coletadas mais o serviço se aproxima do comportamento

natural desejado.

A aquisição e avaliação do contexto de forma automatizada contribui para a construção de AUbis, sem este serviço seria exigido uma entrada de dados ou comandos diretamente por parte dos usuários, o que elimina o aspecto ubíquo. O *framework* proposto neste trabalho adota um Modelo de Contexto, abordado em [13], que permite construir módulos de coleta e interpretação de contexto para atuar em um AmbI.

2.3.2 Modelo de Contexto

Um Modelo de Contexto é proposto em [14] com o objetivo de lidar com questões relacionadas a *variedade de informações de contexto* presentes em AmbIs. Este modelo permite a construção de regras de sensibilidade ao contexto, promovendo *ações* de acordo com mudanças detectadas. A aquisição de contexto ocorre através de um mecanismo de comunicação baseado em eventos (*publish-subscribe*). Usando este mecanismo, uma entidade pode registrar interesse em qualquer informação de outras para receber, na forma de eventos, seu estado atualizado (ver Seção 2.5). A *informação* é denominada variável de contexto (VC) e as *ações* são executadas através de operações (OP). O Modelo de Contexto propõe duas estruturas para auxiliar na construção de regras: um Interpretador de Contexto (IC) para avaliar as VCs coletadas e um Atuador de Contexto (AC) que responde ao IC executando as OPs.

2.3.2.1 Variáveis e operações de contexto

Um AR, como entidade representante de um recurso, possui variáveis e funções fundamentais para seu funcionamento e para serviços de contexto. As suas variáveis expostas a outras entidades são tratadas como VCs. Em uma regra de contexto, VCs são utilizadas como entradas para a sua avaliação. E como ação de uma regra são definidas as operações de contexto (OP). A coleta de estatísticas internas, estado da bateria, e em geral, as funções expostas para que outros agentes possam usar são exemplos de OPs. Em um exemplo de sistema para controle de iluminação inteligente (ver Seção 4.2.1), é definido um serviço onde um agente solicita o desligamento de uma lâmpada (uma OP exposta) quando uma pessoa se ausenta do local durante um tempo (condição da regra). Antes de executar, este serviço se subscreve ao VC de ausência/presença de um sensor. Então o serviço é executado escutando a chegada do evento subscrito, quando este chega indicando ausência, a condição é avaliada e se satisfeita, o sistema realiza a OP representando a ação de desligar a lâmpada.

2.3.2.2 Interpretadores e Atuadores de Contexto

Regras de contexto podem ser estruturadas por Interpretadores de Contextos (IC) e por Atuadores (AC). Além da função de coleta de informações advindas de VCs, um IC pode promover uma agregação destas variáveis e avaliação de condições pré-configuradas; quando satisfeitas geram um evento a ser notificado para os ACs interessados na regra. Esta ideia se baseia no princípio conhecido como Evento-Condição-Ação (ECA) abordados em [31, 21], onde eventos e condições são tratados em ICs e ações são realizadas por ACs.

Tratar as ações em ICs é uma outra abordagem que poderia ser considerada. Entretanto, apesar de não haver restrições quanto as abordagens, por uma questão de disciplina a proposta de separação é conveniente por visar uma maior flexibilidade na definição das regras, a qual reflete em um maior poder de configuração tanto para desenvolvedores quanto para usuários comuns. Por exemplo, na aplicação de economia de energia a ação de desligar a lâmpada poderia ser facilmente replicada para outras, bastando que atuadores de cada nova lâmpada se subscrevessem ao mesmo IC.

2.4 Prototipagem de Aplicações Ubíquas

Aplicações Ubíquas podem exigir uma grande quantidade e diversidade de recursos a serem utilizados no ambiente alvo, o suficiente para desafiar a *disponibilidade de recursos* reais e dificultar seu teste e avaliação. Além disso, em algumas aplicações, recursos podem ser integrados dinamicamente ao ambiente. Para resolver esta questão é proposta uma Interface de Prototipagem de Aplicações Pervasivas (IPGAP) em [16]. Esta aplicação permite tanto a um desenvolvedor quanto a um usuário comum visualizar ARs e instanciar novas entidades simuladas. Para um usuário a IPGAP tem a função de configuração de serviços no conjunto de recursos do AmbI, para o desenvolvedor permite a depuração de aplicações ubíquas em um ambiente ideal simulado. A princípio, esta solução é concentrada em uma mesma GUI, como será visto na Figura 2.1, mas a ideia é que em trabalhos futuros se tenha uma dedicada aos desenvolvedores e outra aos usuários finais.

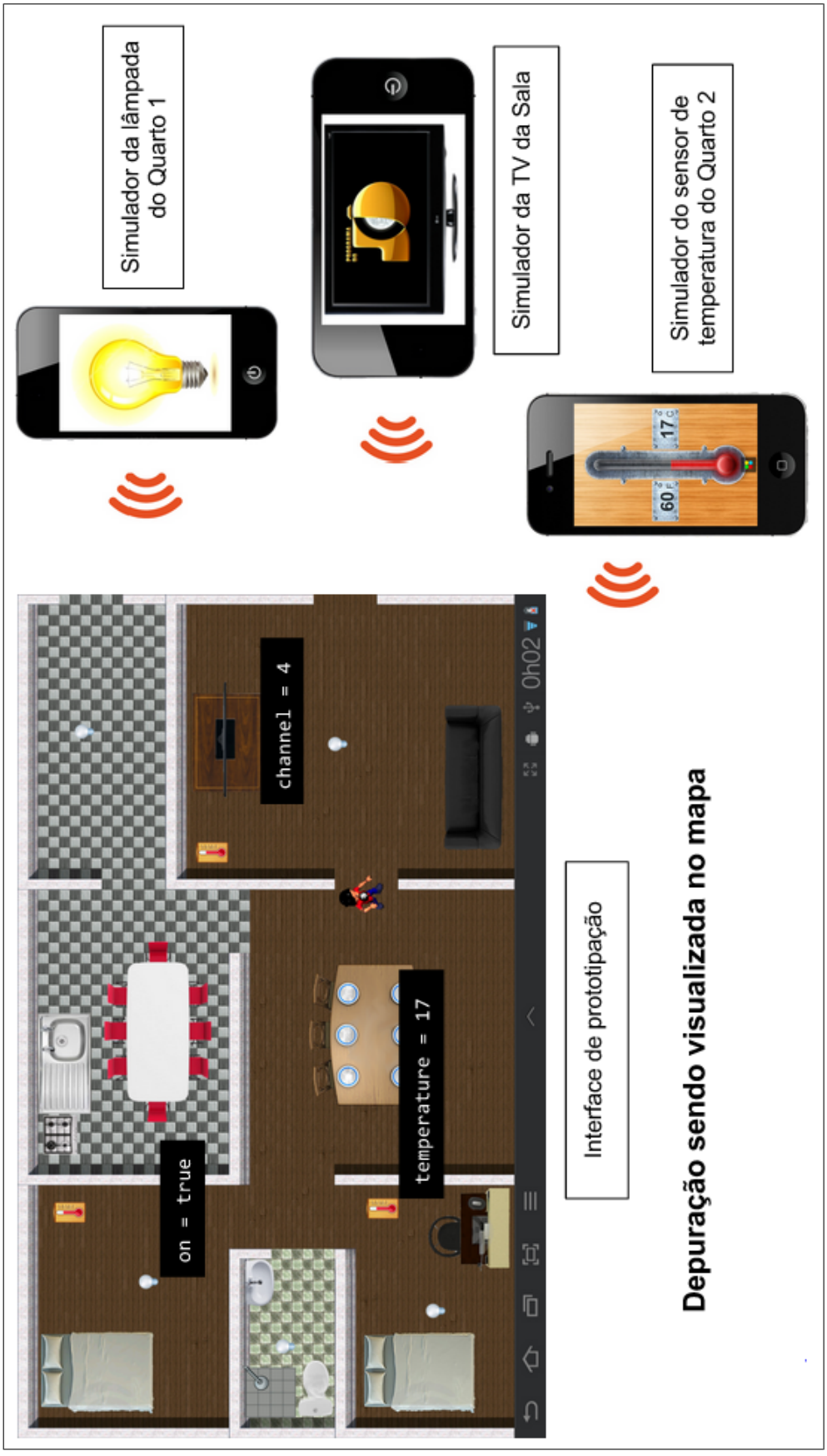

Figura 2.1: Interface de Prototipagem

Na Figura 2.1, é apresentada a GUI da IPGAP onde estão representados os recursos inseridos no mapa do AmbI. Estes recursos podem estar representados por entidades reais ou simuladas dos agentes. Na Figura 2.1 são apresentadas três entidades emuladas em *smartphones* individuais (lâmpada, televisor, e termômetro) e no mapa outras entidades criadas (simuladas) através da própria aplicação (*e.g.*, camas e sofás inteligentes, pc, chuveiro). Todos os tipos de entidades, sejam simuladas, emuladas ou reais, podem ser igualmente visualizadas no mapa.

As IARs que são representadas na IPGAP podem ser acessadas e controladas remotamente desde que suas operações (OPs) são expostas segundo o conceito de ARs. Elementos reais podem interagir com elementos simulados permitindo ao desenvolvedor depurar suas aplicações de forma completa em ambiente híbrido.

A IPGAP é uma aplicação fundamental para a construção de AUbis. Tem como benefícios a agilidade na validação de módulos de uma AUbi proposta. Em particular, ela permite uma avaliação concentrada de seu funcionamento e possibilita o uso de boas práticas de avaliação e testes, tipicamente requeridas no desenvolvimento de sistemas ubíquos. Com estes benefícios foi possível estabelecer exemplos complexos para validar as competências do *framework*.

2.5 Comunicação

Sistemas distribuídos possuem o requisito de transparência na comunicação, o que permite que aplicações concentrem-se na lógica do problema. O *framework* resolve a questão da comunicação utilizando-se de dois mecanismos básicos: de comunicação síncrona pela invocação remota de procedimentos (*Remote Procedure Call* – RPC) e de comunicação assíncrona pela comunicação baseada em eventos (ou interesses) seguindo o paradigma *publish-subscribe* [15].

O mecanismo de RPC traz como principal benefício a interação direta entre os recursos. Este mecanismo permite que métodos remotos sejam invocados identicamente a métodos locais tornando a comunicação transparente. Para métodos remotos, uma instância invocadora utiliza um *proxy* da classe da entidade invocada, nele estando contida a sua interface de chamadas de procedimentos públicos. A chamada e o retorno do procedimento são enviados através de mensagens serializadas (através da notação JSON). Comandos podem ser enviados aos dispositivos através de IARs, por exemplo, para desligar uma lâmpada ou um chuveiro, ou alterar remotamente o *setup* de um ar-condicionado.

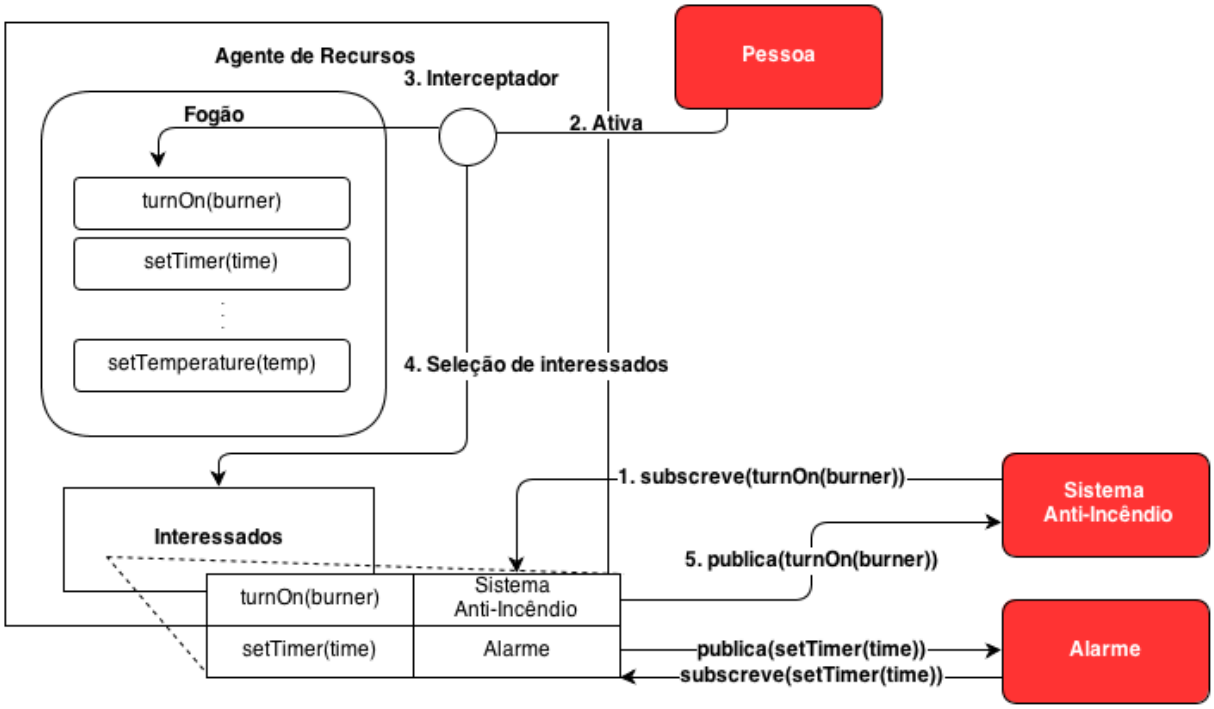

Figura 2.2: Esquema de *publish-subscribe*

Além de comandos, invocações remotas podem ser usadas para manipular dados e adquirir estados internos dos recursos.

O paradigma *publish-subscribe* é utilizado no *framework* proposto para a aquisição assíncrona de informações de contexto de recursos do ambiente, estas informações são representadas pelas VCs (ver Seção 2.3.2.1). A Figura 2.2 ilustra um exemplo de subscrição e publicação em um fogão. Neste exemplo, um fogão é visto como um foco de risco, seu estado é acompanhado por um sistema de informação e controle (Sistema Anti-Incêndio) para que um usuário seja informado sobre situações de risco e para que haja controle quando este se agravar (desligar fogão ligado em tempo excessivo). O fogão pode ter suas VCs subscritas por qualquer entidade, logo um alarme que esteja localizado em um cômodo distante se subscreve na VC do cronômetro (*timer*) para avisar que a comida está pronta por exemplo. Inicialmente, o Alarme e o Sistema Anti-Incêndio se subscrevem aos eventos *setTimer(time)* e *turnOn(burner)*, respectivamente, do Fogão (Passo 1) e esta subscrição é armazenada no campo de Interessados. Posteriormente, uma pessoa liga uma boca do fogão (*turnOn(burner)*) (Passo 2). Em seguida, ocorre a interceptação (Passo 3), que por meio de técnicas de Programação Orientada a Aspectos (POA) ativa o código de seleção e notificação de interessados. Na seleção, o Sistema Anti-Incêndio é escolhido (Passo 4) para receber a notificação (Passo 5). Já o alarme, não selecionado no processo, só recebe notificação quando o método *setTimer(time)* ocorrer. Expressões

lógicas podem ser elaboradas a partir de VCs e usadas para interpretações sofisticadas no Ambiente Inteligente [14].

O suporte necessário para prover a comunicação no *framework* a nível de implantação inclui uma infraestrutura básica de rede. Por conveniência, foi adotado o padrão Wi-Fi que garante uma segurança básica adequada aos ambientes inteligentes. Por exemplo, barreiras a agentes externos podem ser facilmente estabelecidas e mecanismos de criptografia como WPA podem ser imediatamente agregados. Há uma série de requisitos, relativos a questões de segurança, que podem ser atendidos utilizando técnicas de controle de acesso típicas da internet e determinando domínios associados a grupos de usuários como pode ser visto na Seção 3.2.

2.6 Conclusão do capítulo

Esse capítulo apresentou os conceitos e mecanismos básicos utilizados ao longo da proposta do *framework*. Caracterizou-se o espaço de atuação do trabalho, o Ambiente Inteligente (AmbI), separado nas camadas físicas, de *middleware* e de aplicação. Definiu-se o tipo de aplicação executada neste ambiente, denominadas Aplicação Ubíquas (AUbi), que são construídas através do *framework*. Apresentou-se a modularização básica destas aplicações, denominada como Agente de Recurso (AR), utilizado a nível de *middleware*, para manipular recursos da camada física. Para manipular as informações dos recursos, foi apresentado o Modelo de Contexto para construção de serviços sensíveis ao contexto. E para visualizar o AmbI (contendo ARs simulados ou reais), ter acesso as informações de contexto de cada recurso e testar as AUbis, foi apresentada a Interface de Prototipagem de Aplicações Pervasivas (IPGAP).

Capítulo 3

Proposta do Framework

O *framework* provê qualidades e padrões de programação tipicamente requeridos em aplicações ubíquas focadas em AmbI. Dentre as qualidades está a capacidade de abstrair detalhes das partes da aplicação que envolvem a comunicação, a aquisição de contexto, a localização de recursos e a segurança. Como abstração aos recursos são utilizados agentes (AR). O AR é a unidade básica de modularização do *framework*, sendo utilizado na modelagem, implementação e gerenciamento das aplicações. Os ARs podem ser utilizados, por exemplo, para representar sensores de temperatura da casa, ou um sensor de vazamento de gás na cozinha. Em geral, ARs podem encapsular qualquer funcionalidade implementada por software, como um gerenciador de outros ARs de cuidados a saúde, tais como medidores de pressão arterial e de glicemia.

Este trabalho foca na estruturação e suporte do *framework*. Este se encontra inserido no trabalho do grupo do Laboratório Tempo [1] que aborda de forma completa outros pontos da proposta. O Modelo de Contexto é apresentado brevemente na Seção 2.3.2 contemplando o trabalho completo abordado em [14]. E a prototipagem e representação visual das aplicações ubíquas no ambiente é descrito em [16] (ver Seção 2.4.

Para consolidar e avaliar a proposta do *framework*, seus conceitos e mecanismos foram implementados numa plataforma que denominamos *SmartAndroid*(ver Apêndice A). Esta escolha permitiu usufruir da acessibilidade a tecnologia *Android* [28] facilitando a integração com uma diversidade dispositivos e o desenvolvimento de sistemas embarcados. Com isto, foi possível construir rapidamente a plataforma agilizando a experimentação de aplicações ubíquas sofisticadas usufruindo de sua amplitude. Ainda assim, deve ser ressaltado que o *framework* usa conceitos de implementação imediata em outras plataformas

[1] www.tempo.uff.br

distribuídas atuais. Uma forma de implementá-los é utilizar a técnica de *Wrappers* para integrar componentes de outras tecnologias em aplicações *SmartAndroid*. A arquitetura do *framework* é definida a partir de um Modelo de Componentes Distribuídos (Seção 3.1) e de um Modelo de Contexto (Seção 2.3).

Este Capítulo apresenta a arquitetura geral do *framework* através de um Modelo de Componentes Distribuídos (MCoD) (Seção 3.1), e descreve domínios de segurança de informações de ARs em AmbI (Seção 3.2). Inserido na arquitetura geral está a descrição geral dos ARs e de CARs fundamentais para auxiliar serviços de contextos (Seção 2.2.1). E para gerenciar os elementos do MCoD é apresentado o Suporte ao Gerenciamento de Agentes de Recursos (SGAR) que permite organizar semanticamente um AmbI.

3.1 Modelo de Componentes Distribuídos

A arquitetura geral do *framework* é centrada em um Modelo de Componentes Distribuídos (MCoD). Um protocolo para MCoD é documentado em [6] que dentre as inúmeras definições: formaliza chamadas de procedimentos remotos (RPC), estruturação de `objetos` remotos e define operações fundamentais para qualquer componente. Seguindo esta ideia, o MCoD do *framework* é proposto para prover suporte a AmbIs diversificados, se adequando a tipos como casas, prédios e estações de metrô inteligentes, etc.

Como visto anteriormente na Seção 2.1, sistemas para AmbI são geralmente estruturados nas camadas física, de *middleware* e de aplicação. O Modelo de Componentes Distribuídos segue esta estrutura para mapear as aplicações ubíquas no AmbI. Neste Modelo são definidos: a estrutura do AR e o Suporte ao Gerenciamento de Agentes de Recursos (SGAR). O SGAR encontra-se no nível de *middleware* para concentrar informações essenciais das IARs criadas no AmbI, e prover serviços de segurança, descoberta, registro e localização destes elementos.

A Figura 3.1 apresenta a visão estrutural da arquitetura do *framework*. Sua composição básica consiste em: uma aplicação inicializadora denominada *Daemon* de Inicialização; um *daemon* de comunicação que aguarda e encaminha as chamadas RPC e eventos (vide Seção 2.5) de outras IARs; e o SGAR que realiza serviços de suporte quando recebe uma das chamadas ou eventos encaminhados pelo *daemon*.

Os componentes do *middleware*, *daemon* de comunicação e SGAR, permitem a construção de uma variedades de MCoD adaptados para cada conjunto de AUbis e AmbIs. Um exemplo de implementação muito explorado ocorre em casas inteligentes, onde IARs

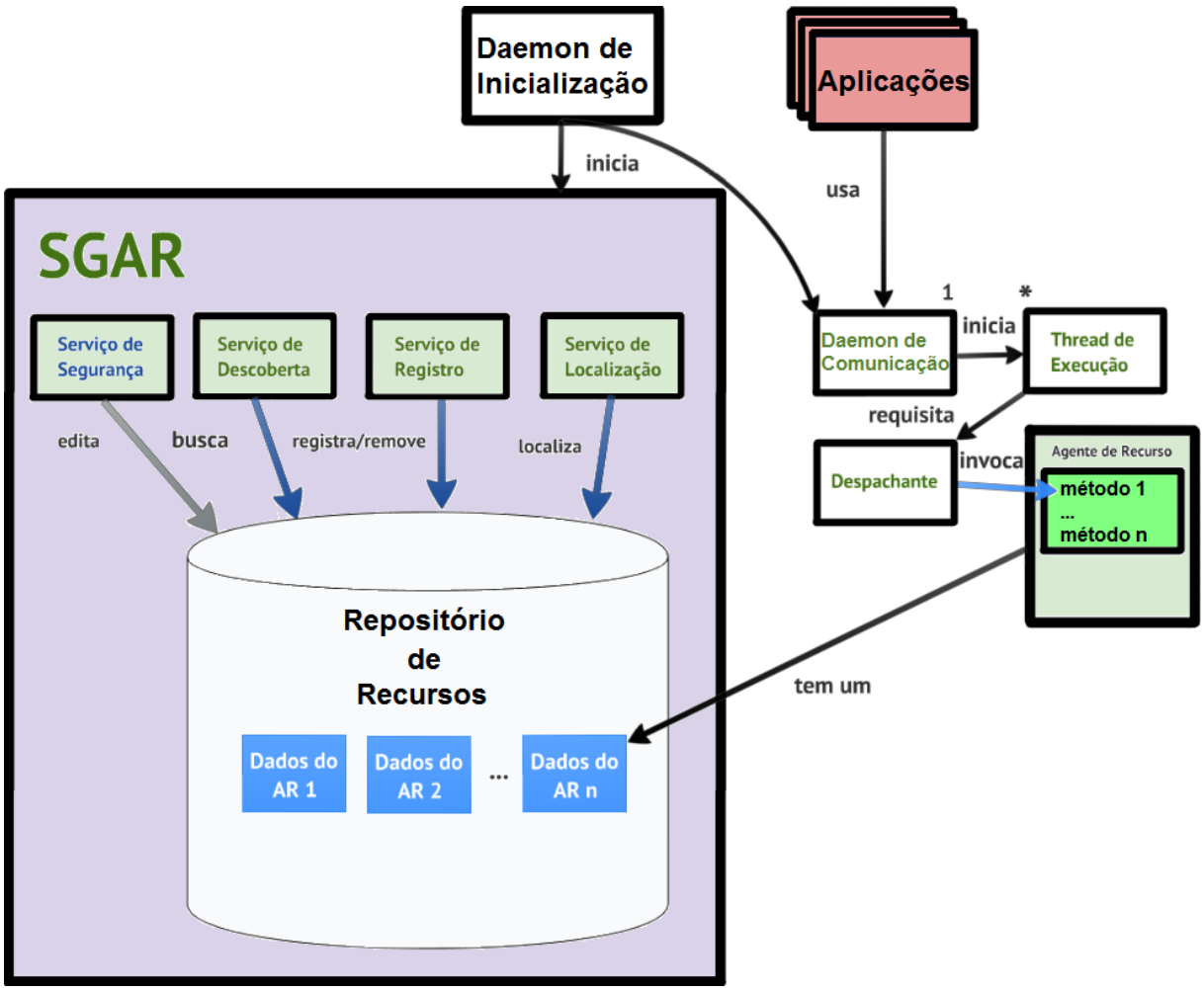

Figura 3.1: Arquitetura do *Framework* na camada intermediária

representam recursos de uma casa como lâmpadas, televisores, fogões, geladeiras, etc. A Figura 3.2 ilustra um exemplo de MCoD em casas inteligentes: na camada de recursos estão os elementos físicos exemplificados por TV, `pessoa` e quarto; na camada de agentes básicos estão localizados seus agentes correspondentes, e no *middleware* estão os serviços de suporte (SGAR) para gerenciar tanto agentes básicos quanto os das camadas de aplicação e atender requisições de outros IARs ou AUbis; na camada de aplicação encontram-se as AUbis e suas IARs diretamente associadas. Na Figura 3.2, um interpretador contexto (*e.g.*, interpretador de atividades domésticas definido pela *App Telecare*) tem a função de coletar informações de contexto de instâncias (*e.g.*, informações da TV e do quarto para apresentar a atividade exercida na casa). Outras instâncias de interpretadores podem ser utilizados para coletar informações das mesmas e de outras instâncias do AmbI.

Existe uma grande variedade de CARs que se encontram em AmbIs, o que dificulta a adaptatividade e interatividade com AUbis. Por conta disso, propomos o estabelecimento de CARs padronizados que proveem uma referência semântica aos componentes

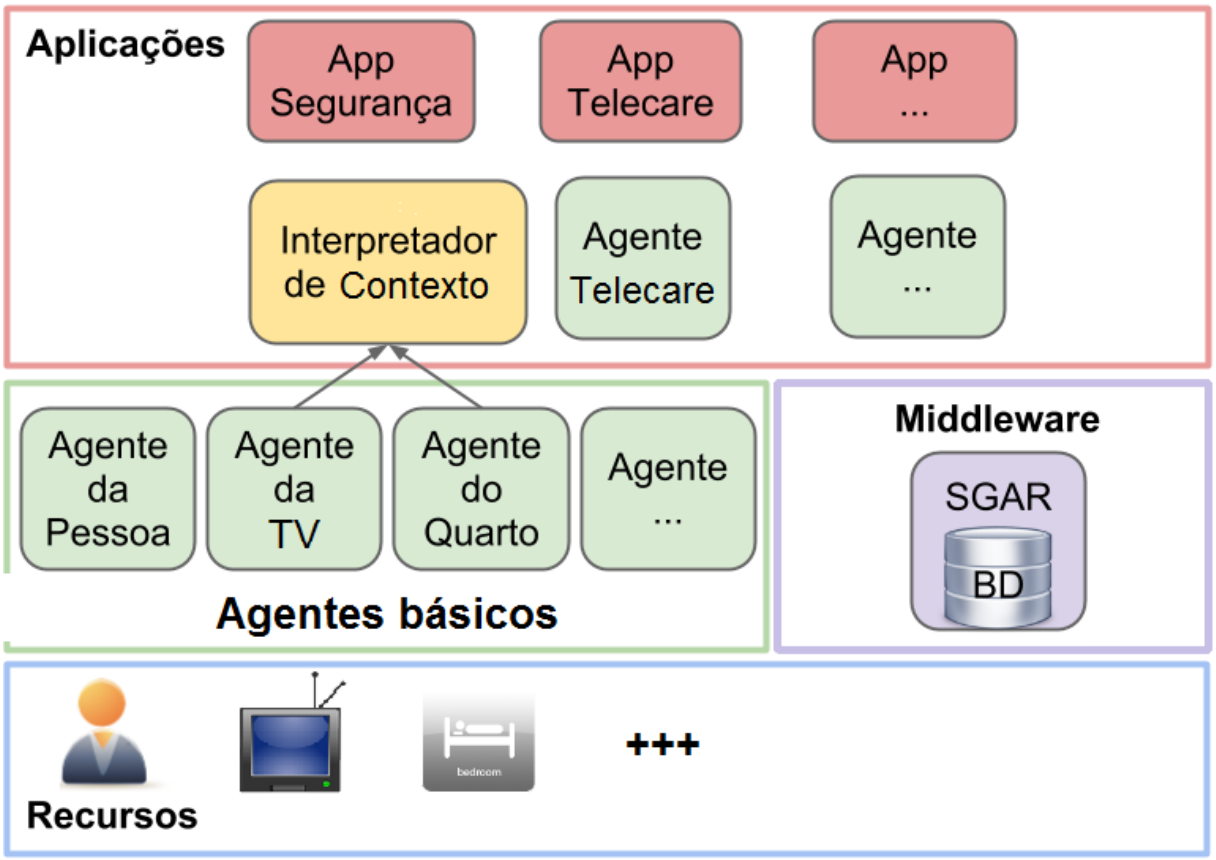

Figura 3.2: Modelo de componentes em uma casa inteligente

interessados. Na Seção 3.1.1, além de apresentar estas classes fundamentais, é descrita a estrutura e serviços básicos providos por um AR. Na Seção 3.1.2 são descritos os serviços oferecidos pelo SGAR para gerenciar componentes.

3.1.1 Estrutura de Agentes de Recursos e classes fundamentais

O Agente de Recursos reúne as características básicas necessárias para atuar em AmbIs e interagir com AUbis. A Figura 3.3 ilustra o exemplo de uma IAR de lâmpada descrita na estrutura básica de um AR. Toda IAR deve possuir: um `nome` (lâmpadaDaSala) para identificá-lo unicamente no AmbI; um `tipo` para identificar sua `classe` e sua respectiva hierarquia (luz\lâmpada); o `local` (Sala) para definir onde se encontra fisicamente; o código da `classe` (CAR de Lamp) com as variáveis e operações específicas (ligado e ligar()); e a lista de IARs interessados em suas informações de contexto (celular interessado em ligado).

A lista de interessados é um requisito que atende ao esquema *publish-subscribe*. Seguindo este esquema, é utilizada pela instância corrente para selecionar as IARs que devem

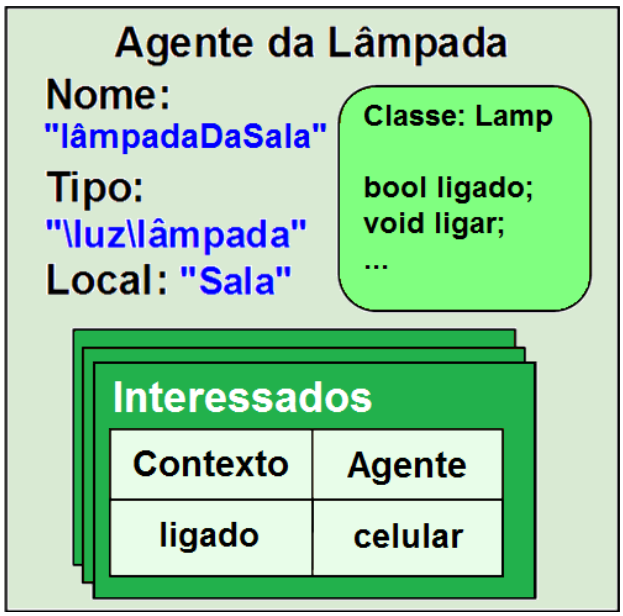

Figura 3.3: Arquitetura do *Framework* na camada intermediária

ser notificadas sobre eventos subscritos. Esta seleção corresponde ao Passo 4 ilustrado pela Figura 2.2. A subscrição a interesse é um requisito que atende serviços de coleta de informações de contexto, executados a priori por ICs. Através deste suporte é permitida uma flexibilidade em soluções para adquirir avaliar e atuar sobre estas informações. Por exemplo, é possível escolher entre interpretadores especializados e motores de regras para aplicar os serviços de contexto.

Quando um agente desejar se subscrever a outro ainda não registrado, poderão ser utilizadas informações de local e de tipo para efetuar este procedimento. Por exemplo, um agente pode querer se subscrever a uma `TV da sala`, só que não há nenhuma registrada, então este agente se subscreve a sala para receber atualizações dos agentes da classe TV que são registrados ou mudam sua localização para a sala. Este procedimento de subscrição a tipo e local é detalhado na Seção 3.1.1.1.

Algumas informações de contexto mudam com muita frequência e para não sobrecarregar a rede é preciso estabelecer precondições para sua notificação. Por exemplo, os dados de aceleração, captados por acelerômetros, podem mudar em questão de milissegundos. O *framework* permite uma flexibilização da programação interna do CAR, inclusive permite estabelecer a rotina adequada para a notificação de mudanças em cada uma de suas variáveis. Por exemplo, para variáveis que mudam com pouca frequência técnicas de programação orientadas a aspectos (POA) podem ser utilizadas para ativar a notificação assim que mudam. E para variáveis que mudam com muita frequência, como por exemplo o posicionamento de uma pessoa, podem ser estabelecidas condições para reduzir este

número, por exemplo: notificar a mudança de posicionamento a cada período de tempo.

A definição de CAR é fundamental para classificar IARs e o seus respectivos serviços oferecidos. O uso de CARs permite que serviços sejam realizados sem a necessidade de que sejam definidas instâncias específicas para atendê-los. Por exemplo, para mostrar um vídeo basta descobrir um agente audiovisual qualquer, pois este tipo possui definida a operação de mostrar vídeo, a qual aparelhos como TVs e *tablets* herdam. Como visto na Seção 2.2.1, essa ideia de definir entidades que devem realizar um serviço através das propriedades requeridas vem de [27]. Neste trabalho, uma ontologia é utilizada para requisitar operações através de um conjunto de propriedades, conjunto o qual define uma entidade virtual que configura uma busca criteriosa pela entidade real mais similar. A similaridade é verificada por uma função de utilidade.

Existem classes que se baseiam em características específicas em relação a um AmbI, é a partir delas que boa parte dos requisitos típicos podem ser concretizados. As classes fundamentais definidas no *framework* são: de `pessoas`, esta possui a característica de agregar recursos de sensoriamento corporal e ter uma grande mobilidade no ambiente; de `objetos` que correspondem aos dispositivos do ambiente; e de `lugares` que definem subespaços característicos nos ambientes. A relação entre estas três classes é constante, com `pessoas` geralmente utilizando ou interagindo com `objetos` e se movimentando por `lugares`. Estes três tipos são discutidos em [1] como atores fundamentais de produção de informações de contexto.

3.1.1.1 Classe de Lugares

Um ambiente consiste de um conjunto de espaços e subespaços cada qual com características, funcionalidades e abrangendo uma região. Esta classe serve como suporte as operações de localização e o conjunto de instâncias relacionadas são representados em um mapa (ver 3.1.2.5), ao considerar estes aspectos definiu-se a classe Lugar, cuja característica comum é a de possuir uma área. Dentre as funcionalidades (eventos, métodos e variáveis) estão operações de consulta de pertinência e eventos de entrada e saída de recursos (*e.g.*, `pessoa` sai do quarto).

Como proposta básica para a prototipação foram especificadas estruturas para o posicionamento de entidades e a definição de área. O posicionamento de recursos foi definido por uma coordenada cartesiana (*e.g.*, x e y para representação em duas dimensões), logo a área de um `lugar` é definida a partir de um conjunto de coordenadas. Para reduzir os custos de armazenamento e consequentemente em operações de pertinência, definiram-se

representações específicas para áreas geométricas:

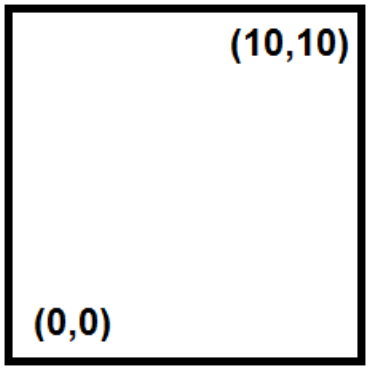

Figura 3.4: Representação da área de um quadrado

- Áreas retangulares: geralmente encontradas em casas inteligentes, são representadas por duas coordenadas (x,y), cada uma representando vértices diagonalmente opostos. A Figura 3.4 ilustra um exemplo de quadrado definido pelas coordenadas (0,0) e (10,10) que representam uma área 100 u. m.;

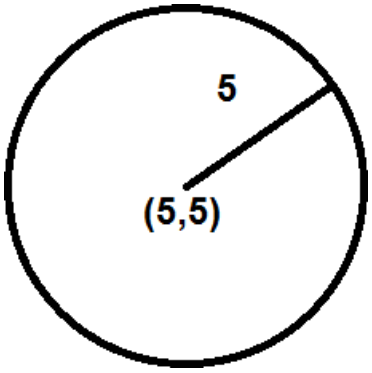

Figura 3.5: Representação da área de um círculo

- Áreas circulares: são representadas por uma coordenada centroide e um raio r. A Figura 3.5 apresenta um círculo com um centroide em (5,5) e com raio igual a 5 representando uma área de 25π u. m.;

- Áreas híbridas: por exemplo, na Figura 3.6 é apresentada uma área onde 3/4 é retangular e 1/4 completado por aresta circular, neste caso, é composta de uma lista de subáreas (círculo e quadrado), e a verificação de pertinência ocorre em dois níveis, primeiro definindo se pertence ao quadrante circular tendo como base o centroide, se pertencer compara-se a posição do recurso com a área do círculo senão com área do quadrado.

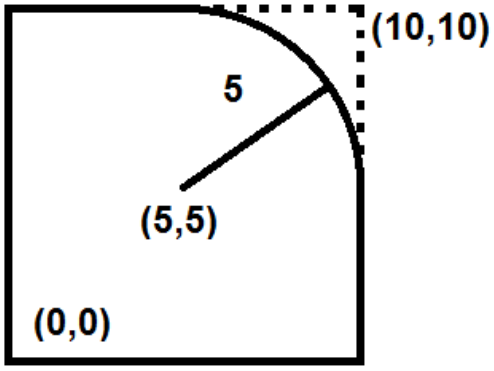

Figura 3.6: Representação de área híbrida com 3/4 de um quadrado e 1/4 de um círculo

Outras áreas geométricas podem seguir essa ideia para que a operação de pertinência seja mais próxima a complexidade constante (O(1)), ao invés de simplesmente percorrer uma lista de coordenadas e chegar a uma complexidade linear (O(n)).

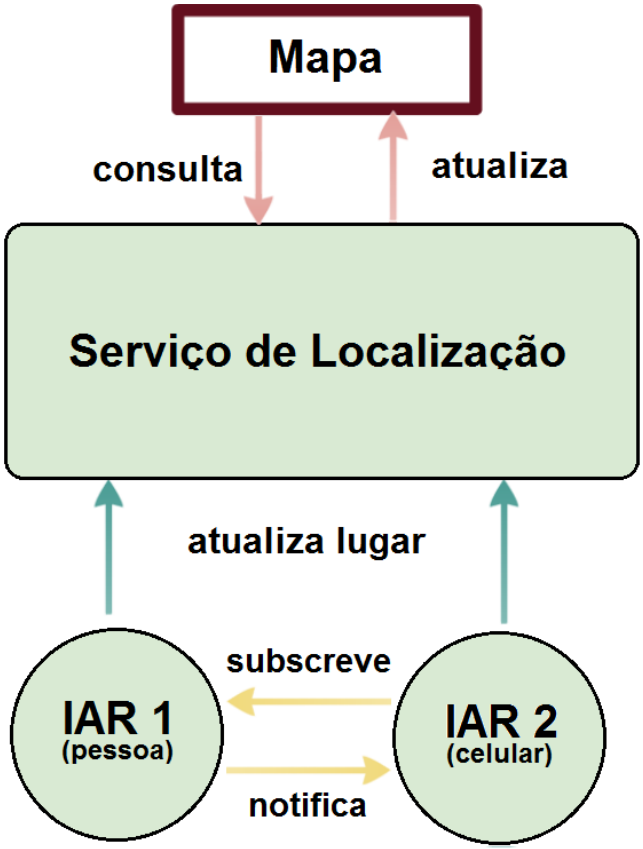

Figura 3.7: Atualização de localização de instâncias

3.1.1.1.1 Utilização de localização

Os eventos de entrada e saída são fundamentais para AUbis que desejam se subscrever ou atuar em recursos quando chegam ou saem de um `lugar`. Por exemplo, a Figura 3.7

ilustra o processo de atualização de localização. Uma aplicação de controle de iluminação (ver Seção 4.2.1) pode avisar ao celular (IAR 2) de uma `pessoa` (IAR 1) que ele esqueceu a luz do banheiro ligada. O evento de saída da `pessoa` deve ser previamente subscrito pela aplicação do celular; quando a `pessoa` sai do banheiro, o seu posicionamento e o `lugar` mudam (entra no cômodo vizinho e sai do banheiro) e isso ativa o evento subscrito (sai do banheiro); a aplicação do celular é notificada com uma mensagem contendo a informação do evento (saída) e o recurso ativador (agente `pessoa`); então, a partir da notificação, a `pessoa` é sinalizada em seu celular.

As mudanças de localização são administradas pelo Serviço de Localização (SLR) (ver Seção 3.1.2.5), que recebe as alterações de posicionamento e atualiza o mapa do repositório caso haja mudança de local (*e.g.*, sair de cômodo da casa). Assim, ao se subscrever ao SLR, é possível mapear o ambiente e a localização de instâncias de forma consistente com o estado atual.

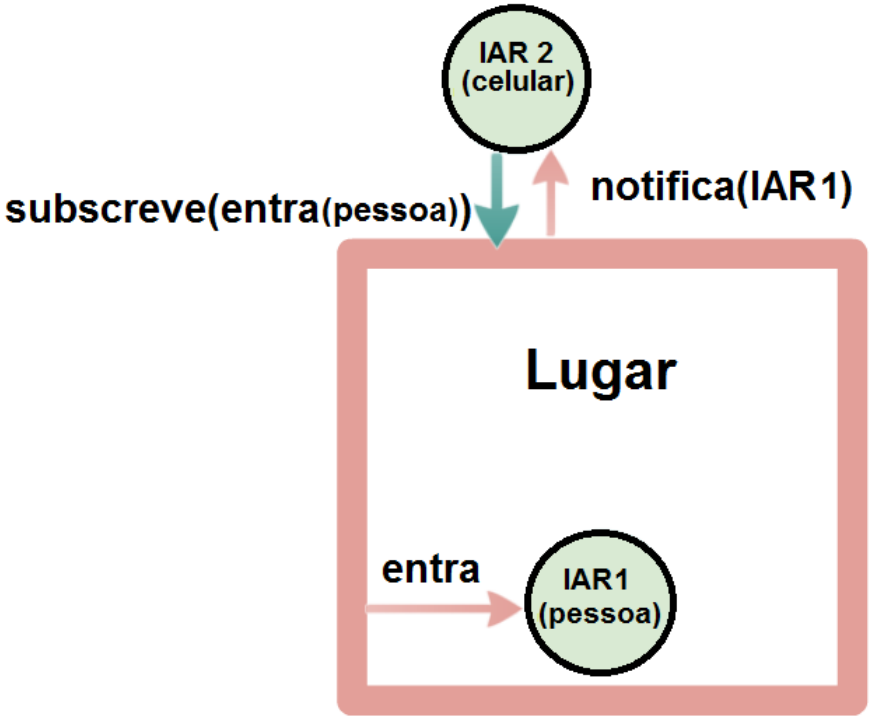

Figura 3.8: Evento de entrada de instância

Continuando na aplicação de controle de iluminação, esta deve ligar a lâmpada quando uma `pessoa` entra no respectivo `lugar`. Para isto, a aplicação de controle localizada no celular precisa se subscrever ao evento *entra(pessoa)* do agente do `lugar`, então sempre que um agente `pessoa` entrar no `lugar`, o evento é ativado e a lâmpada recebe o comando do celular para que seja ligada pelo seu agente. Este exemplo segue a ideia ilustrada na Figura 3.8, onde IAR 2 representa a `pessoa` e IAR 1 representa o celular.

Este processo demonstra a funcionalidade de subscrição de entrada e saída de instâncias de um `lugar` específico. Existem sensores de presença que se ligam diretamente a uma lâmpada e realizam a mesma função, sem a necessidade de um *middleware*. O uso do *middleware* permite a reusabilidade destes sensores. Por exemplo, o sensor de presença pode ser utilizado em múltiplos serviços como mudar o canal de uma TV ou ligar o ar-condicionado.

Na subscrição, pelo mecanismo de *publish-subscibe* (visto na Seção 2.5), se define o tipo de interesse (no caso do exemplo da Figura 3.8 se definiu o tipo pessoa(*entra(*`pessoa`*)*)). Este tipo de evento é ativado no processo de atualização de localização (Figura 3.7), quando se verifica uma mudança de `lugar`, além da de posicionamento, o método que ativa o evento (entrar ou sair) no agente de `lugar` é invocado pela IAR provocadora (IAR 1). O método notifica os interessados nos eventos de entrada e saída geral (IAR 2) e os interessados naquela classe ativadora (aplicação de controle da lâmpada interessada na entrada de `pessoas`).

3.1.1.2 Classe de Objetos

A classe de Objetos representa recursos físicos do ambiente, logo é imprescindível a especificação de sua localização. Os `objetos` estão associados semanticamente a três entidades básicas do *framework*: na classe de Pessoas, na de Lugares e na base de dados da SGAR. Em `pessoas`, está associada uma subclasse específica de `objetos` conhecidos como sensores corporais, os quais são utilizados para coletar informações sobre seu estado (*e.g.*, dados fisiológicos e status de saúde). Diferente da definição básica do AR, os `objetos` possuem necessariamente uma localização, existem `objetos` estacionários em cômodos de uma casa inteligente como por exemplo as janelas, portas, lâmpadas, os ventiladores de teto. Além disso, há outros característicos de certos tipos de `lugares`, mas que podem ser realocados em outro, como televisores em salas de estar, camas em quartos, chuveiros em banheiros, etc. A diferença é que estes atualizam a localização menos do que os agregados em `pessoas` e mais do que os estacionários.

Assim, é possível identificar a existência de dois tipos de objetos: móveis, e fixos. Em termos de implementação (ver *SmartAndroid* no Apêndice A), a aplicação de tecnologia de localização pode ocorrer de diversas formas, a questão é aberta ao desenvolvedor. Com os avanços tecnológicos em localização *indoor*, o *framework* contempla a possibilidade de auto localização de objetos, os quais possuem uma variável de posicionamento que pode ser alterada internamente por este mecanismo. Não sendo possível a auto localização,

agentes externos (*e.g.*, GPS, piso magnético) podem provocar, por invocação remota, a mudança desta variável.

3.1.1.3 Classe de Pessoas

Os agentes de `pessoas` proveem suporte à mobilidade e ao conjunto de sensores corporais disponíveis. Pessoas tem por características uma alta mobilidade, o que pode ocasionar uma sobrecarga de notificações de posicionamento no *middleware*. Para controlar esta sobrecarga, a frequência de atualização pode ser configurada pela aplicação específica, podendo ser em horas, minutos, segundos, etc. Portanto, uma lista de posições é definida para guardar o histórico de mobilidade.

Uma `pessoa` pode agregar uma diversidade de informações de contexto dependendo de sua situação. Um paciente, por exemplo, pode estar com febre em determinado momento e passar a usar um termômetro para acompanhar sua temperatura, depois que a febre passa não há mais necessidade de acompanhar esta informação. Além disso, o indivíduo pode adquirir uma outra doença como a diabetes que exige o acompanhamento da taxa de glicemia, logo é preciso agregar um medidor de glicose a seu agente. Ou seja, uma `pessoa` pode agregar e desagregar dinamicamente atributos sobre seu estado. Para solucionar esta questão, uma lista de `objetos` pessoais ativos é mantida como referência para consultas sobre eventuais informações específicas sobre o estado de uma `pessoa` (*e.g.*, pressão arterial captada por um medidor, nível de atividade física captado por um acelerômetro). Ou seja, este tipo de instância adiciona e remove objetos de sua estrutura, conforme suas necessidades e preferências, e ainda permite a outros agentes associar as informações de contexto vindas destes objetos a uma pessoa. Uma abordagem genérica e sistemática para controle da configuração de aplicações em ambientes inteligentes é descrita em [10].

Em uma aplicação visando a economia de energia, a lista de posições guardadas é disponibilizada para que eventualmente um mecanismo preditor identifique meios de utilizar os recursos de forma otimizada em relação ao posicionamento. Sistemas que precisam de um tempo de pré-processamento para realizarem um serviço ideal, como ar-condicionados, podem utilizar estas informações para prover mais conforto ao usuário de forma antecipada. A lista de `objetos` facilita na coleta de informações agregadas de `pessoas` através de ICs.

3.1.2 Suporte ao gerenciamento de recursos

Este suporte foi inspirado no padrão de projeto *Data Access Object* (DAO). O SGAR é composto por três serviços essenciais que que têm interface com a base de dados: Serviço de Descoberta de Recursos (SDR), Serviço de Registro de Recursos (SRR), Serviço de Localização de Recursos (SLR). Estes serviços manipulam o repositório de recursos onde: o SRR adiciona e remove registros de agentes; o SDR consulta os agentes registrados; e o SLR os localizam dentro do espaço físico do ambiente. No repositório estão armazenados os dados representativos de cada agente registrado, estes registros estão organizados de acordo com uma hierarquia de tipos que cada agente representa. Os dados representativos (ver Seção 3.1.2.2 correspondem ao nome, tipo, `lugar` e a referência de acesso por comunicação (ver 3.1.2.1). O uso de DAO permite separar a lógica da aplicação do esquema de banco de dados, os serviços mantém um endereço fixo (ver Seção 3.1.2.1) independentemente de haver ou não replicação na base, o que é desejável para atender os requisitos de tolerância a falhas. Portanto, o repositório pode mudar sua localização na rede, a qual se mantém indexada a um mesmo endereço lógico.

Apresentado na Figura 3.1, o SGAR concentra os serviços a nível de *middleware* do *framework*. Para uma IAR utilizar outra, consulta-se sobre sua referência, através do SDR, para obter a ligação com o respectivo *proxy*. Para que se disponibilize a outras, deve-se registrar através do SRR. Para encontrar instâncias a partir de um `lugar` consulta-se o mapa manipulado pelo SLR.

Os serviços do SGAR são IARs únicas localizadas em uma unidade centralizada e predefinida do ambiente, logo, seu endereço na rede da AmbI é divulgado a qualquer outra unidade que chega por meio do protocolo *multicast* de comunicação. Esta configuração não é mandatória, podendo-se adotar a estrutura de DNS, modelo que é facilitado dada a definição de RANS (ver Seção 3.1.2.1).

3.1.2.1 *Resource Agent Name System*

Assim como uma IAR se identifica unicamente no AmbI através do nome, sua identificação única no nível de comunicação ocorre por meio do *Resource Agent Name System* (RANS). Sua função é similar a de um DNS, assim independentemente de seu endereço de rede mudar, o seu RANS continua o mesmo, apenas atualizando a entrada respectiva na tabela com a nova referência física de rede. Diferentemente do nome do IAR, o RANS não pode ser editado depois de definido inicialmente. A notação definida para identificar um agente

exige a expressão ".ra"no fim (*e.g.*, "tv.ra"). O RANS é utilizado para configurar um *proxy* de acesso direto a uma IAR.

Os esquemas de nomeação e endereçamento podem ser generalizados para operarem no nível da chamada *Internet of things*. Esse aspecto será desenvolvido futuramente.

3.1.2.2 Objeto de dados do Agente de Recurso

O Objeto de Dados do Agente de Recurso (ODAR) é composto de RANS, nome, tipo e `lugar`. Os três primeiros são dados obrigatórios e `lugar` é opcional devido a irrelevância dessa informação no caso de uma IAR representar um módulo de software. Dados exclusivos de uma IAR (*e.g.*, canal de tv, temperatura do ar condicionado) não são armazenados no *middleware*, aplicações que desejem obtê-las devem acessá-las diretamente consultando pelo seu tipo (CAR) ou nome da instância.

3.1.2.3 Serviço de Registro de Recursos

As operações básicas do SRR são o registro e a remoção de ODARs no repositório de recursos. Adicionalmente, recursos podem se subscrever a essas operações permitindo ações de acordo com elementos que chegam ou saem do AmbI em geral, ou de `lugares` específicos (ver Seção 3.1.2.5).

O serviço de registro é definido como uma CAR e de instância única no ambiente. Esta é alocada junto ao repositório permitindo maior agilidade no serviço. Dada a possibilidade de replicação de base, o chaveamento do serviço para a réplica em outra unidade pode ocorrer, mas a instância do SRR manterá o mesmo RANS. Toda IAR criada no AmbI, por definição da superclasse de AR, requisita o registro de sua ODAR.

3.1.2.4 Serviço de Descoberta de Recursos

O SDR realiza buscas por quatro critérios básicos:

- Nome: retorna um ou mais ODARs compatíveis com a sub-expressão do parâmetro;
- RANS: retorna um único ODAR;
- Tipo: retorna um conjunto de ODARs relacionados ao tipo propriamente e aos subsequentes filhos da hierarquia. Por exemplo, uma busca por audiovisual retorna ODARs de computadores e de seus subtipos como *tablets* e notebooks; também é

Tabela 3.1: API do SDR

Operação	Consulta	Retorno
search(NAME, query)	Nome ou subexpressão ("lâmp")	Lista de ODARs ((ODAR(name = "lâmpadaDaSala"); ODAR(name = "lâmpadaDoQuarto")))
search(RANS, query)	RANS completo (eg., "lâmpada1.ra")	ODAR (ODAR(rans = "lâmpada1.ra"))
search(TYPE, query)	Tipo ou hierarquia (*e.g.*, Lamp.class)	Lista de ODARs ((ODAR(type = Lamp, name = "lâmpadaDaSala"); ODAR(type = Lamp, name = "lâmpadaDoQuarto"); ODAR(type = Lamp, name = "lampDaCozinha")))
search(PLACE, query)	nome do lugar (*e.g.*, "Sala")	Lista de ODARs ((ODAR(place = "Sala", name = "lâmpadaDaSala")))

possível preencher consultas com uma hierarquia, otimizando o tempo de resposta (O(log(n)));

- Local: retorna um ou mais ODARs no local especificado.

A Tabela 3.1 descreve as operações disponíveis na API do SDR para consulta a IARs.

O diretório de recursos (ver Seção 3.1.2.6) consiste de uma árvore de tipos e uma lista de `lugares`, esta última se indexa aos recursos armazenados na arvore de tipos. Nas consultas por nome, RANS e tipo são realizadas pesquisas na árvore de tipos. Nas consultas por lugar utiliza-se a lista de `lugares`.

3.1.2.5 Serviço de Localização de Recursos

O SLR opera sobre o repositório e o mapa do AmbI. Suas operações incluem: a busca por proximidade, a configuração de mapa e atualização de localização. A busca por proximidade retorna uma lista ordenada de ODARs mais próximas a coordenada cartesiana indicada pelo parâmetro, esta ordem considera os limites físicos do ambiente. Em uma casa inteligente, por exemplo, quando se busca a TV mais próxima primeiro se busca instâncias no mesmo cômodo do ponto de referência. O mapa é composto por uma lista de `lugares` e possui a área geral definida como um grande `lugar` que engloba os outros (ver definição de área na Seção 3.1.1.1). A edição consiste em redefinição de área, adição e remoção de `lugares`. A atualização de localização (ver Seção 3.1.1.1) é uma operação requisitada por agentes que atualizam seu posicionamento, logo necessitam atualizar esta

informação no mapa centralizado no repositório, como ilustrado na Figura 3.7).

A IPGAP, descrita na na Seção 2.4, é uma aplicação específica que utiliza as funções do SLR, principalmente a configuração de mapa. No caso da IPGAP, o mapa é editado visualmente, depois as coordenadas dos `lugares` e dimensões do mapa são repassadas para o SLR por meio da função de configuração (*setMap()*). Como o SLR recebe atualizações de posicionamento de todas as instâncias, a IPGAP se subscreve a esse evento para atualizar o posicionamento dos recursos representados na sua GUI.

3.1.2.6 Repositório de Recursos

O repositório de recursos é base de dados do AmbI manipulado pelo *framework* proposto. Nele existem duas estruturas de armazenamento:

- Diretório de recursos: é estruturado segundo a hierarquia de tipos dos recursos registrados, conforme é ilustrado na Figura 3.9. A estrutura se assemelha a uma árvore de diretórios de sistema operacional, sendo a hierarquia representante do caminho (*path*) e os ODARs dos arquivos. Quando ODARs representando uma nova hierarquia de tipos são registrados, esta estrutura é atualizada anexando o novo caminho. Complementarmente, a remoção de todos ODARs de uma hierarquia ocasiona a remoção do respectivo caminho no diretório.

- Mapa: é uma lista de indexação de ODARs a locais. Como visto na Seção 3.1.2.2, a informação de local é armazenada no ODAR, este dado é utilizado para indicar em qual entrada da lista será anexado. Cada entrada do mapa possui um ODAR do agente de `lugar` anexado.

O diretório de recursos permite melhor desempenho em consultas por tipo, e o mapa auxilia na melhora de consultas por localização. O esquema de rotulação é favorável na atribuição de vários tipos a uma mesma entidade, algo inviável na estrutura de diretórios. Esse esquema é utilizado, por exemplo, pelo sistema de e-mail Gmail para rotular mensagens. Por outro lado, as consultas teriam uma complexidade de tempo maior por acessar um tipo em uma lista (O(n)), ao invés de ser em uma árvore (O(log(n)), o que ocasiona uma perda de eficiência em ambientes de maior dimensão, com grande dinamicidade. Além disso, a hierarquia de tipos simplificada segue a de classes imposta por linguagens amplamente difundidas como Java.

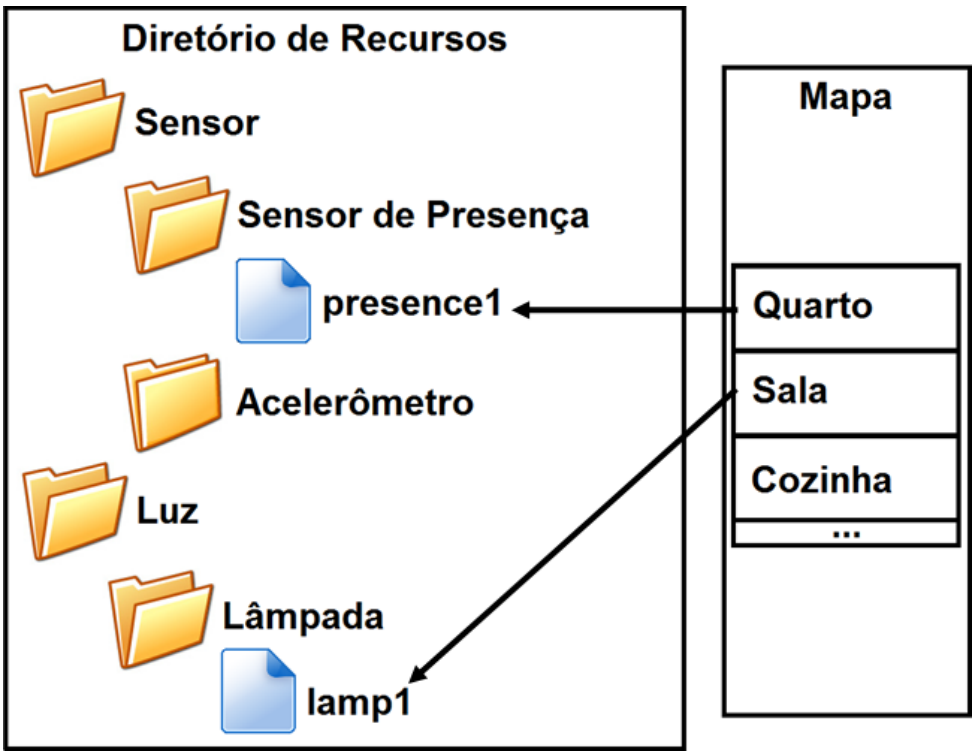

Figura 3.9: Estrutura interna do repositório de recursos

3.2 Domínios de Segurança em Ambientes Inteligentes

A segurança intrínseca da rede adotada (Wi-Fi) fornece uma base, confinando as informações na rede do ambiente. Para a comunicação entre diferentes ambientes são usados os recursos de segurança típicos da Internet. Entretanto, há requisitos de segurança interna ao AmbI que devem ser considerados para estabelecer interações seguras entre agentes. Por esse motivo é proposto o Serviço de Gerenciamento de Segurança (SGS) que complementa o SGAR definindo domínios de seguranças e grupos de usuários. Nossa proposta para o SGS aborda aspectos conceituais, atendendo requisitos em alto nível, sendo as questões em mais baixo nível definidas de acordo com a plataforma utilizada.

Na computação ubíqua, qualquer interação homem-máquina deve ser a mais imperceptível possível. Entretanto, quando abordada a questão da segurança, são discutidos meios de autenticação de usuários através de interações explicitas como: digitar uma senha, impressão digital, reconhecimento de retina ocular, etc. Outros modos menos intrusivos vem progredindo, como o reconhecimento facial, porém isto agride questões de privacidade. No *framework* buscou-se desenvolver soluções que independam de tecnologia específica, como a proposta para o contexto de localização. Nesta seção será abordada uma proposta de segurança mostrando as adaptações necessárias para agregar esta carac-

terística ao ambiente inteligente. Técnicas como as descritas podem ser usadas quando requeridas em uma determinada aplicação.

A Figura 3.10 ilustra o relacionamento das entidades de segurança com as restantes do *framework*. A solução consiste de duas aplicações: uma gerenciadora de credenciais de usuários e outra de domínios, que constituem o serviço de segurança (SGS) na camada do *middleware*. A aplicação gerenciadora de credencial é utilizada pelo administrador do ambiente para delegar autorizações para outros usuários da casa. A aplicação gerenciadora de domínios é utilizada para editar o escopo de AUbis e IARs pertencentes a cada domínio. Um domínio pode representar um conjunto de aplicações agrupando uma característica (*e.g.*, domínio de saúde, domínio de vigilância, domínio multimídia), ou uma restrição de recursos (*e.g.*, domínio adulto, domínio visitante, domínio livre). O SGS permite que modificações de segurança sejam efetivadas no nível de *middleware*.

Estas entidades são descritas em detalhes nas próximas seções. Na Seção 3.2.3 é descrito o Gerenciador de credenciais incluindo a definição de autoridades para usuários. Na Seção 3.2.1 é apresentado o Gerenciador de domínios, aplicação que manipula diretamente o SGS. E a Seção 3.2.2 complementa a abordagem de segurança descrevendo o incremento provocado em módulos básicos do *framework*.

3.2.1 Gerenciador de domínios

O Gerenciador de domínios pode ser utilizado por qualquer usuário, mas apenas o administrador pode gerenciar todos os domínios. A Figura 3.11 ilustra o processo de gerenciamento para um usuário qualquer. Esta aplicação verifica a autoridade do usuário e obtém, através do SGS, a lista de domínios permitidos. Cada domínio da lista possui os dados das IARs que podem ser copiados, movidos, ou deletados entre os domínios. No caso da deleção, se não houver registros em outros domínios isso ocasiona na remoção de registro da instância. Consequentemente, através do gerenciador de domínios é possível a remoção de instâncias do ambiente inteligente. Após a manipulação da lista pelo administrador, as alterações são encaminhadas para SGS que efetiva as mudanças no repositório central.

3.2.2 Domínios em aplicações e agentes

Quando não se deseja as qualidades de segurança interna, o funcionamento da arquitetura original não é alterado com a inclusão do SGS na camada do *middleware*. O SGS é mais um bloco da construção de uma sistema ubíquo inteligente que, quando utilizado, permite

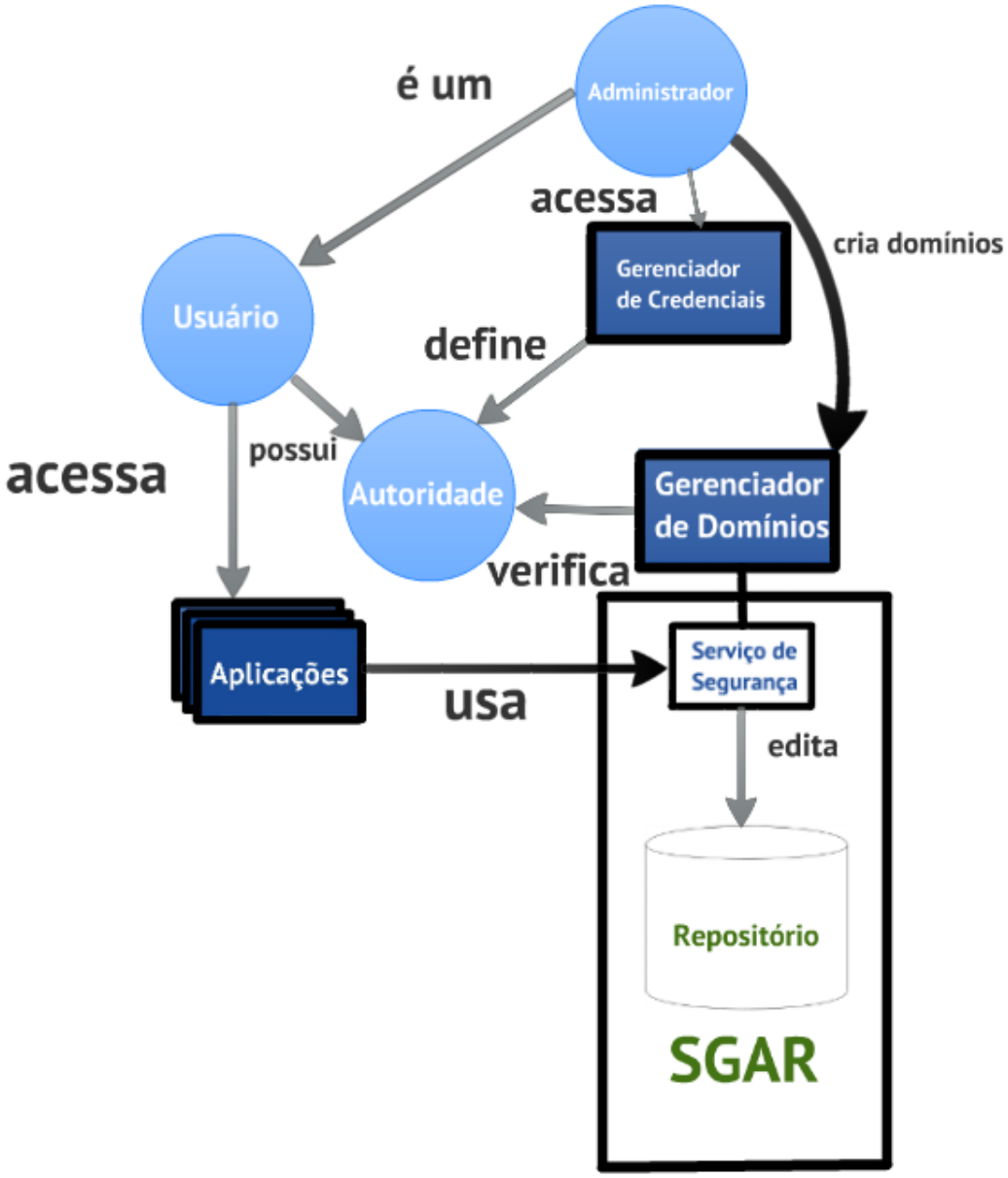

Figura 3.10: Visão geral de segurança aplicada a arquitetura

agregar a característica de segurança, e só neste caso provoca alterações no comportamento e adaptações na estrutura a nível de *middleware* e de aplicações. Nesta Seção são definidas as alterações agregadas por este serviço.

Inicialmente, há o domínio público que não necessita de autenticação para ser utilizado pelas aplicações. A partir do momento que o administrador cria o primeiro domínio personalizado do ambiente, uma estrutura auxiliar surge para agrupar os dados pertencentes ao domínio criado. Então a partir disso, o gerenciador de domínios pode proceder na edição de domínios. Quando uma instância é deslocada do domínio público para outro, esta passa a ser exclusiva ao novo domínio, quando copiada permanece pública, e

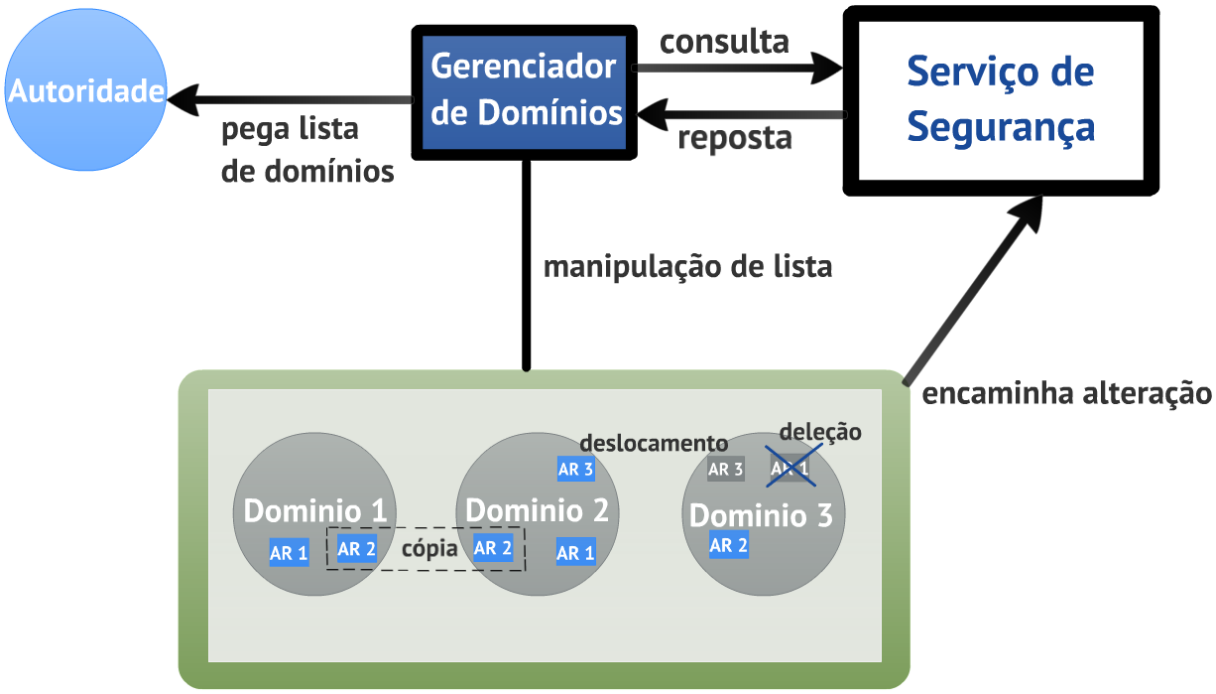

Figura 3.11: Processo de gerenciamento de domínios pela aplicação

quando deletada é removida do repositório. Ou seja, se o administrador desejar restringir o acesso a um conjunto de recursos deverá proceder no deslocamento, quando desejar restringir todos, deverá deslocá-los para os domínios criados, logo só usuários autenticados devidamente no *Daemon* de comunicação (ver Figura 3.1) poderão acessá-los.

A Figura 3.12 representa um repositório com um domínio público e outro criado pelo administrador. O diretório de recursos em ambos domínios mantém a sua estrutura em árvore. Em comparação com a Figura 3.9, o ODAR **lamp1** é deslocado do Domínio público para o Domínio 1. No exemplo, o caminho luz\lâmpada foi removido do Domínio público devido a **lamp1** ser a única a pertencer a esta hierarquia, caso contrário a hierarquia seria mantida. Se houver alguma instância do tipo luz não deslocada, apenas este tipo da hierarquia é mantido, e o restante do caminho (lâmpada) é removido. Cada ODAR carrega a respectiva hierarquia de tipo consigo, com isso é possível remontá-la no novo domínio, facilitando a geração da subárvore.

Generalizando, o deslocamento é uma sequência de remoção no domínio origem, e criação no domínio destino (incluindo a hierarquia). Quando ocorre uma remoção de instância verifica-se a existência de filhos (subclasses) e de outras instâncias na classe, caso tenha, a classe e respectivas super e subclasses são mantidas, caso contrário é removida a classe. No caso de remoção da classe, faz-se a mesma verificação para a superclasse, e assim sucessivamente.

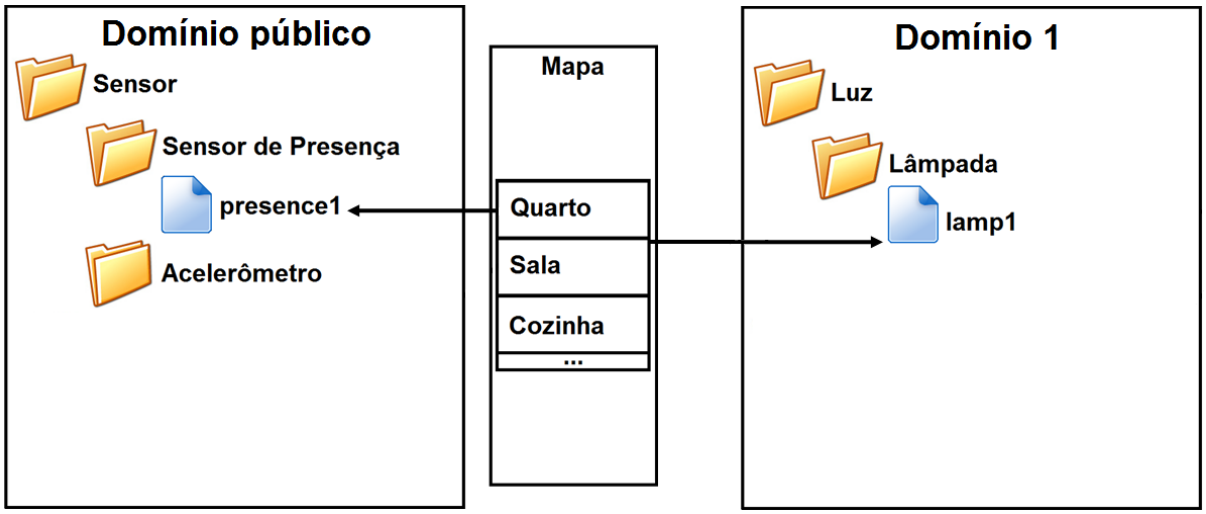

Figura 3.12: Repositório um domínio público e outro privado

3.2.2.1 Especificação de domínios em aplicações

Dado que agentes podem ser criados e instanciados pelas AUbis, estas também devem especificar quais são seus domínios, isto só é efetuado no momento de sua execução no ambiente. Existem duas maneiras básicas de configurar uma aplicação no ambiente: através (i) da IPGAP e (ii) da API do *framework*. No caso (i), é possível especificar domínios, autorizados pelo usuário autenticado, a qualquer aplicação, pois a IPGAP permite sua visualização e controle remoto até de aplicações instaladas em dispositivos sem interfaces. No caso (ii), o programador possui funções básicas que permitem expor uma interface de escolha de domínio. A exposição da interface pode ser inviável caso o dispositivo de implantação da aplicação não possua interface de tela compatível como, por exemplo, a *touchscreen*. Portanto, é reforçada a importância da IPGAP na aplicação de segurança no ambiente, pois através desta aplicação o usuário pode se autenticar, ter sua autoridade averiguada e especificar a aplicações que lhe são permitidas. É importante ressaltar que uma aplicação quando executada sem especificação de domínio, pertence ao domínio público.

AUbis podem instanciar novos agentes, os quais herdam todos os seus domínios se estes não forem explicitamente especificados neste momento. O uso do SGAR pela aplicação fica restrito ao domínio permitido pela AUbi ou IAR cliente. Assim, apenas nos domínios permitidos o SDR consulta, o SRR registra e o SLR localiza instâncias.

3.2.3 Gerenciador de credenciais

Em uma casa inteligente habitada por uma família composta pela figura do pai, da mãe, dos filhos e dos avós, podem existir restrições que podem ser gerenciadas pelo chefe da casa. Supondo o pai o chefe da casa, este geralmente é o responsável por gerenciar as permissões de acesso aos recursos da casa e no sistema é o administrador de segurança. O administrador é um tipo privilegiado de usuário que cria, autoriza ou restringe o acesso de usuários a domínios específicos.

O administrador, o qual se autentica com senha única de administração do ambiente, é o único que pode utilizar a aplicação de gerenciamento de credenciais e possui acesso a todos os domínios. O Gerenciador de credenciais permite especificar em cima de domínios existentes a autoridade de acesso de cada usuário. A autoridade é uma estrutura que cada usuário possui que armazena os domínios permitidos. O Gerenciador de domínios utiliza essa informação para verificar domínios que podem ser manipulados. O uso de credenciais pode ser melhor compreendido através do exemplo da IPGAP. Um administrador que utiliza a IPGAP pode visualizar todos as instâncias do ambiente, um visitante só pode visualizar as contidas em domínios presentes em sua autoridade.

A Figura 3.13 ilustra um exemplo de especificação de autoridade. Neste exemplo, o administrador delega autoridade apenas ao domínio 1 para o usuário. É importante observar que a especificação de domínios obedece a teoria de conjuntos, logo uma entidade pode estar contida em mais de um domínio, como é o caso de entidades na região de interseção entre os domínios 1 e 2.

3.3 Persistência do ambiente

O armazenamento de informações relevantes sobre o estado de um AmbI é importante para manter a sua persistência. Desta forma, os serviços do *middleware*, no caso de ocorrência de falhas, podem recuperar o estado atual do ambiente. A base de dados de um AmbI é estruturada seguindo o Modelo de Entidade e Relacionamento (MER). A Figura 3.14 apresenta o modelo aplicado para persistir os ODARs no repositório de recursos. O Diretório é único para cada ambiente e é composto de Domínios sendo composto inicialmente pelo público. Também é composto por Caminhos tendo inicialmente a raiz do diretório definida. Cada Caminho, Domínio e Lugar pode ter vários recursos, para o primeiro a associação é definida pelo tipo que cada IAR representa, para o segundo a associação ocorre de acordo com as restrições de segurança impostas através do SGS, e para

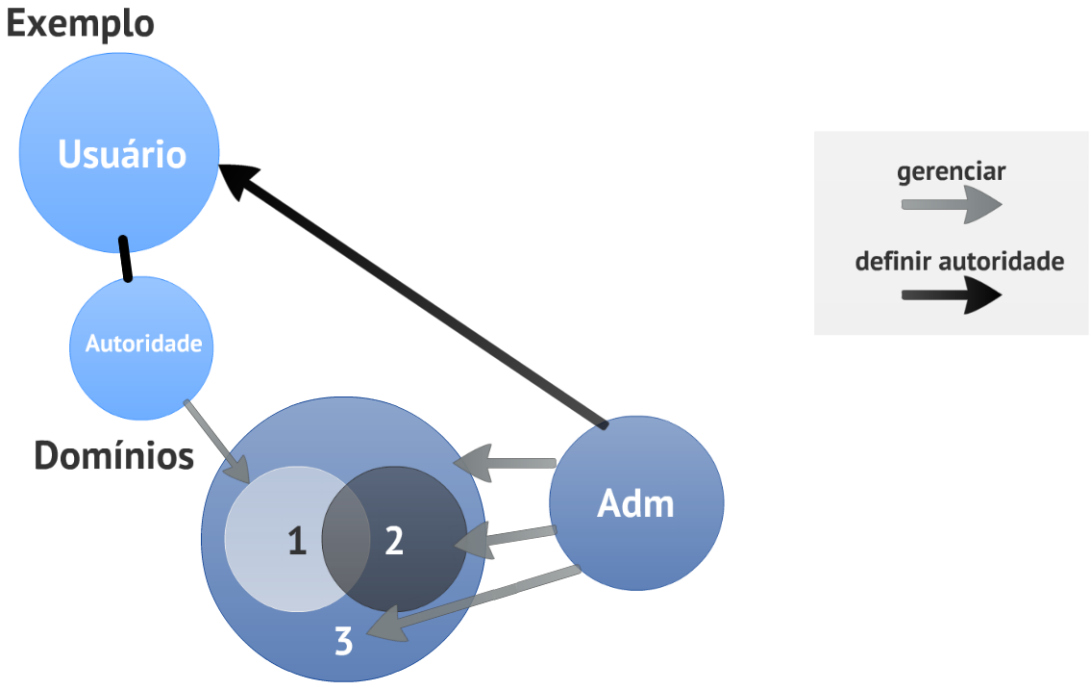

Figura 3.13: Exemplo de gerenciamento de autoridade

o último é verificada a localização física. O Mapa é composto de um ou mais `lugares`. Cada recurso contém os campos do ODAR mapeados, além das referências de Caminho, Lugar e Domínio associado. A tabela de Usuário contém o registro do Administrador, indicado por *flag* booleana, e cada registro possui os campos de autenticação (login e senha(encriptada)). Cada usuário possui certificação com um conjunto de autoridades, e o administrador possui certificação para todas. Cada autoridade pode ser utilizada por vários usuários e pode controlar um conjunto de domínios.

Quando é adicionado um Recurso, é especificado um domínio, o caminho e, se houver, o `lugar`. Quando o SRR especifica a remoção definitiva do recurso, além de remover o registro da tabela Recurso, são removidas todas as associações com Domínios. Por outro lado, quando o SGS realiza a remoção do registro em um domínio, além de remover a associação, é verificada a existência de outras, caso não haja, o registro é removido da tabela de recursos, o que garante a remoção de registro do repositório, previsto na concepção do SGS.

A base de dados é transcrita parcialmente para memória permitindo a manipulação pelo SGAR. A tabela de caminhos é utilizada para montar a estrutura do Diretório de recursos, a tabela de `lugares` é utilizada pra montar o Mapa. A indexação de IARs a ambas estruturas é captada pelo relacionamento dos registros da tabela de recursos com os registros de ambas tabelas. A estrutura de domínios, que é manipulada pelo Gerenciador

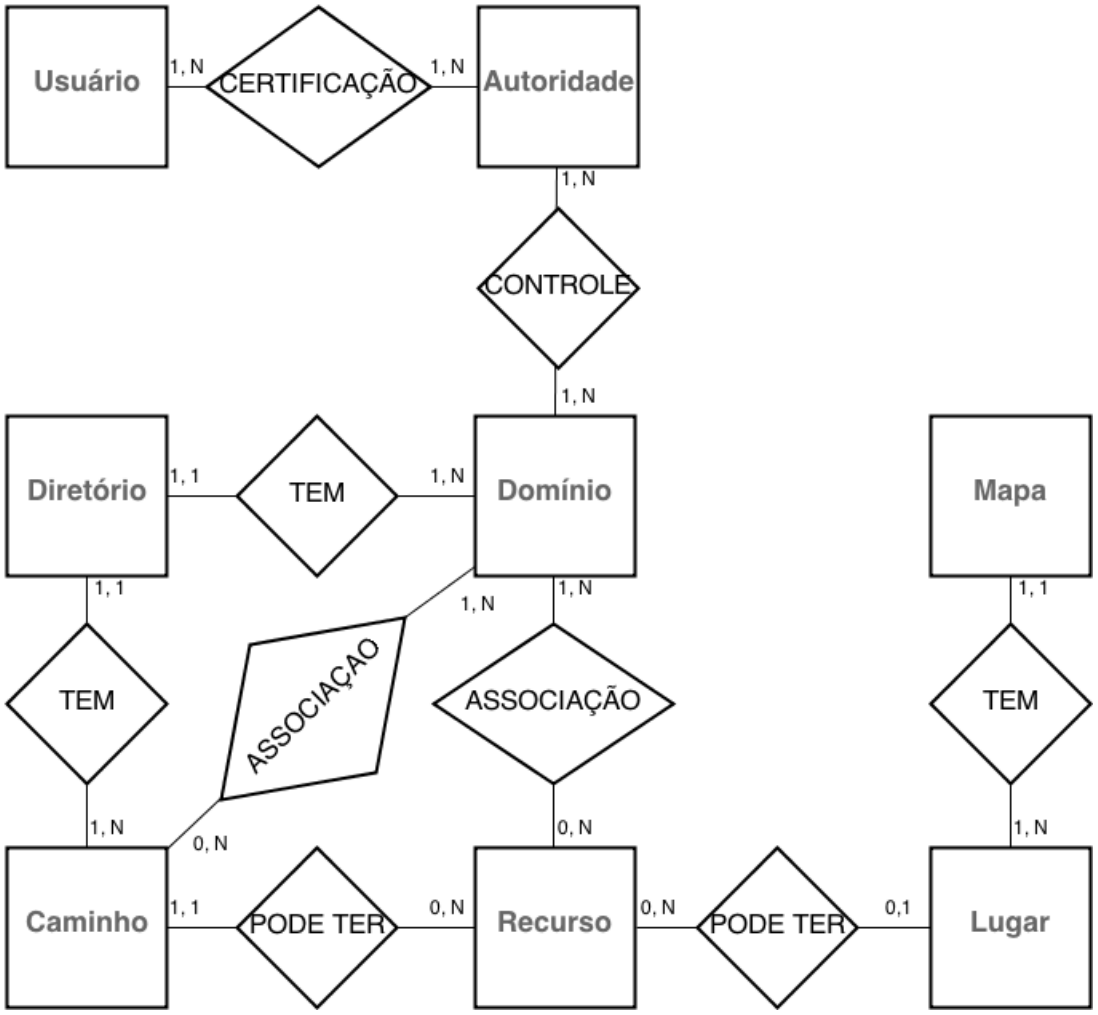

Figura 3.14: Modelo de Entidade e Relacionamento do repositório

de Domínios, armazena a associação entre a tabela de domínios e recursos. Por exemplo, "lâmpada1.ra"pode estar associado a domínios 1 ("domínio1","lâmpada1.ra") e 2 ("domínio2","lâmpada1.ra"), e domínio 1 pode ter também a "tv1.ra"("domínio1","tv1.ra"), estas informaçoes ficam armazenadas nessa tabela associativa (Domínio/Recurso). O mesmo tipo de relação ocorre entre Domínios e Caminhos, pois um usuário pode escolher associar grupos de IARs de determinado tipo (tipo define caminho) aos domínios. A estrutura de autoridade manipulada pelo Gerenciador de Credenciais é definida pela associação de controle (Autoridade/Domínio) entre a tabela de autoridades e domínios. A certificação entre usuários e autoridades é captada pela tabela associativa entre as tabelas correspondentes (Usuário/Autoridade).

Os ODARs, que estão mapeados na entidade Recurso, correspondem a maior porção de dados da base. A transcrição destes registros para memória ocorre parcialmente, de acordo com a operação recebida pelo SGAR. Em uma busca por tipo, só se transcreve

os ODARs dentro do respectivo diretório. Na busca por localização, apenas os ODARs indexados são transcritos. Em geral, um Sistema Gerenciador de Banco de Dados (SGBD) busca otimizar este processo de transcrição, logo podem ser acoplados como solução de implementação do esquema lógico apresentado nesta seção.

3.4 Conclusão do capítulo

Esse capítulo descreveu a proposta do *framework* focando na definição de componentes, suporte e segurança. A descrição detalhada do Modelo de contexto e da IPGAP é apresentado nos trabalhos [13, 5] respectivamente. Inicialmente, foi fornecida uma visão geral da arquitetura do *framework* no *middleware*. Apresentou-se o MCoD utilizado para modelar soluções sobre a arquitetura. Foi definida a estrutura do AR, e as principais classes utilizadas em AUbis, `pessoas`, `coisas` e `lugares`. A diversidade de classes impostas por AmbIs complexas motivou a construção do SGAR. Este suporte manipula ODARs que são representações de IARs no repositório. Esta manipulação ocorre através dos serviços SDR, SRR e SLR. O SDR permite a descoberta de ODARs, o SRR promove a adição e remoção e o SLR dispõe funcionalidades de localização. Concluída a definição de suporte é detalhado o repositório de recursos. O diretório de recursos armazena os ODARs em uma estrutura de árvore seguindo a hierarquização de tipos representados por estas entidades. O mapa indexa os ODARs dos respectivos `lugares` e dos agentes localizados neles.

Em complemento ao suporte é apresentado o SGS que através das aplicações de gerenciamento de credenciais e de domínios agrega segurança interna ao AmbI e seus ocupantes. O gerenciador de credenciais é utilizado apenas pelo administrador, este é responsável por delegar autoridades de acesso a domínios aos usuários. O mesmo administrador é responsável por criar domínios através do gerenciador de domínios, nesta aplicação qualquer usuário pode editar domínios dentro de sua autoridade. Foi concluído que o uso de segurança provoca mudanças no repositório, pois este passa a possuir uma cópia de subdiretório para cada domínio.

As informações do ambiente são persistidas em uma base dados seguindo o modelo de entidade e relacionamento (MER). A persistência ocorre mapeando as estruturas do SGAR para o modelo MER. Os ODARs tem seus atributos armazenados diretamente na tabela de recursos, a hierarquia de tipos é armazenada na tabela de caminhos, os espaços definidos no mapa são armazenados na tabela de `lugares`. As informações manipuladas

pelos gerenciadores de credenciais e domínios são armazenadas nas entidades associativas de certificação e controle respectivamente.

Capítulo 4

Exemplos de Aplicações

Como descrito ao longo do Capítulo 3, o *framework* reúne características e desafios os quais o torna único. Então para avaliar a qualidade e contribuição do trabalho é proposto o uso de questões de competência, como definido em [18]. Adotamos metologia similar a descrita em [18] para avaliar nossa proposta de *framework*. Estas questões representam requisitos de aplicações que devem atendidos utilizando os conceitos e mecanismos associados ao *framework* proposto. Segundo consta em [18], quatro etapas são requeridas para avaliação por questões de competência: (i) definição dos requisitos ontológicos em forma de questões que a proposta deve responder; (ii) definir terminologia da ontologia (elementos como objetos, atributos e relacionamentos) especificando a linguagem de expressão dos requisitos requeridos pela aplicação; (iii) usar a terminologia para representar a solução da aplicação; e (iv) testar a solução em termo das competências citadas, provando formalmente sua eficácia. No contexto desta dissertação, a avaliação do *framework* substitui cada etapa da seguinte forma: em (i) as questões são elaboradas textualmente, em (ii) são definidas instâncias de MCoD representando as questões de competência, em (iii) é proposta uma resolução representada pelo MCoD da aplicação utilizada como exemplo, e em (iv) estas instâncias de competência (definidas na etapa (ii)) são executadas em um cenário de uso da aplicação.

Existem três grandes questões que o *framework* de desenvolvimento de aplicações ubíquas se propõe a solucionar:

- *a heterogeneidade de dispositivos*: é resolvido essencialmente na parte do *middleware* que inclui o SGAR, repositório e agentes;
- *a variedade de informações de contexto e serviços*: é resolvido na proposta do serviço de contexto [13];

- *a disponibilidade de recursos*: é resolvido na proposta da IPGAP [5].

As propostas de serviço de contexto e IPGAP, usufruem de entidades no nível de *middleware* para proporcionar as funcionalidades específicas. O serviço de contexto utiliza o mecanismo de *publish-subscribe* para coletar informações de contexto, as CARs como referência informacional para indicar os elementos alvo de coleta, e o SGAR para acessar as IARs para efetivar estas aquisições. A IPGAP utiliza *publish-subscribe* para atualizar a visualização do estado de suas entidades e o SGAR para instanciar entidades virtuais, emuladas e encontrar as reais para efetuar a conexão.

Este trabalho concentra-se nas questões que envolvem soluções apresentadas na camada de *middleware*. Para cumprir a primeira etapa da avaliação (i - elaboração textual), são enumeradas as seguintes questões:

1. Questões de suporte
   (a) Qualquer agente que se comunica com o servidor do ambiente pode ser registrado?
   (b) Toda a informação de um agente está acessível, quando registrado, para consulta?
2. Questões de aquisição de contexto
   (a) Qualquer agente registrado pode ser acessado de acordo com suas propriedades?
   (b) Um agente pode ser atualizado sobre qualquer evento e mudança de estado de um outro?
3. Questões de localização
   (a) Qualquer agente pode descobrir onde um AR está localizado?
   (b) Qualquer agente pode saber quando alguém entrou ou saiu de um espaço definido do ambiente?
   (c) Qualquer agente pode saber quem entrou ou saiu de um espaço definido do ambiente?
   (d) Qualquer agente pode saber quais são os agentes mais próximos de um ponto de referência específico?
4. Questões de segurança

(a) Qualquer usuário do *framework* e com acesso ao servidor do ambiente possui acesso a agentes de domínio público sem necessitar se autenticar?

(b) Um administrador do ambiente pode restringir acesso a qualquer agente específico do ambiente?

(c) Qualquer usuário autenticado pode configurar domínios seguros de interação entre agentes desde que tenha permissão para atuar neles?

(d) Aplicações ubíquas podem obter acesso a outros domínios desde que o usuário autenticado nelas possua autorização para manipulá-los?

No restante deste capítulo serão apresentadas as etapas (ii), (iii) e (iv) para demonstrar a utilização e validar, através da implementação de aplicações ubíquas, as questões de competência apresentadas. A avaliação destas competências é separada em quatro aspectos que caracterizam sistemas ubíquos em ambientes inteligentes: subscrição a eventos (Seção 4.1), sensibilidade a localização e mobilidade (Seção 4.2) e sensibilidade ao contexto complexo (Seção 4.3). Neste último, será abordada a interação entre diferentes aplicações formando uma outra aplicação que explora todas as competências abordadas envolvendo questões de segurança.

As aplicações descritas neste capítulo, foram implementadas e executadas sobre a plataforma Android, através do projeto denominado *SmartAndroid*. Os detalhes sobre este projeto encontram-se no Apêndice A.

4.1 Subscrição a eventos

Nesta seção serão exploradas as questões 1 e 2, relacionadas ao suporte (SGAR) e a aquisição de contexto, através do desenvolvimento de um jogo da velha ubíquo. Esta aplicação é uma adaptação de uma versão simples na plataforma Android, que executa em apenas uma tela, para uma versão *multiplayer*, onde para uma mesma partida cada jogador possui um dispositivo individual para indicar os movimentos efetuados. O objetivo desta adaptação demonstra também a capacidade do *framework* em facilitar a programação de sistemas distribuídos para aplicações inteligentes.

Cumprindo a etapa (ii), as formulações para solucionar individualmente as Questões de 1.a até 2.b são descritas antes de apresentar a solução no cenário de aplicação. O servidor do ambiente, que possui o SGAR da camada de *middleware*, possui um endereço bem conhecido na rede, isto permite que qualquer agente programado através do *framework* se

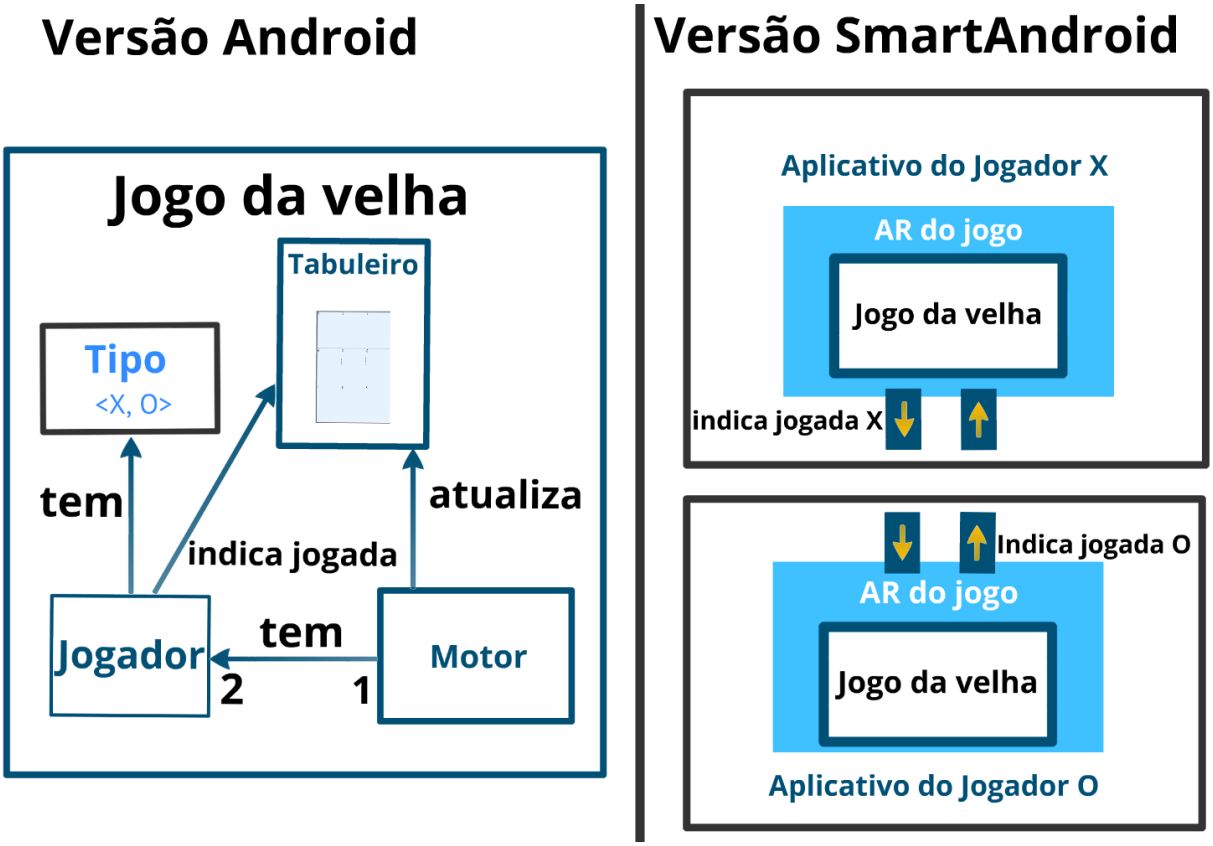

Figura 4.1: Comparação entre versão simples e versão ubíqua

comunique com esta unidade e requisite os serviços do SGAR. Incluído no suporte está o SRR, o qual provê o serviço de registro, resolvendo a Questão 1.a. A Figura 4.2 ilustra as etapas da resolução genérica para a Questão 1.b. O SDR permite a consulta por nome, tipo e localização (etapa 1), que retorna uma lista de ODARs que atende os critérios estabelecidos (etapa 2). Através do RANS de cada registro é possível acessar, por meio de comunicação remota (etapa 3), a IAR e obter suas propriedades específicas (etapa 4), resolvendo a Questão 1.b. Ao permitir esta acessibilidade, referências a agentes podem ser obtidas de acordo com as propriedades, basta consultá-las em sua instância original através de sua identificação (nome, local ou tipo), resolvendo a Questão 2.a. A Questão 2.b é resolvida usando, após descoberta a IAR desejada, a subscrição em qualquer informação desejada (VCs), como foi descrito na Seção 2.5 e ilustrado na Figura 2.2.

Cumprindo a etapa (iii) é proposta a solução da aplicação do jogo da velha ubíquo. A Figura 4.1 destaca a diferença entre a versão simples e a ubíqua. Na versão ubíqua houve a inclusão de um **AR do jogo** que tem a função de escutar as jogadas do adversário e publicar jogadas do usuário. Para possibilitar esta interação entre jogadores, é utilizado o SRR para registrar o **AR do jogo** que representa cada jogador (Questão 1.a), e o SDR para consultar sobre a existência ou não de um adversário no ambiente (Questão 1.b),

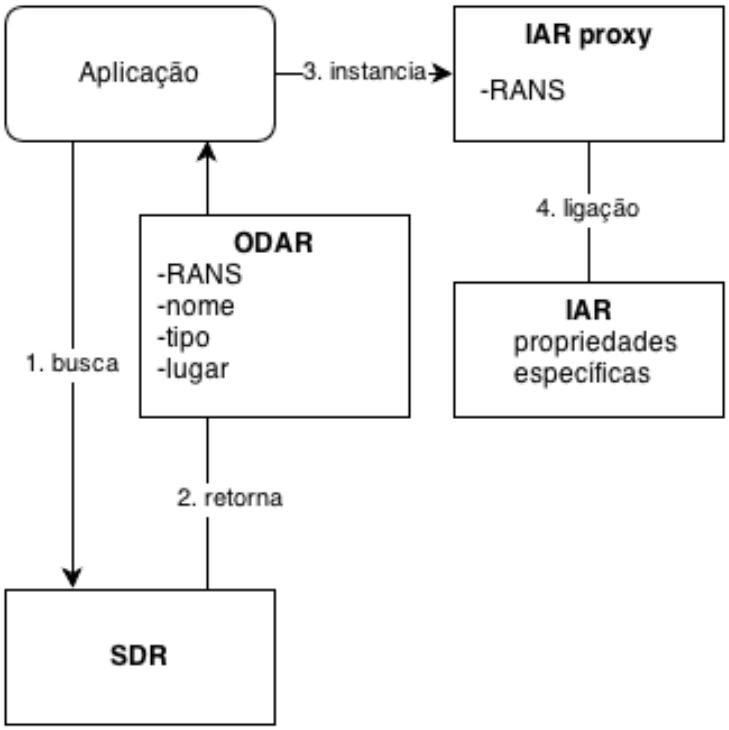

Figura 4.2: Etapas de resolução da Questão 1.b

realizando consulta por tipo **AR do jogo**. Caso não encontre nenhum no ambiente o agente se declara representante do jogador **X**, caso encontre apenas um então representa jogador **O**, caso haja mais de um, então, o agente se declara como espectador. Para que um agente receba atualização sobre jogadas este, no caso de jogador **O**, deve se subscrever ao seu adversário (jogador **X**) e remotamente, utilizando seu *proxy*, fazê-lo se subscrever de volta. Assim, jogador **X** aparece na lista de interessados do jogador **O** e vice e versa. O espectador aparece na lista de interessados de ambos. Consequentemente, quem está na lista de interessados é notificado sobre novas jogadas do agente que está com o turno. A aplicação utiliza proxy dos agentes para realizar as subscrições mútuas, através de comunicação remota realiza a subscrição a instância original (adiciona entrada com RANS e variável alvo na lista de interessados da IAR).

Portanto, as soluções formuladas das Questões 1.a a 2.b podem ser executadas usando um exemplo de uso do jogo da velha ubíquo. Para concluir a etapa (iv) da avalição é utilizado um cenário onde o exemplo conta com três participantes, jogador **X**, jogador **O**, e espectador, cada um com seu dispositivo conectado ao servidor do ambiente. A solução da Questão 1.a é utilizada quando um jogador **X** se registra através de seu **AR do jogo** no ambiente, bem como o jogador **O** e o espectador. A competência de 1.b é utilizada quando os três participantes usam o SDR para consultar pelo tipo **AR do jogo**, o que permite o acesso informações das instâncias retornadas deste tipo. Porém esta informação

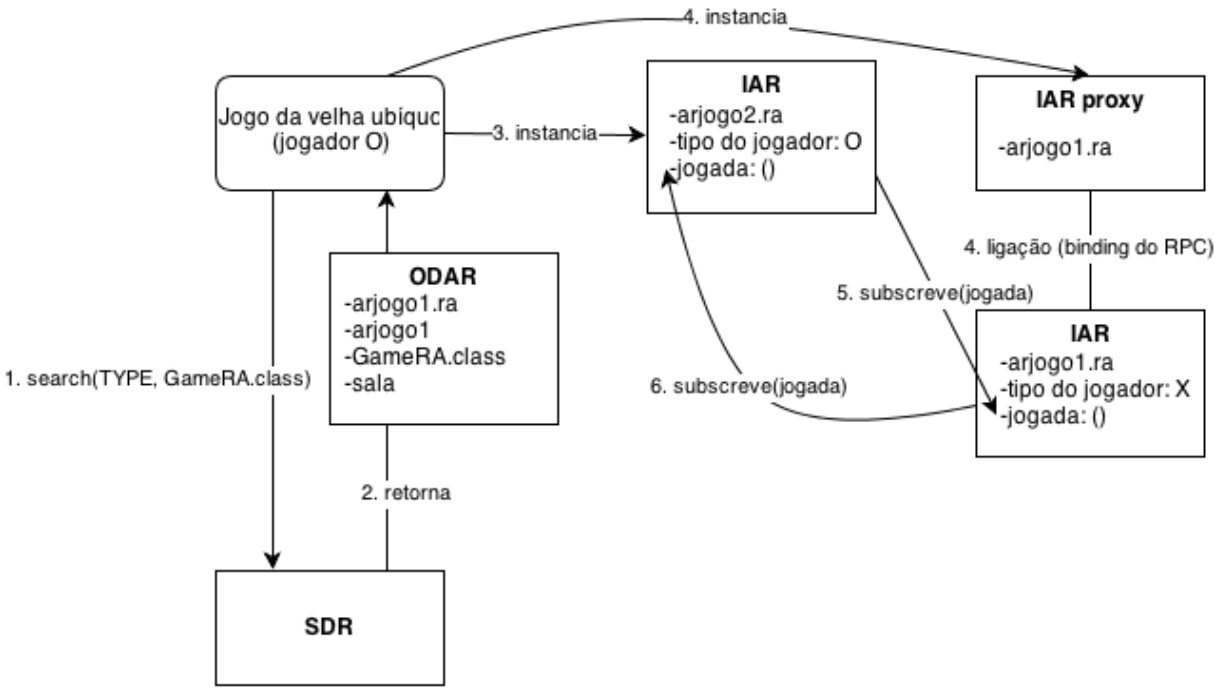

Figura 4.3: Etapas de resolução da Questão 2.b no cenário do jogo da velha

é utilizada apenas para cada participante saber o número de agentes do jogo. A Questão 2.a é caracterizada quando, um agente deseja configurar o tipo do jogador, logo, ao saber o número de participantes, um estado é atribuído. A Figura 4.3 apresenta as etapas de exercício da competência de 2.b. A partir do momento que o jogador **O** sabe que há apenas um participante (jogador **X**) (1. search(TYPE, GameRA.class)), este procede na subscrição da jogada do jogador X (5. subscreve(jogada)) e requisita ao jogador X (através de seu *proxy*) a subscrição nas suas jogadas (etapa 6). O espectador ao perceber a presença de dois jogadores subscreve-se a estes.

4.2 Sensibilidade a localização e mobilidade

Esta seção cobre a Questão 3 através de dois exemplos: primeiramente abordando localização física através do aplicação de controle inteligente de iluminação (*Smart Light Controler* - SmartLiC) e depois avaliando a questão da mobilidade através do jogo de esconde-esconde ubíquo.

4.2.1 Controle inteligente de iluminação

A aplicação SmartLiC é proposta objetivando o uso inteligente de aparelhos de iluminação em residências visando a economia no consumo de energia. Estes aparelhos são

responsáveis pela maior faixa de consumo em ambientes residenciais. A aplicação utiliza sensores de presença para identificar a localização de pessoas na casa e para decidir em que locais lâmpadas devem estar acesas e em quais devem estar apagadas. Isto para prevenir o esquecimento de luzes acesas.

Nesta seção, a etapa (ii) formula as Questões 3.a, 3.b e 3.c. para esta aplicação. O ODAR contém a informação de localização, e, através do SDR, consultando por algum atributo identificador como nome, tipo ou a própria localização, este dado pode ser obtido como resposta para um conjunto de ARs, logo resolve-se a questão 3.a obtendo localização de qualquer instância. As instâncias de agentes da classe lugar possuem dois eventos: entrada e saída; quando um desses dois eventos ocorre, IARs interessadas no lugar são notificadas (quando), resolvendo Questão 3.b, e, ao serem notificadas, obtêm a informação da CAR do agente que provocou o evento (quem), resolvendo a Questão 3.c. As resoluções desta etapas são descritas em detalhes na Seção 3.1.1.1. E a Figura 3.8 ilustra as etapas das questões 3.b e 3.c,

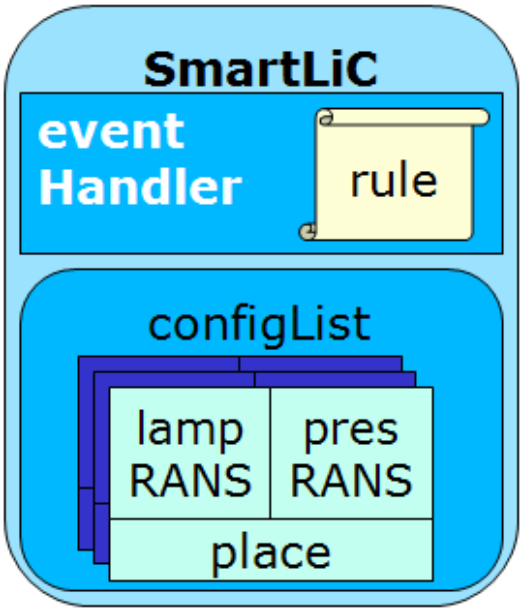

Figura 4.4: Estrutura do SmartLiC

Cumprindo a etapa (iii), é apresentada a estrutura do SmartLiC na Figura 4.4. Para que esta aplicação funcione é preciso instanciar na casa um IC com a regra (**rule**): "se lâmpada ligada e pessoa ausente durante x unidades de tempo então desliga lâmpada"; que pode ser aplicada arbitrariamente aos cômodos da casa conforme for desejado. Antes de executar os ICs é preciso configurá-los, para isto consulta-se o mapa da casa, através do SLR, e obtém-se todas as instâncias de lugar, e para cada lugar consulta-se por referências de instâncias da classe (tipo) lâmpada e sensor de presença e as armazena como elemento

da lista de configurações (**configList**), que possui a RANS da lâmpada (**lampRANS**), do sensor (**presRANS**) e o local (**place**) que estes representam. Cada configuração deve ter o IC subscrito nas respectivas VCs booleanas de ligado da lâmpada e de presença do sensor. Com todas as variáveis subscritas, o IC é iniciado, e então passa a escutar os eventos de mudanças de estado (através do **eventHandler**). Quando um evento chega, é verificada a configuração que ele representa, e assim, o restante das variáveis necessárias são obtidas para verificar a validade da regra. Nesse momento o temporizador é iniciado e, quando o tempo da condição é satisfeito, é realizada a notificação de ação de desligar a lâmpada para o atuador do cômodo representado pela configuração selecionada. É importante ressaltar que as preferências do usuário devem ser priorizadas, este pode desejar desativar a ação de desligamento ou configurar outra. Esta aplicação pode também evoluir para um simulador de presença de moradores, para que isto ocorra, as etapas adicionais de desenvolvimento incluem: armazenamento de histórico de ações (ligamento e desligamento de lâmpadas pelos moradores), aprendizagem e simulação de rotina.

Em relação a etapa (iv), imagina-se um cenário em que há uma pessoa andando pela casa, e que sai do quarto para sala esquecendo a luz ligada. A Figura 4.5 apresenta as etapas de resolução das questões 3.b e 3.c seguindo este cenário. Na inicialização da aplicação, todas as lâmpadas são descobertas a partir de uma consulta ao mapa da casa, logo a competência 3.a é satisfeita, pois a aplicação pode obter todas as lâmpadas da casa através de uma busca por tipo (*search(TYPE, Place.class)*) e obter as respectivas localizações a partir dos ODARs retornados. Ao sair do quarto e esquecer a luz ligada, o agente da pessoa ativa o evento de atualização de localização (ver Figura 3.7), este evento ativa outros dois eventos de duas IARs de lugar: o sair do quarto (5. evento(saída)), e o entrar da sala. O IC subscrito ao quarto (4. subscreve(saída)) recebe o evento e verifica, através da busca por local do SDR, se há outra pessoa presente no cômodo, se não houver atualiza VC de presença como "não ocupada"e executa a regra. Quando condição se mantém satisfeita durante tempo definido, um atuador registrado na ação disparada pela regra desliga a lâmpada do quarto, resolvendo-se a questão 3.b. A competência 3.c é satisfeita também, pois o evento notifica quem saiu do quarto (6. notifica(pessoa.ra)).

4.2.2 Jogo da prenda

Esta aplicação tem por objetivo comprovar a competência exposta em 3.d. O objetivo do jogo é encontrar uma prenda com localização oculta no ambiente. O jogador tem a disposição o mapa da casa por onde pode se mover para encontrar a prenda, a medida que

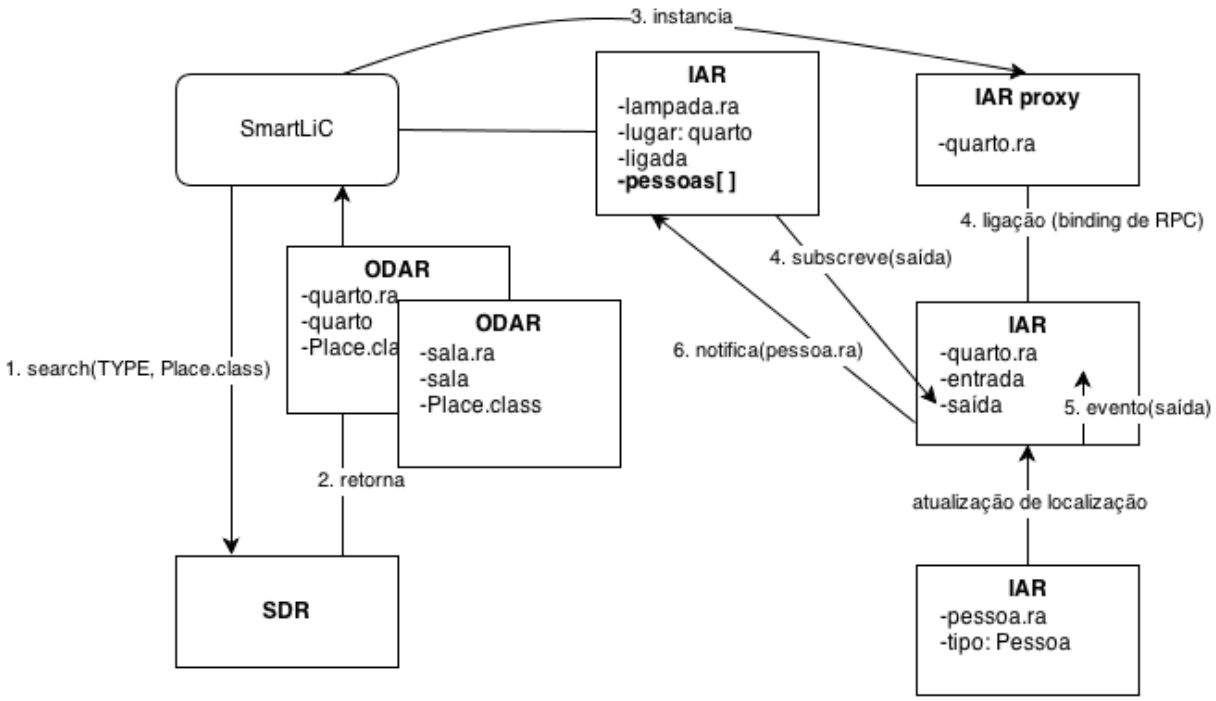

Figura 4.5: Etapas de resolução das questões 3.b e 3.c no cenário do SmartLiC

a movimentação ocorre, o jogo informa a proximidade subjetiva da pessoa com a prenda (através do número de pontos no texto da mensagem), até que esta seja encontrada. O jogo permite a participação de múltiplos jogadores por ambiente, ao obter um vencedor a prenda é reposicionada para a nova partida. O jogo ocorre em ambiente simulado, utilizando a IPGAP para simular a movimentação de avatares das pessoas participantes.

O SLR possui a função de busca pelo critério de proximidade, que retorna a lista ordenada das instâncias mais próximas ao posicionamento indicado, resolvendo a Questão 3.d para etapa (ii). O jogo da prenda é composto por dois agentes: o da pessoa e o da prenda; e utiliza uma aplicação de simulação de movimento, para indicar a movimentação da pessoa no mapa e mostrar a mensagem de proximidade com a prenda. Cada participante deve ter um simulador, onde o primeiro participante além de registrar um agente simulado de pessoa no ambiente, executa o registro e a seleção oculta de localização da prenda no mapa, que atribui o posicionamento do agente da prenda. Para saber se um jogador é o primeiro, a aplicação utiliza o SDR para uma busca pelo tipo prenda, se não houver registros, isto indica que não há outros participantes. Se entrar outro jogador, por outro simulador, este apenas registra o agente da pessoa e obtêm a localização oculta da prenda registrada pelo primeiro. Após cada movimentação de um jogador, é feito o cálculo da distância euclidiana, sem considerar limitações físicas, entre o agente da pessoa e a prenda, o jogo termina quando esta distância atinge um delta mínimo. A regra de

verificação do vencedor está explicitada abaixo:

Entidades: `Pessoas, Prenda, Jogo da Prenda`

Regra: `Pessoa.posição - Prenda.posição <= delta mínimo`

Ações: `Declara vitória(Pessoa) e Encerra(Jogo da Prenda)`

Descrita a aplicação em termos do *framework* conclui-se a etapa (iii) de avaliação. Cumprindo a etapa (iv), o cenário do jogo para competência 3.d é definido por um jogador querendo saber quem está mais próximo da prenda. Este realiza a busca por proximidade do SLR e consulta os agentes do tipo pessoa mais próximo a posição da prenda.

4.3 Aplicação sensível ao contexto complexo

Nesta proposta, denomina-se contexto complexo aquele que agrupa várias informações de contexto representando uma terceira em nível mais alto. Este processo é concretizado através da proposta de agregadores em [1], e que no *framework* é viável por meio da composição de interpretadores de contexto. Quanto ao desafio da *variedade de informações de contexto e serviços*, há os cenários envolvendo um conjunto de aplicações de diversas áreas e domínios do ambiente residencial. Existem os domínios de saúde, onde propostas como SCIADS [9] o exploram e requerem certa privacidade nas informações, existem também inúmeros sistemas de vigilância que requerem identidades de ocupantes para liberar espaços e recursos, caso morador, ou restringir acesso caso invasor. Em geral, essas informações necessitam ativar os domínios de segurança internos propostos na Seção 3.2. As questões de segurança (Questão 4) propiciam diversos cenários de sensibilidade ao contexto e requerem as características de privacidade e proteção.

Nesta Seção, o grupo de questões de segurança será abordado pela Aplicação de Segurança Residencial (SegRes) que tem por objetivo manter privacidade das aplicações instanciadas no ambiente e garantir a proteção do ambiente, dos espaços internos e de recursos contra acessos por entidades indevidas. O *framework* disponibiliza o Gerenciador de domínios que permite estabelecer domínios seguros e o Gerenciador de credenciais que determina grupos de usuários que podem manipular cada domínio. O SegRes acrescenta sensibilidade ao contexto ao que é provido por estas, e desta forma permite a criação de regras de segurança envolvendo domínios e usuários. Por exemplo, uma regra em que sempre que chegar um visitante na porta, o administrador recebe a foto da pessoa, após identificada, uma autoridade pode ser atribuída para acesso a domínios específicos. Outras

regras poderiam automatizar o processo de atribuição de domínios a agentes de acordo com a respectiva classe, por exemplo, quando agentes de sensores de pressão e sensores de glicose forem registrados, estes são incluídos no domínio de saúde. Ou seja, regras que lidam com domínios de segurança e credenciais devem ser criadas pela SegRes, que exige autenticação, e dependendo da autoridade do usuário determina o escopo de atuação. Além disso, é acrescentada a funcionalidade de atribuição de nível de autoridade para cada usuário, esta funcionalidade só pode ser utilizada pelo administrador. Os níveis de autoridade podem ser configurados diversos níveis (*e.g.*, de 1 até 5), quanto maior o nível de um usuário, maior é o poderio de criação de regras em relação com outros usuários.

Na etapa (ii), a Questão 4.a é solucionada, dado que inicialmente o ambiente tem todos os seus recursos públicos e não exige autenticação, o administrador é o primeiro a se autenticar, utilizando uma chave de fábrica, e pode delimitar domínios. O Administrador é o único que pode criar novos domínios para deslocar entidades (AUbis e IARs) para domínios privados. A privacidade do domínio é configurada através do gerenciador de credenciais, que permite ao administrador indicar quais usuários podem acessá-los, resolvendo a Questão 4.b. Qualquer usuário pode utilizar o Gerenciador de domínios, mas pode apenas manipular o que é permitido por sua autoridade. Estando as entidades dentro de sua autoridade, um usuário pode deslocar, copiar e deletá-las; ou instanciar novas, resolvendo-se a Questão 4.c. Uma vez autenticado no *middleware*, o usuário pode, por meio da IPGAP acessar qualquer aplicação dentro de sua autoridade. Ao acessá-la, pode configurar novos domínios dentro de sua autoridade, permitindo acesso desta aplicação a outros domínios, o que resolve a Questão 4.d.

A aplicação é apresentada em termos do *framework*, cumprindo a etapa (iii) de avaliação. Esta aplicação possui inicialmente o serviço de autenticação de usuário (nome e senha), existindo registro relacionado a autoridade é associada e os domínios permitidos são obtidos da SGS. Cada domínio aparece na tela de usuário contendo a lista dos dados representativos de instâncias associadas, e em outra parte a lista de usuários autorizados. Além disso, em outra seção da tela há a lista de todos os usuários do ambiente. Somente o administrador pode configurar regras entre todos os usuários e domínios. Este pode configurar nível de autoridade para cada usuário, de 1 a 5. Os usuários de nível 1 não podem manipular regras envolvendo outros usuários, que não a si mesmo; já os usuários de nível 2 a 5 podem envolver apenas usuários de níveis abaixo. Por exemplo, um usuário nível 2 pode configurar uma regra em que, quando um determinado usuário de nível 1 tentar entrar num quarto, a tranca eletrônica trava a porta. Para evitar entrar em questões de conflito, qualquer regra feita por usuário de nível acima de outro envolvendo as

mesmas entidades cancela as do outro. Por exemplo, um adolescente pode configurar o exemplo anterior para uma criança (nível abaixo), mas quando um adulto (nível acima), estipular regras para esta criança, todas as regras do adolescente e de usuários de seu nível envolvendo ela são desconsideradas. A ideia é que um aplicativo disponha a visualização de todas as regras criadas no ambiente, e permita que usuários cancelem ou editem regras criadas por usuários de níveis abaixo.

Finalizando com a questão (iv), as questões são avaliadas no cenário de casa inteligente ocupado por uma família composta por pai, mãe, filho jovem, e filho criança, onde o pai é o administrador. Neste cenário, o pai deseja impedir que seus filhos entrem no seu quarto e controle recursos daquele lugar. O quarto possui trava eletrônica na porta de acesso e uma *smart TV*. Primeiramente, o pai cria o domínio do quarto incluindo a trava e a TV e dá autoridade apenas a ele e sua esposa. No SegReg define seus filhos (identificados por RFID) como usuários do tipo nível 1 e cria a seguinte regra: se (posição(nível 1) = posição(trava)) e (trava desligada) então liga(trava); o interpretador se subscreverá a posição dos filhos, e ao estado da trava do quarto. Os filhos não precisam se autenticar para usar entidades e aplicações públicas, resolvendo a Questão 4.a. O pai restringiu acesso a *smart TV* e a trava do quarto, resolvendo a Questão 4.b. A *Smart TV*, a trava e o interpretador da regra estão no mesmo domínio e podem se comunicar, resolvendo a Questão 4.c. O pai adicionou a trava ao domínio do quarto, e permitiu que a *smart TV* interaja com ela, resolvendo a questão 4.d.

4.4 Conclusão do capítulo

Boa parte das aplicações avaliadas foram implementadas na plataforma SmartAndroid, exceto pela SegRes que é avaliada conceitualmente nos termos da proposta de alto nível do SGS. O grupo de competências 1 e 2, relativas ao suporte e a aquisição de contexto respectivamente, são avaliadas no cenário do jogo da velha ubíquo. As competências do grupo 3 são explorados por duas aplicações: o SmartLiC exerce as questões 3.a, 3.b e 3.c; o jogo de esconde-esconde ubíquo exerce a Questão 3.d. E o grupo de competências de segurança (Questão 4) é avaliado pela aplicação SegRes.

Através da prova das competências relacionadas nos itens de questões 1 e 2, foi possível demonstrar como ocorre o processo de transformação de uma aplicação local em ubíqua. Para isso utilizou-se o cenário de jogo da velha. Neste processo, verificou-se que não houve a necessidade de se preocupar com questões de comunicação, e nem com identidade única

de cada agente. Para estabelecer uma ligação através do interesse através de informações de contexto (jogadas), basta saber da existência de entidades do tipo definido no jogo.

Ao provar o item 3 de questões de competências, verificou-se a capacidade em lidar com aplicações sensíveis a localização. Através do cenário do SmartLiC, foi demonstrada a capacidade de entidades reagirem ao contexto de localização de outras. Tendo em conta as dificuldades de estabelecer uma localização interna precisa, a simulação do agente da pessoa foi necessária para viabilizar essa demonstração. Além disso, ao invés de rastrear cada pessoa individualmente, a subscrição no evento de entrada de pessoas em um lugar permite a generalização da regra que gera a notificação para o agente da lâmpada. O subitem 3.d foi provado no cenário do esconde-esconde ubíquo, onde é provada a capacidade de especificar serviços de acordo com critérios como a proximidade sem passar por muitas etapas.

Finalmente, a prova do item 4 das questões de competência apresentou o potencial de prover segurança e privacidade interna ao AmbI. O uso do SegRes, através da especificação de regras relacionando usuários e domínios as outras informações de contexto, possibilitou não só demonstrar como também incrementar os serviços oferecidos pelo SGS.

Capítulo 5

Trabalhos Relacionados

Este trabalho envolve conceitos de computação ubíqua, inteligência ambiental, sensibilidade ao contexto e à localização. Alguns trabalhos abordam o desenvolvimento de *frameworks* sobre os mesmos conceitos contemplando competências diversas [19, 27, 29, 26]. Outros são mais concentrados em aspectos de computação sensível ao contexto tal como [1] que foca na definição de entidades e estruturas contextuais. E há aqueles preocupados em melhorar aspectos de localização interna [2].

Este capítulo apresenta uma visão geral de trabalhos influentes nestes conceitos e que foram referência para comparação em termos de competências atendidas. Nesta parte são pontuados aspectos que foram considerados na concepção do *framework* relacionados a cada abordagem. Além disso, é comum que boa parte dos trabalhos abordem os aspectos de sensibilidade ao contexto e prototipagem. A avaliação comparativa geral do *framework*, a partir destas referências, leva em conta a parte conceitual e minimiza comparações de performance por serem dependentes de plataformas e recursos físicos utilizados. Boa parte das propostas, utiliza fundamentos bem estabelecidos e aceitos na área de computação ubíqua.

5.1 Propostas de *frameworks* e *middlewares*

Uma solução em uma arquitetura organizada em camadas, semelhante a já apresentada neste trabalho, é proposta em [19]. Além das camadas física, de *middleware* e de aplicação, há a camada de conhecimento que tem função similar ao SGAR, e a camada de contexto que tem função similar ao Modelo de Contexto, mas atua em uma granularidade maior. Em comparação ao SGAR, a camada de conhecimento possui uma ontologia bem definida dos dispositivos e um motor de raciocínio. O motor verifica se serviços compostos, simi-

lares ao desempenhado por AUbis e ICs, estão disponíveis. Isto permite uma consistência entre o que está ativo na rede e o que está registrado no repositório. Entretanto, nosso *framework* possui uma estrutura básica definida para adaptar ontologias na categoria de AmbIs, já a consistência de base de dados é garantida por cada IAR, pois, em seu ciclo de vida básico, o seu desativamento provoca a remoção de seu registro no repositório, o que mantem o controle de consistência descentralizado. Em geral, nosso *framework* descreve mecanismos para construção e instalação de novas aplicações, algo que em [19] não é explorado. Complementando, nossa proposta oferece recursos equivalentes e compatíveis ao uso de ontologias. Desta forma, quando consolidadas, elas poderiam ser imediatamente agregadas ao *framework*. Esta agregação pode ser realizada através da criação de um agente de ontologias que tem como função armazenar e estruturar as propriedades dos agentes (tipos de VCs e OPs, vistos na Seção 2.3.2.1) que são registrados no ambiente. Assim, AUbis que desejam utilizar uma determinada ontologia realizariam consultas neste agente que funcionaria como um servidor de ontologias.

Em [26] encontramos uma proposta de *middleware* integrada ao projeto Gaia que permitem parar e re-iniciar por completo uma aplicação ubíqua. Este projeto define uma infraestrutura que permite a execução de comandos de alto nível, similares ao de um sistema operacional, em entidades de um AmbI. Nessa proposta, dois desafios são destacados: a *variedade de informações de contexto e serviços* e a *heterogeneidade de dispositivos*. O primeiro é resolvido com uma ontologia de predicados, os quais seguem um modelo parecido com o utilizado na linguagem Prolog. O segundo é resolvido pelo uso do próprio Gaia, que conta com suporte de CORBA e Jini para comunicação. Como diferencial é apresentada uma solução para conflitos entre regras de contexto através do uso de prioridades. Há também um breve suporte ao uso de mecanismos de aprendizagem de máquina.

Em [27] é detalhado o conjunto de operações de alto nível do Gaia. Em nosso *framework* há as funções apresentadas na Seção 3.1.2. Outras funções como parar, iniciar, suspender, reiniciar são definidas por cada AR e aplicação ubíqua do sistema, permitindo uma distribuição do controle de serviços do ambiente. Particularmente, no *SmartAndroid*, este ciclo de operações é provido pela classe *Service* (original do ambiente Android padrão), da qual a CAR básica herda.

Em [8] é proposta a descoberta e o monitoramento de recursos seguindo uma ontologia predefinida. Esta ontologia é definida em arquivos XML de forma adaptativa de acordo com instâncias monitoradas e o que se deseja de informação delas. Um tipo de

arquivo representa a configuração do recurso, e outro representa a consulta de informação requerida pelo serviço de contexto. O uso dessa metodologia em um ambiente inteligente foi avaliado para uma aplicação de videoconferência, por envolver adaptação da qualidade de serviço (QoS) de acordo com a banda de rede identificada. O serviço de contexto configura o arquivo de consulta para obter a informação de banda de rede disponível para determinar a qualidade do vídeo. A proposta, por utilizar ontologia, necessita de uma carga de configuração para determinar os metamodelos para a diversidade de classes de dispositivos existentes. Por exemplo, quando chega um novo modelo de TV um arquivo representando esse dispositivo deve ser criado. Em nossa proposta a adaptação pode ser simplificada configurando programaticamente um agente, o novo modelo é definido dentro da hierarquia de televisores, e com isso, herda a sua interface básica, mantendo as operações antigas em funcionamento.

Abordando implementações em plataformas específicas, o JaCa-Android [29] utiliza o esquema de captura de mudança de estado em recursos pela escuta de eventos através de *Broadcast Receivers* (ver Seção A.1). Isto é conceitualmente semelhante à comunicação por eventos apresentada na Seção 2.5, mas limitado a componentes em um mesmo dispositivo. O *SmartAndroid* realiza um mecanismo de comunicação remota para notificar eventos em outras unidades do ambiente.

Em [17], é proposta uma linguagem de programação para usuários finais em AmbI. Esta proposta utiliza o conceito de controle indireto baseado em regras. Ou seja, a proposta aborda o controle indireto de recursos utilizando sensibilidade ao contexto. Assim, como o Modelo de contexto, é implementada a lógica ECA com temporização. Nosso trabalho, incorpora a programação por usuários finais através da definição de regras usando a IPGAP. Assim, de forma menos descritiva e mais visual, o usuário pode utilizar a interface de toque para selecionar VCs de recursos representados, usá-las em uma condição de uma regra, selecionar uma OP de outro recurso para indicar a ação, e iniciar a regra.

A proposta apresentada em [7] é focada na simulação de entidades, para permitir uma avaliação do controle em sistemas ubíquos. Diferentemente do *framework*, essa proposta separa a definição de recursos real para simulado, no nível de agente. Isto porque, na prática, os códigos se diferem na parte de atuação de serviços. Quando atua-se na entidade simulada, a operação é realizada por comandos de software. Na entidade real, opera-se pensando em elementos de baixo nível ligados ao hardware, como bits de portas lógicas de um circuito integrado ao recurso. Montar uma estrutura prevendo esta separação é útil, mas dificulta o processo de descoberta e controle na camada de aplicação. Na descoberta,

deve-se indicar se há necessidade de simulado ou real, na atuação a regra deve determinar o tipo.

O artigo [30] é uma proposta com abordagem distribuída para o serviço de descoberta, o qual ocorre através de um protocolo sobre comunicação *multicast*. O módulo de descoberta possui portas especializadas para escuta, busca e manipulação. Em analogia com a nossa proposta de registro de recursos, poderia ser estipulado registro local de recursos através da porta de escuta. Porém, o custo adicional de espaço, ocasionado pela replicação de dados em repositórios locais, pode ser significativo e vir ocasionar sobrecarga na comunicação. Esta sobrecarga ocorre devido a necessidade constante de que estas bases replicadas mantenham sua consistência entre si, logo quanto mais recursos maior a sobrecarga no AmbI.

5.1.1 Sensibilidade a localização

Pessoas possuem como característica a grande mobilidade, com base nisso, em [2] é proposto um *framework* para aplicações baseadas em caminhos. É apresentado um conjunto de soluções focadas em caminhos em larga escala. Esse *framework* possui as operações de gravação, manipulação e consulta de caminho de uma pessoa definido por coordenadas captadas por GPS. Em nossa proposta a tecnologia não é abordada, mas deve-se levar em consideração mecanismos de alta precisão para espaços internos (localização *indoor*). Em busca de simplicidade, o *framework* não possui funções específicas para mobilidade, como comparação de caminhos, estas funções devem ser implementadas em CARs específicos, como o de pessoas.

5.1.2 Segurança em sistemas ubíquos

Tendo em vista as limitações de segurança na plataforma Android, [24] propõe um *framework* para interceptar interações e embutir verificações de segurança adicionais em relação as padrões do Android. Para cada aplicação esta interceptação ocorre em duas etapas: na instalação do aplicativo e na sua execução. Durante a instalação é gerado o seu provedor de políticas e durante a execução é utilizado um mediador que consulta as políticas geradas para validar a requisição e encaminhá-la para o verificador de permissões do Android que pode destinar para a aplicação alvo. Este processo ocorre em chamadas internas a uma mesma unidade de sistema Android. O esquema de aplicação de políticas de segurança em [24] complementa nossa proposta de domínios e credenciais por promo-

ver restrições a nível de serviços e operações. Porém, o tratamento é restrito a interações internas a um mesmo dispositivo Android, algo que pode ser contornado agregando as proposta de comunicações descritas na Seção 2.5.

5.2 Trabalhos anteriores

Anteriormente a este trabalho, que propõe uma solução geral, foi abordada uma aplicação ubíqua voltada para a área da saúde e de implantação em casas inteligentes. O projeto é denominado Sistema Computacional Inteligente de Assistência Domiciliar à Saúde (SCIADS). Seu protótipo foi demonstrado no Salão de Ferramentas do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), em 2010, sendo descrito em [9]. Este sistema possui diversos desafios, dentre eles está o monitoramento de pacientes em ambiente doméstico e o reconhecimento de suas atividades cotidianas. Um módulo foi desenvolvido em [23] e trata esta questão através do uso de acelerômetros no corpo do paciente para captar seus movimentos. Um esquema de aprendizagem de máquina foi utilizado para estabelecer padrões de movimentos e associá-los a atividades. E um módulo interpretador utiliza os padrões aprendidos para inferir o comportamento do paciente de acordo com os sinais recebidos. A parte de captação de sinais e interpretação de padrões foi desenvolvido sobre a plataforma Android.

A adoção do Android foi bem avaliada, o que favoreceu sua utilização como padrão para implementação dos conceitos deste *framework*. Por este motivo, um estudo ampliado foi elaborado para construção de aplicativos robustos sobre esta plataforma. Este estudo bem avançado, nos permitiu ministrar cursos para atrair o interesse de outros discentes em participar do projeto *SmartAndroid* [1]. Com a vinda de novos integrantes ao grupo do projeto é possível aumentar o escopo de aplicações ubíquas desenvolvidas e com ajuda de outras áreas, com as da engenharia, é possível estabelecer implementações robustas nível de hardware. Uma tecnologia muito utilizada pelo Laboratório Tempo é o microcontrolador *BeagleBone* que suporta a plataforma *SmartAndroid*, e com isso permite ligar os agentes a recursos reais.

Os desafios encontrados no desenvolvimento e implantação do SCIADS foram motivadores para concepção deste *framework*. O framework para AmbIs é resultante de um trabalho em grupo, no qual cada participante focou um grande desafio da computação ubíqua. O trabalho de Erthal [14], publicado no Simpósio Brasileiro de Computação Ubí-

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

qua (SBCUP) de 2013, aborda a sensibilidade ao contexto. O trabalho de Barreto [16], publicado no SBRC de 2013, trata o problema da disponibilidade e acessibilidade aos recursos através da prototipagem de aplicações ubíquas dentro do AmbI. Publicado no mesmo evento, o trabalho apresentado em [22] é base principal desta dissertação, neste são abordados a estrutura e o uso de sensibilidade ao contexto no *middleware* proposto. Esta dissertação descreve em detalhes a parte de infraestrutura e suporte para as ideias exploradas nos três trabalhos citados, os quais resumem as dissertações de mestrado de seus autores.

5.3 Conclusão do capítulo

Neste capítulo foram apresentados trabalhos que estabeleceram as bases de nossa proposta. Em [26] são tratados os desafios da (i) *heterogeneidade dos dispositivos* e da (ii) *variedade de informações de contexto e serviços*. A abordagem desse trabalho foi seguida em [19], que se diferenciou por separar, em camadas distintas, o tratamento de contexto da aprendizagem de máquina. Em [7] é estabelecido o foco no desafio da (iii) *disponibilidade de recursos*, o que complementa os desafios abordados neste trabalho. Em [17], buscou-se um equilíbrio entre flexibilidade e simplicidade no desenvolvimento, por usuários finais, de soluções ubíquas. Como diferencial, tratamos os três desafios (i, ii e iii) aliando o equilíbrio dos dois fatores, para facilitar a configuração de serviços tanto para usuários quanto para desenvolvedores.

O *framework* proposto oferece a infraestrutura básica e flexibilidade para integrar as soluções apresentadas neste capítulo. Soluções como o motor de raciocínio proposto em [19], as funções de mobilidade abordados em [2] e as políticas de segurança propostas em [24] podem ter seus requisitos atendidos através de extensões do *framework*. O *SmartAndroid* (ver A) foi utilizado para a implementação dos exemplos apresentados.

Capítulo 6

Conclusão

Este trabalho apresentou um *framework* para desenvolvimento de aplicações ubíquas em ambientes inteligentes (AmbI). Sua arquitetura foi planejada com o objetivo de integrar a computação ubíqua a ambientes inteligentes, e com isso prover serviços de sensibilidade ao contexto e prototipagem de aplicações ubíquas. O desenvolvedor, ao utilizar as ferramentas definidas pelo MCoD e o Modelo de Contexto [14], constrói aplicações que manipulam os recursos de um espaço preenchido por dispositivos diversificados. As aplicações reagem ao comportamento identificado no ambiente, utilizando-se de sensibilidade ao contexto, para prover serviços que atendem aos ocupantes e outras aplicações interessadas. Usuários comuns, que podem ser ocupantes, podem configurar regras de contexto através da interface de prototipagem (IPGAP) [16] para personalizar serviços. O conjunto de soluções proposto foi fundamental para resolver o desafio da *heterogeneidade de dispositivos*, com isso permite a integração entre serviços e conceitos, providos por meio de aplicações, a recursos físicos do ambiente. Consequentemente, possibilita a construção de soluções inovadoras voltadas para ambientes residenciais, comércios e industrias.

Contemplando a sensibilidade ao contexto, foi elaborado o Modelo de Contexto através do MCoD. Neste modelo foram incluídos dois componentes fundamentais, o interpretador de contexto (IC) utilizado para executar regras, e o atuador(AC) para aplicar os serviços. O IC escuta eventos, avalia a condições de uma ou mais regras, e notifica uma ação. O AC se subscreve a ICs, e quando recebe uma notificação realiza a operação correspondente. O Modelo de Contexto, aplica os conceitos de ECA (event-condition-action) com o adicional da temporização na condição, sendo evento, condição e temporização tratados por IC e a ação pelo atuador. O programador tem a disposição um modelo flexível, podendo construir desde serviços simples até mais complexos, por exemplo, envolvendo motor de regras e tratamento de conflitos. O usuário pode, através da IPGAP, visualizar os recursos

e respectivas variáveis (VCs) para criar suas próprias regras.

O MCoD proposto aliou simplicidade e flexibilidade para resolver questões de diversidade de recursos. O modelo incluiu elementos como ARs e o SGAR para reduzir a complexidade de integração entre o ambiente computacional e físico. O AR foi definido como módulo básico que tem suas funcionalidades fundamentais herdadas por ICs, ACs, serviços do SGAR (SDR, SRR, SLR e SGS) e outros CARs definidos por desenvolvedores. Mantendo a simplicidade, inclui-se a definição de CARs de elementos característicos em AmbIs: pessoas, objetos e lugares. Seus CARs resolvem, sem a necessidade de ICs e ACs, questões de agregação e avaliação de informações de mobilidade para pessoas, de localização para objetos, e área de ocupação e eventos de entrada e saída para lugares.

O *framework* foi concebido como uma ferramenta flexível que oferece mecanismos para manipulação de recursos, tanto para ocupantes quanto para desenvolvedores. Os possíveis riscos consequentes da manipulação são tratados pelo serviço de gerenciamento de segurança (SGS). Com o serviço, agregou-se o controle de autorização a grupos de usuários e aplicações, além de adicionar a definição de domínios privados de recursos. Foi vista como fundamental a utilização de uma interface de controle, através da IPGAP, para atribuir domínios para AUbis, principalmente para as instanciadas em dispositivos sem GUI a sua disposição. Em geral, foi possível estabelecer uma segurança interna, restringindo acesso de usuários, através de controle de credenciamento, a grupos de entidades privadas, as quais estão associados a domínios específicos.

Com o objetivo de viabilizar a proposta, foi desenvolvida a plataforma *SmartAndroid*, beneficiando-se da acessibilidade e da adaptabilidade do ecossistema Android incluindo a diversidade de dispositivos disponíveis. A implementação foi utilizada para demonstrar de forma prática a viabilidade dos conceitos cobertos pela proposta. Na avaliação das questões de competência foram apontadas pontos importantes em suporte, coleta de contexto, localização e segurança. Através da transformação de um aplicação Android em uma versão *SmartAndroid*, foram demonstradas facilidades adquiridas com o suporte e a técnica aquisição de contexto (publish-subscribe). Uma aplicação de controle de iluminação inteligente permitiu atestar praticidade no uso de contexto de localização fornecido por entidades do *framework*. Na parte de mobilidade foi elaborada uma aplicação que reage aos movimentos de ocupantes para demonstrar a viabilidade de rastreamento dos recursos (e.g, pessoas). Além disso, a capacidade de configuração de segurança foi demonstrada através de uma aplicação que permite criar regras de contexto envolvendo domínios e grupo usuários.

O trabalho utiliza no suporte conceitos básicos já fundamentados e provados em outra propostas na área de sistemas distribuídos [19, 27]. O uso de mecanismos de *publish-subscribe* e POA para comunicação baseada em eventos, e de RPC para comunicação síncrona direta, são amplamente difundidos na área de Sistemas Distribuídos. Como diferencial, os conceitos e mecanismos incluídos na proposta facilitam a criação e configuração de AmbIs personalizados provendo dinamicidade e adaptatividade para aplicações. Estes benefícios estão disponíveis para usuários que adicionalmente podem configurar regras e serviços em alto nível, através da IPGAP. Ademais, desenvolvedores podem criar novas classes de recursos e aplicativos a serem instalados em AmbIs em execução.

6.1 Trabalhos Futuros

Em geral, os desafios futuros consistem em agregar novas funcionalidades ao nível de suporte, com a preocupação de manter o equilíbrio entre flexibilidade e simplicidade para os desenvolvimento de aplicações ubíquas. Um objetivo em específico esta em desenvolver aplicações propostas em outros trabalhos do grupo do Laboratório Tempo [1]. Dentre estes trabalhos, além da IPGAP proposta em [16] (ver Seção 2.4) e da Sensibilidade ao Contexto proposta em [14] (ver Seção 2.3.1), incluem-se: a proposta de Modelagem de Linha de Produtos Dinâmicas para Aplicações Ubíquas [10], e a Gerência de Configuração em Tempo de Execução para Sistemas Autoadaptáveis[20]. O primeiro trabalho lida com mudanças arquiteturais nos sistemas ubíquos, a ideia é especificar modelos com variabilidades que permitam a adaptação de novos produtos (novas CARs e IARs, ou AUbis). Um metamodelo é proposto para gerar estas configurações a partir de contratos arquiteturais. Nosso *framework* deve ser capaz de, a partir do modelo estabelecido, garantir que essas configurações sejam implantadas em um AmbI. No trabalho apresentado em [20], os serviços de sensibilidade ao contexto no AmbI são acompanhados, armazenados e gerenciados em tempo de execução. Neste caso, o *framework* pode se beneficiar em ter o desempenho melhorado para os serviços de contexto, ao utilizar aprendizagem de máquina em cima dos dados armazenados. Assim, em uma etapa mais avançada, regras de contexto, mais comuns e pertinentes ao comportamento identificado, podem ser configuradas de forma automatizada.

Ainda existem questões em aberto na computação ubíqua que merecem mais atenção em trabalhos futuros. Dentre estas, propostas na área de economia de energia tentando, por exemplo, reduzir a notificação de mudanças de contexto. Na segurança entre intera-

[1] www.tempo.uff.br

ções é preciso refinar o que foi proposto para o nível de autorização de invocação. No suporte ao serviços de contexto, a resolução de conflitos em regras são vistas como essenciais para o funcionamento adequado dos serviços do ambiente. Para aprimoramento no fornecimento de serviços automatizados estuda-se a identificação das preferências dos usuários. Utilizando técnicas de aprendizagem de máquina padrões comportamentais podem ser identificados, e posteriormente utilizados para configurar regras e rotinas adequadas em um AmbI. Outro ponto desafiador está na integração entre dispositivos microprocessados e recursos físicos. Cada recurso físico pode exigir uma configuração elétrica e um circuito lógico específico. Uma forma de reduzir este esforço está definir classes de recursos físicos, de acordo com a configuração elétrica e eletrônica, e assim poder gerar integradores apenas para cada classe.

Uma das questões mais preocupantes em sistemas distribuídos é a segurança de comunicação entre aplicações, quando se trata de computação ubíqua esta interatividade costuma aumentar. O trabalho cobre o credenciamento de acesso de usuários a recursos de domínios privados. No entanto, há a preocupação de restringir acesso a operações específicas de cada recurso. Para promover esta solução, deve haver uma classificação de VCs e OPs (eg., VC de temperatura, VC de medida clínica, VC de segurança), como a que ocorre a nível de agente. Consequentemente, a especificação de domínios deve ser refinada para restringir a estas estruturas.

Nessa primeira abordagem, o padrão Wi-Fi foi adotado por ser popular, barato, extensível e por permitir economizar etapas de configuração e implantação. Porém existe um amplo estudo sobre economia de energia em redes de sensores, onde se afirma que grande parte do consumo de um nó é consequência da comunicação. Controle de potência no sinal, redução no número de envios de mensagens e protocolos de encaminhamentos eficientes são abordados para tratar esta questão. Quando se trata de um sistema ubíquo caracterizado por notificar mudanças em informações de contexto, avalia-se a possibilidade de controlar o número destas notificações. Focado neste controle, é preciso determinar um grau de diferença entre mudanças. Quanto maior este grau menos envios ocorrerão, porém menos precisa é a sensibilidade a mudança. Encontrar o ponto de equilíbrio entre economia e precisão é o objetivo de um sistema de economia de energia em AmbIs. A *SmartLiC* (ver Seção 4.2.1) é uma solução de economia no nível da aplicação, uma extensão que pode ser avaliada é o controle de qualquer recurso consumidor de energia. Por exemplo, a aplicação pode escalonar a ligação de dispositivos de alto consumo (*e.g.*, chuveiro elétrico, ar-condicionado) para horários de tarifação reduzida.

O projeto contempla as preferências do usuário através da GUI de criação de regras. Para agregar inteligência ao AmbI, o comportamento dos recursos devem ser armazenados no repositório como base de conhecimento, como proposta em [19], para um sistema de mineração de dados identificar padrões benéficos ao estabelecimento de serviços inteligentes. As preferências do usuário formam a sua base particular de conhecimento, que através de mineração estabelece padrões do seu comportamento, os quais são utilizados para definir os serviços. Um serviço pode estar na forma de regra de contexto que utiliza como condição um padrão relacionado e como entrada para avaliação, o comportamento corrente. A relação entre um serviço e determinados padrões pode ser identificada através de classificação por aprendizagem de máquina. Por exemplo, ao identificar que uma pessoa ao acordar no domingo (padrão) costuma ligar a televisão do quarto, verifica-se que houve o serviço *ligar*. Logo, o padrão é classificado com a ação de ligar TV, e a uma regra é criada automaticamente associando-os (se pessoa acordar e é domingo então ligue a TV). A proposta é viável, desde que sejam selecionadas as técnicas de mineração e aprendizagem de maneira adequada. Entretanto, deve haver restrições sobre até onde o sistema inteligente deve intervir. Em [23] é apresentado uma proposta concreta para o reconhecimento de atividades. Nesse trabalho, utiliza-se os dados de aceleração, captados por acelerômetros, que são treinados para estabelecer a classificação da atividade doméstica de uma pessoa. A proposta foi classificada entre os três melhores da área de Engenharias no Prêmio UFF Vasconcellos Torres de 2011.

A proposta homogeniza a interação entre agentes de diversos tipos, entretanto quando se aborda a interação entre agentes e os recursos físicos ainda existe uma lacuna, a padronização a nível de hardware. Para associar agentes a recursos é preciso utilizar microprocessadores (*e.g.*, *Arduino*, *BeagleBone*), onde executa-se o agente, com suas portas interligadas aos elementos físicos (*e.g.*, lâmpada, TV). Para atender cada operação fornecida pelo agente deve haver um código correspondente para sinalizar as portas lógicas e um circuito integrado que seja capaz de traduzir corretamente os sinais de todas as operações para o aparelho elétrico. Um esforço que ocasiona desafios e oportunidades.

Referências

[1] Abowd, G.; Dey, A.; Brown, P.; Davies, N.; Smith, M.; Steggles, P. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing* (1999), Springer, pp. 304–307.

[2] Ananthanarayanan, G.; Haridasan, M.; Mohomed, I.; Terry, D.; Thekkath, C. A. Startrack: a framework for enabling track-based applications. In *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009), ACM, pp. 207–220.

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

[4] 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.

[5] Barreto, D. Interface de prototipagem e gerenciamento de aplicações pervasivas. Dissertação de mestrado em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, (Em preparação), 2013.

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

[7] Bruneau, J.; Jouve, W.; Consel, C. Diasim: A parameterized simulator for pervasive computing applications. In *Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International* (2009), IEEE, pp. 1–10.

[8] 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.

[9] Carvalho, S.; Erthal, M.; Mareli, D.; Sztajnberg, A.; Copetti, A.; Loques, O. Monitoramento remoto de pacientes em ambiente domiciliar. *XXVIII SBRC-Salao de Ferramentas, Gramado, RS, Brasil* (2010).

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

[12] 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 (Dec. 2001), 97–166.

[13] Erthal, M. Interpretação de contexto em ambientes ubíquos inteligentes. Dissertação de mestrado em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, (Em preparação), 2013.

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

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

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

[17] García-Herranz, M.; Haya, P.; Alamán, X. Towards a ubiquitous end-user programming system for smart spaces. *Journal of Universal Computer Science 16*, 12 (2010), 1633–1649.

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

[19] 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.

[20] Heráclio, D. Gerência de Configuração em Tempo de Execução para Sistemas Autoadaptáveis. Dissertação de mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, 2013.

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

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

[23] Mareli, D. F. M. Reconhecimento de atividades em um sistema computacional pervasivo de assistência domiciliar à saúde. Monografia de conclusão de graduação, Ciência da Computação, IC-UFF, 2011.

[24] Ongtang, M.; McLaughlin, S.; Enck, W.; McDaniel, P. Semantically rich application-centric security in android. *Security and Communication Networks 5*, 6 (2012), 658–673.

[25] Park, J.; Moon, M.; Hwang, S.; Yeom, K. Cass: a context-aware simulation system for smart home. In *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on* (2007), IEEE, pp. 461–467.

[26] Ranganathan, A.; Campbell, R. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware* (2003), Springer-Verlag New York, Inc., pp. 143–161.

[27] 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. PerCom 2005. Third IEEE International Conference on* (2005), IEEE, pp. 7–16.

[28] Saha, A. A Developer's First Look At Android. *Linux For You*, January (2008), 48–50.

[29] Santi, A.; Guidi, M.; Ricci, A. Jaca-android: an agent-based platform for building smart mobile applications. *Languages, Methodologies, and Development Tools for Multi-Agent Systems* (2011), 95–114.

[30] Villanueva, F.; Villa, D.; Santofimia, M.; Moya, F.; Lopez, J. A framework for advanced home service design and management. *Consumer Electronics, IEEE Transactions on 55*, 3 (2009), 1246–1253.

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

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

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

APÊNDICE A – Desenvolvimento do SmartAndroid

Para consolidar a proposta do *framework* através de dispositivos computacionais implementou-se o projeto SmartAndroid [1]. A escolha pela plataforma Android se fundamenta em geral em sua acessibilidade e poder de absorção dos conceitos do *framework*. A flexibilidade provida permite que aplicações ubíquas executem em computadores *desktops*, *tablets*, *smartphones*, e inclusive em microprocessadores (e.g. *BeagleBone*), os quais podem ser associados a recursos físicos.

Entender o *framework* do Android e implementar aplicações seguindo seu modelo foi um dos desafios em desenvolver o projeto. Apesar de oferecer um bom suporte que facilita a programação, alguns requisitos do SmartAndroid necessitaram de um esforço maior para adequação. Requisitos como comunicação remota direta e escalável, localização física interna de recursos e múltiplas interfaces gráficas podem ser destacados.

Para permitir uma comunicação remota entre dispositivos, foi implementado o Deamon de Comunicação (Figura 3.1) que é instância única para cada unidade usuária do *framework*. Este componente inicializa uma Thread de Execução para cada comando recebido e o Despachante se encarrega de repassá-lo para a IAR e método especificado. O comando, que segue o padrão de uma requisição RPC, foi serializado seguindo a notação JSON. Para localização interna é proposta a estrutura de dados apresentada na Seção 3.1.1.1 e complementarmente há a possibilidade de acoplar a proposta do Google Interactive Spaces [2]. Esta proposta engloba as funções relacionadas a manipulação de informações do espaço físico e utiliza recursos como câmeras sensoras de mancha para captar a localização precisa de objetos e pessoas. Com esses dados à disposição, basta aplicar os conceitos definidos pelo SLR do *framework* 3.1.2.5. A IPGAP é implementada como uma forma de visualização abreviada do ambiente (Figura 2.1), e assim permite que usuário decida a interface de qual recursos deseja focar, e minimizando a necessidade de utilização de múltiplas GUI em uma tela.

[1] www.tempo.uff.br/smartandroid
[2] https://code.google.com/p/interactive-spaces/

Apesar da escolha pela plataforma Android, o *framework* pode ser adaptado para outras tecnologias e linguagens utilizando o padrão de projeto *Wrappers*. Este padrão consiste no agrupamento de elementos de interfaces distintas em uma única padronizada, que eleva as respectivas linguagens a um nível comum de uma terceira protocolada.

A.1 Arquitetura do Android

Dentre a arquitetura do Android há 4 componentes principais: *Activity*, *Service*, *Broadcast Receiver* e *Content Provider*. Estas classes tem em comum métodos representando eventos ativados pela base do sistema Android.

A *Activity* é uma classe utilizada para implementação da GUI e possui métodos representando eventos relacionados com o seu estado de visualização. Métodos como os denominados onCreate, onResume e onDestroy, representam respectivamente a inicialização, retomada, e destruição da *Activity*. Ao herdar esta classe estes métodos podem ser sobrescritos com codificação específica. No SmartAndroid, uma Activity representa a interface de uma aplicação ubíqua, tal como a da IPGAP, onde se encontram concentradas as informações de IARs de interesse e onde podem ser configuradas regras relacionadas ao seu escopo.

A classe *Service* é utilizada para especificar funcionalidades em plano de fundo em relação a uma *Activity*. Nesta classe ocorre a implementação de serviços de suporte como atualização de banco de dados, execução de arquivos multimídias, etc. No SmartAndroid esta classe é utilizada para implementar o serviço do SGAR, e construir as CARs. Assim mantém-se uma relação semelhante ao que ocorre no Android: atividades (*Activity*) manipulam serviços (*Service*) e no SmartAndroid aplicações ubíquas manipulam ARs através de funcionalidades do SGAR.

A classe *Broadcast Receiver* implementa o conceito de *publish/subscribe* para componentes Android. Em seu modelo consta o método relacionado ao evento de recebimento de notificação, o onReceive. Este componente geralmente encontra-se agregado entidades que desejam expor seus serviços para o sistema ou para outras aplicações específicas. No SmartAndroid, o *Broadcast Receiver* é agregado a estrutura básica do agente de recurso, sendo responsável por manipular eventos recebidos de outros agentes notificadores.

Para permitir acesso a base de dados a aplicações externas é utilizada a classe *Content Provider*. Nesta classe, são definidos métodos de manipulação da base de dados e a formalização do tipo da estrutura de dados (através *mymetypes*) armazenada em cada

registro. No SmartAndroid, esta classe implementa a interface de acesso ao repositório centralizado de recursos. Isto permite que aplicações externas em um mesmo dispositivo o acessem diretamente, além de possibilitar que o Deamon de Comunicação do *middleware* (ver Figura 3.3) realize uma ponte com solicitações vindas de outros dispositivos.

A exposição de dados por meio de *Content Provider* permite a integração de aplicativos externos ao servidor. Estes aplicativos podem ser dispostos em um *marketplace* [3] permitindo que tanto usuários comuns quanto programadores incrementem o ambiente com novas aplicações ubíquas.

A.2 Implantação em AmbI

Nesta seção é apresentada a implantação do SmartAndroid em um ambiente inteligente. O SmartAndroid encapsula os componentes principais do Android provendo os serviços previstos no *framework*. Como visto na Seção A.1, cada componente básico do Android foi especializado em componentes do SmartAndroid.

O SmartAndroid é um aplicativo que inicializa o SGAR, repositório e serviços de comunicação. Este aplicativo deve ser executado em uma única unidade do ambiente, deixando o *Deamon* de Comunicação em estado de espera até receber requisições de outras aplicações. Em outras unidades (clientes) constam os aplicativos instalados com a biblioteca do SmartAndroid, a qual é necessária para executar o serviço de comunicação e *proxys* dos serviços do SGAR. Assim cada unidade pode consultar sobre IARs, através do SDR, obtendo seus ODARs, cada um com o respectivo RANS, o qual é utilizado pelo *proxy* como referência de acesso.

A.3 Código da classe primitiva de Agente de Recurso

A Figura A.1 mostra as principais funcionalidades de um agente de recurso. A implementação da sua classe determina as funcionalidades básicas das entidades do *SmartAndroid*. A classe herda o ciclo de vida de *Android Services* e implementa a interface do AR, a qual é utilizada tanto para invocações locais quanto para invocações remotas aos seus métodos. No construtor da classe são passados os valores dos atributos básicos (nome, tipo, referência(rans) e posição). No evento `onCreate()` é implementado o registro do agente no AmbI. No método `registerStakeholder()` é feita a subscrição, uma

[3] http://developer.android.com/distribute/open.html

outra instância invoca esse método passando a variável de interesse e sua RANS, esses argumentos são adicionados a lista de interessados (`stakeholders`) do agente corrente. Em `notifyStakeholder()` a mudança de estado é notificada passando a identificação da variável e o seu valor. Para receber notificações sobre mudanças é definido o método abstrato `notificationHandler()` que a identificação da mudança (RANS e variável) e o novo valor. Cada classe que herda de AR deve implementar o tratamento das notificações recebidas de acordo com sua especificação. Por fim tem o método de atualização de localização que envia mudanças de posicionamento para o SLR atualizar o Mapa e notificar mudanças a possíveis interessados.

```
package br.uff.tempo.middleware.management;

import java.io.Serializable;

public abstract class ResourceAgent extends Service implements IResourceAgent, Serializable {
    public ResourceAgent(String name, String type, String rans, Position position) {
        //...
    }

    //getters and setters
    //...

    @Override
    public void onCreate() {
        //register agent
    }

    //subscription method
    @Override
    public void registerStakeholder(String method, String rai) {
        stakeholders.add(new Stakeholder(method, rai));
    }

    public void notifyStakeholders(String method, Object value) {
        for (Stakeholder stakeholder : stakeholders) {
            if (stakeholder.getMethod().equals(method) || stakeholder.getMethod().equalsIgnoreCase("all")) {
                new ResourceAgentStub(stakeholder.getRANS()).notificationHandler(this.getRANS(), method, value);
                Log.d("SmartAndroid", String.format("notifying stakeholder: %s method: %s value: %s",
                        stakeholder.getRANS(), method, value));
            }
        }
    }

    public abstract void notificationHandler(String rai, String method, Object value);

    @Override
    public void updateLocation(Position position) {
        //...
    }
}
```

Figura A.1: Código Java do Agente de Recurso

A.4 Exemplo de subscrição a eventos

No exemplo do Jogo da velha Ubíquo (ver Seção 4.1) é apresentado um exemplo de subscrição mútua de eventos. O código deste processo é apresentado na Figura A.2. No momento que a partida é criada incia-se o processo. O SDR é utilizado para pegar a lista dos jogadores (`gameList`) (Linha 3). O agente do jogador é instanciado e registrado no ambiente (Linhas 5 e 6). E se houver outro jogador, se identifica como "jogador O"(Linha

9), caso contrário se identifica como jogador X (Linha 18). Na lista houver mais de um jogador, a instância corrente se registra como espectador. Para o primeiro e o último caso é executado o processo de subscrição mútua (Linhas 11-15). Primeiramente o AR corrente pega o *proxy* do outro participante (Linha 12) e remotamente faz o outro se subscrever a jogada dele (Linha 13). E finaliza o processo se subscrevendo as jogadas do outro (Linha 14). No final da inicialização do jogo, o agente do jogo é associada ao motor do jogo para que as jogadas sejam capturadas e notificadas.

```
1  public TicTacToe() {
2      IResourceDiscovery discovery = new ResourceDiscoveryStub(IResourceDiscovery.rans);
3      List<ResourceData> gameList = discovery.search(ResourceData.TYPE, ResourceAgent.type(GameAgent.class));
4      int id = (int) Math.round(50 * Math.random());
5      gameAgent = new GameAgent("Game"+id, "Game"+id);
6      ((GameAgent) gameAgent).identify();
7      if (gameList != null){
8          if (gameList.size()==1) {
9              gameAgent.setPlayer(Player.Name.O);
10         }
11         for (ResourceData iGameAgentData : gameList) {
12             IGameAgent iGameAgent = new GameAgentStub(iGameAgentData.getRans());
13             ((GameAgentStub) iGameAgent).registerStakeholder("setMove", ((GameAgent) gameAgent).getRANS());
14             ((GameAgent) gameAgent).registerStakeholder("setMove", iGameAgentData.getRans());
15         }
16     } else
17     {
18         gameAgent.setPlayer(Player.Name.X);
19     }
20     this.game = new Game(this, gameAgent);
21     gameAgent.setGame((Game)this.game);
22 }
```

Figura A.2: Código Java do Jogo da Velha Ubíquo