Geração Automática de GUIs para Objetos Inteligentes em Dispositivos Móveis

Ercílio Gonçalves Nascimento

Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brasil

cihao@gmail.com

Abstract. *This article presents the development of a mobile application with the intent to facilitate the interaction between humans and smart objects. The system uses a genetic algorithm to build an evolutionary way graphical interfaces that offer the services provided by each smart object, and are also adaptable to the mobile screen. The software was developed on the Android platform and follows design patterns that promote code reuse in other environments of the Java language.*

Resumo. *Este artigo apresenta o desenvolvimento de um aplicativo para dispositivos móveis com a intenção de facilitar a interação entre homem e objetos inteligentes. O sistema utiliza algoritmo genético para construir de forma evolutiva interfaces gráficas que apresentem os serviços disponibilizados por cada objeto inteligente, e que sejam também adaptáveis à tela do dispositivo móvel. O software foi desenvolvido na plataforma Android e segue padrões de projeto que favoreçam o reuso de código em outros ambientes da linguagem Java.*

1. Introdução

O desenvolvimento de um sistema para dispositivos móveis capaz de gerar interfaces gráficas automáticas nas áreas da computação ubíqua e pervasiva e Internet das Coisas (IoT) e que também seja capaz de descobrir objetos inteligentes (OI) e seus serviços publicados na rede. O aplicativo desenvolvido estabelece comunicação através de requisições HTTP e utiliza Algoritmos Genéticos para gerar, de forma evolutiva, telas de interação com os Objetos Inteligentes.

Como não existe atualmente um sistema que gere, de forma automatizada, interfaces gráficas de interação com os objetos inteligentes, desenvolveu-se um aplicativo para dispositivos móveis com sistema operacional Android que auxilie nesse quesito, evitando assim o esforço de criar interfaces fixas e engessadas para cada tipo de OI que se queira manipular.

2. Objetivos

O objetivo geral e específicos deste trabalho serão apresentados nos tópicos seguintes.

2.1 Objetivo Geral

O objetivo geral deste trabalho foi desenvolver um sistema computacional para dispositivos móveis que atenda às necessidades de descoberta e apresentação dos serviços disponibilizados pelos objetos inteligentes através de *requisições HTTP* e gerar uma interface gráfica automaticamente para fácil interação do usuário com os objetos inteligentes.

2.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Compreender as características necessárias à realização de um aplicativo que descubra serviços fornecidos por objetos inteligentes nas proximidades e gere dinamicamente uma GUI para o usuário interagir com esses OI;
- Implementar o sistema proposto em dispositivos móveis para atender às necessidades em IoT;
- Testar e avaliar a implementação do sistema; e
- Documentar o desenvolvimento e os resultados.

3. Características Necessárias à Realização do Aplicativo

3.1 Dispositivos Móveis

O foco principal no desenvolvimento de controle de objetos móveis está em dispositivos móveis, já que hoje há um fácil acesso a estes aparelhos e a praticidade no desenvolvimento de aplicações pra estes sistemas computacionais do que em outros equipamentos que possuem um sistema embarcado com capacidade reduzida de processamento e memória. Alguns sistemas embarcados possuem em sua estrutura um sistema operacional embarcado, que auxilia no controle de tarefas e no desenvolvimento de regras.

3.2 ZigBee

ZigBee pode ser utilizado para estabelecer comunicação entre os objetos inteligentes deixando as soluções mais eficientes para áreas da automação residencial e comercial, gerenciamento de energia e consumo de equipamentos elétricos numa casa, entre outros (Han e Lim, 2010). ZigBee é o nome de um conjunto de protocolos sem fio de alto nível, destinados ao uso em equipamentos de baixa potência, aplicações que requerem uma comunicação segura de dados e maximização da duração da bateria. ZigBee é muito utilizado na automação doméstica.

3.3 Modbus

Modbus, segundo o site responsável pelo projeto, “é um Protocolo de comunicação de dados utilizado em sistemas de automação industrial. Criado originalmente na década de 1970, mais especificamente em 1979, pela fabricante de equipamentos Modicon”. Os objetos inteligentes implementam esse protocolo, o qual é utilizado para comunicação. Modbus conta com um formato ASCII (American Standard Code for Information Interchange) para facilitar a leitura humana.

3.4 Android e Geração Automática de GUI

A criação de aplicativos Android para controle de objetos inteligentes tem se intensificado levando em consideração o requisito imprescindível de a interface gráfica ser criada dinamicamente para atender todas as possibilidades de serviços dos objetos inteligentes. Este sistema operacional possui uma classe chamada Activity, que é responsável por toda a apresentação e eventos de uma tela nos aplicativos Android e que foi utilizada nesse trabalho.

3.5 Algoritmos Genéticos

A geração automática de interfaces gráficas para os objetos inteligentes foi feita através de uma lógica evolucionária baseada em algoritmos genéticos. Inspirado na evolução dos seres vivos, o algoritmo aplica métodos de *crossover* e mutação genética para gerar novos descendentes. Tais descendentes são avaliados e os melhores indivíduos são salvos para participarem da próxima iteração, numa simulação da seleção natural. Ao fim do algoritmo tem-se uma população de indivíduos mais adaptados para o problema em questão, que no contexto desse trabalho correspondem a telas gráficas melhor dispostas.

4. Desenvolvimento

4.1 Modularidade

O sistema proposto foi desenvolvido em módulos (pacotes) que facilitam o entendimento e a manutenção. Cada pacote é responsável por funções específicas. A modularização do sistema juntamente com outras técnicas de programação formam um fundamental conjunto de ferramentas para a elaboração de sistemas visando os aspectos de confiabilidade, legibilidade, manutenção e flexibilidade. A Figura 1 mostra a estruturação em pacotes da aplicação desenvolvida neste trabalho e suas responsabilidades.

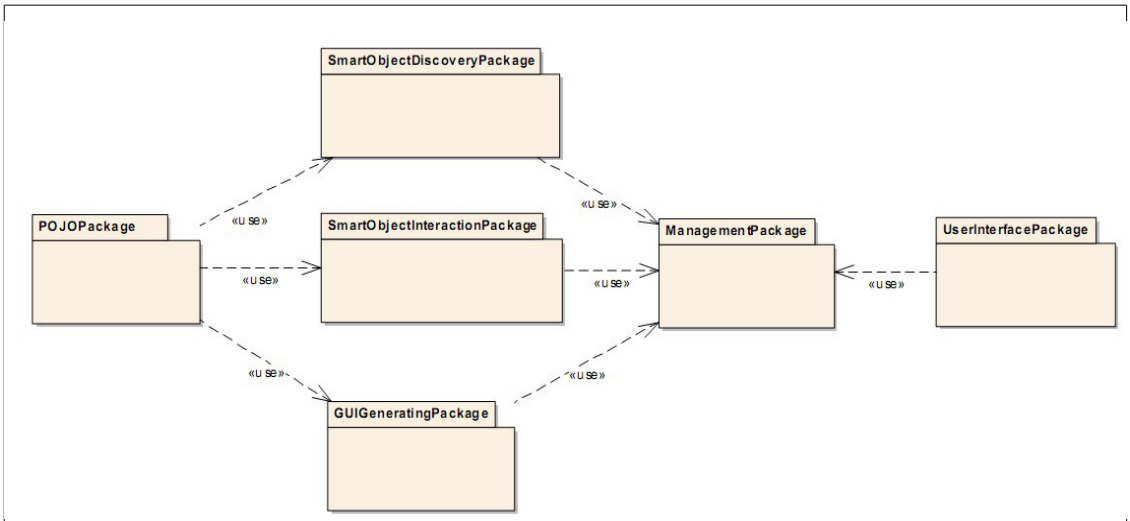

Figure 1: Diagrama de Pacotes

4.2 Módulo de Geração Automática de GUI

A função desse pacote é gerar automaticamente uma interface gráfica através de algoritmos genéticos (AG) que melhor organize os serviços disponibilizados pelo objeto inteligente.

Existe um número muito grande de classes nesse pacote, sendo assim inviável colocá-los no documento, a classe mais importante desse pacote é demonstrada pela Figura 2.

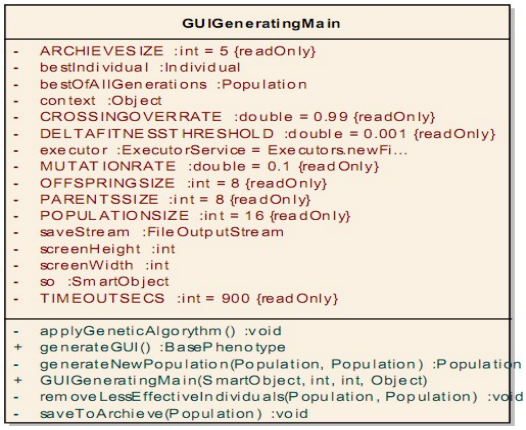

Figure 2: Classes do Módulo de Geração Automática de GUI

A classe *GUIGeneratingMain* é a principal classe desse pacote e nela é implementado a principal parte do processo evolutivo do algoritmo genético. Métodos responsáveis por seleção dos pais, *crossover*, que efetua a troca dos genes entre dois indivíduos para diversificar o código genético, *mutation*, uma seleção randômica dos indivíduos são invocados.

No contexto do sistema desenvolvido neste trabalho, cada indivíduo representa uma possível distribuição dos serviços disponibilizados pelo objeto inteligente na interface

gráfica, e uma população consiste em uma lista de indivíduos. Cada indivíduo possui seu grau de aptidão (*fitness*), que diz o quão bom o indivíduo é de acordo com a tela do dispositivo móvel utilizado.

Para cada nova população, os indivíduos sofrem recombinação genética e calculam seus graus de aptidão, o algoritmo ficará em um *loop* até que satisfaça uma condição de parada. Quando atingida, os 5 indivíduos que melhor tiverem seus graus de aptidão serão guardados para que o usuário possa alternar entre outras possíveis gerações de telas até que encontre uma que o agrade.

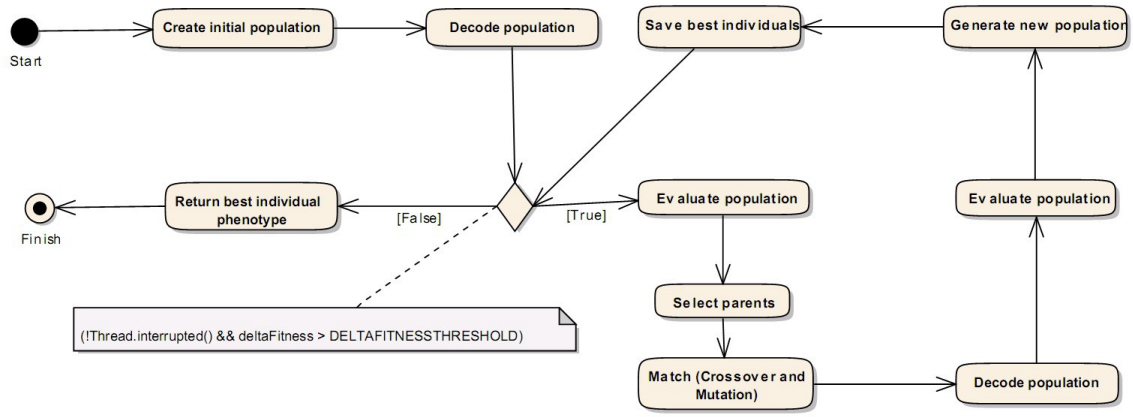

Figure 3: Diagrama de Atividades da Geração de Tela

- **Create initial population**: etapa que cria a população inicial com seus indivíduos. Cada indivíduo possui seu genótipo, fenótipo e aptidão (taxa que diz quão bom é o indivíduo) e representa uma tela com os serviços disponibilizados pelo objeto inteligente.

  Para criar uma população o método *create(SmartObject so)* é chamado e para cada serviço e parâmetro do objeto inteligente é criado um Gene com seus atributos e adicionado ao Genótipo, ao final o genótipo é adicionado ao indivíduo.

- **Decode population**: método que decodifica o genótipo do indivíduo em fenótipo. Genótipo representa a constituição genética do indivíduo, ou seja, os genes que ele possui. O fenótipo é empregado para designar as características de cada indivíduo como cor, textura entre outras. Para o presente trabalho um fenótipo é a própria tela gerada;

  Esta é a etapa que mais demanda processamento, pois consiste em transformar o genótipo em fenótipo, ou seja, criar componentes gráficos, texturas, cores, layouts, elementos de interação com o usuário, eventos de toque entre outros. O principal método da decodificação com a função *preOrderTree(Gene root, LinearLayout view)*. Essa função percorre a árvore de genes de forma recursiva e cria os componentes por eles representados. O código-fonte completo está disponível nos Apêndices do documento.

- **Decision Node**: define o ponto de parada da iteração do algoritmo evolucionário, seja por *timeout* ou pelo limite de evolução alcançado pelo algoritmo;
- **Evaluate population**: é responsável por avaliar o fenótipo de cada indivíduo da população. Um cálculo é realizado e o resultado é atribuído à aptidão (*fitness*) do indivíduo. Essa aptidão varia de 0 a 1 e quanto mais próximo de 1, melhor;

  Para se realizar o cálculo de avaliação do fenótipo primeiramente é necessário que o fenótipo não seja caracterizado como um fenótipo ruim na etapa de decodificação, e que a área em branco restante do retângulo que envolve todos os serviços do objeto inteligente seja menor que 20%. Caso o fenótipo atenda essa condição, o cálculo então é realizado através da relação entre área total dos serviços do objeto inteligente e área da tela do dispositivo hospedeiro do sistema.
- **Select parents**: etapa que tem a função de selecionar os pais para que seja feita a tarefa de *crossover*. Os pais são selecionados de acordo com suas taxas de aptidão, quanto maior a sua aptidão maior será a probabilidade de ser selecionado;

  A seleção dos pais é feita de acordo com suas aptidões: quanto maior sua aptidão, maior será sua chance de ser selecionado para reprodução. É feito um laço e um sorteio através da função *Math.random()*, da própria linguagem Java. Caso seja menor que a sua probabilidade de reprodução, o indivíduo é selecionado. Ao final, tem-se uma nova população de pais com os indivíduos sorteados, os quais serão utilizados para as próximas etapas.
- **Crossover**: responsável por efetuar a troca de material genético entre dois indivíduos, o pai e a mãe, gerando assim um outro indivíduo filho contendo material genético tanto do pai quanto da mãe. Dessa forma há uma boa variabilidade genética e uma boa chance de gerar melhores indivíduos;

  A troca de material genético implementada nessa etapa consiste em clonar o genótipo do pai e da mãe para que possam ser manipulados sem a perda de informação dos pais originais, problema clássico na programação orientada a objetos. Depois de clonados, os genes da mãe são adicionados de forma aleatória no genótipo clonado do pai. Assim temos um filho com materiais genéticos erroneamente duplicados. Para a resolução desse problema, o método preOrderTree(Gene root) é chamado e nele é feita uma chamada recursiva retirando os genes duplicados. Após essa limpeza temos um genótipo com material genético tanto do pai quanto da mãe e sem duplicidades.
- **Mutation**: etapa de mutação genética. O genótipo é submetido a uma mutação de seus genes, cada gene que representa um layout deverá ter seus atributos alterados aleatoriamente. Não há garantia de que todos os genótipos mutados sejam bons, porém se isso ocorrer eles terão suas taxas de aptidão baixas e por consequência descartados;

  A mutação é a troca de disposição dos layouts da tela. Para isso, é feito um sorteio dos genes a serem mutados e número de elementos em cada novo layout

gerado. Esse sorteio é feito utilizando novamente a função Math.random() do Java.

- **Generate new population**: método que adiciona a nova população decorrente do crossover e da mutação na população anterior, aumentado o número de indivíduos;

  A geração da nova população é feita através da adição da população gerada pelas etapas de crossover e mutation à população antiga. Depois disso os elementos são ordenados pelas piores taxas de aptidão e removidos da população.

- **Save best individuals**: última etapa do algoritmo que tem por função salvar os melhores indivíduos já gerados, evitando assim a perda de bons indivíduos e certificando de que o algoritmo correrá de forma evolutiva;

  Os melhores genótipos da população gerada são inseridos em um NavigableMap<Double, Genotype> e adicionados ao objeto inteligente representado pela classe SmartObject. Esse objeto será posteriormente persistido em disco para que possa ser reutilizado em outras interações com o usuário. O mapa contendo os melhores genótipos será utilizado para alternar entre diferentes telas sem que haja a necessidade de aplicar novamente o algoritmo genético, melhorando o desempenho do sistema. O usuário pode gerar novas telas sempre que desejar, porém as melhores telas são mantidas por facilidade ao usuário.

4.3 Demais Aplicações Desenvolvidas

A descoberta de objetos inteligentes é feita através de uma requisição HTTP para uma aplicação web JSP utilizando método POST para maior segurança, ou seja, os parâmetros não ficam visíveis na URL, apenas no corpo da mensagem. O ambiente de desenvolvimento possui o Apache Tomcat v6.0 como servidor de aplicação Java instalado e também banco de dados Oracle MySql v5.2.

O servidor de banco de dados possui todos os objetos inteligentes cadastrados com seus serviços assim como os usuários com acesso autorizado. Como ilustrado na Figura 5 da seção 3.2, o sistema desenvolvido nesse trabalho acessa o servidor de banco de dados para descoberta dos objetos inteligentes.

Para enviar os comandos de manipulação dos serviços do OI é feita uma nova requisição HTTP com método POST para um outro servidor web que, por sua vez, traduz os parâmetros capturados para a sintaxe ASCII ModBus e encaminha via porta serial para um dispositivo *gateway*. Esse dispositivo envia o comando via ZigBee para o objeto inteligente selecionado.

5 Teste e Validação

5.1 Algoritmo Genético

As figuras a seguir ilustram o ciclo completo do algoritmo genético codificado nesse trabalho. A partir de um fenótipo pai e outro mãe, é realizada a etapa de *crossover*, misturando seus materiais genéticos para a elaboração de um novo indivíduo filho. Em seguida aplica-se o algoritmo de mutação genética, o qual altera os valores dos genes (somente genes que representam layouts) do filho aleatoriamente.

As *screenshots* foram retiradas do smartphone Google Nexus 5 com tela de 5” na posição paisagem. É possível deslizar a tela em *scroll* para que o serviço “CONFIGURAR” fique visível. Os serviços ficarão totalmente visíveis ao término do ciclo, contemplando assim o propósito do algoritmo genético e desse trabalho. A Figura 4 demonstra o fenótipo do pai, à esquerda a árvore de layouts com seus serviços e ao lado a tela por ela representada.

Figura 4: Fenótipo Pai

O fenótipo da mãe possui os mesmos serviços, porém estão dispostos de forma diferente. A Figura 5 ilustra a árvore e a tela desse fenótipo.

Figura 5: Fenótipo Mãe

Após feita a etapa de *crossover*, o genótipo do filho foi criado mesclando material genético do pai com o da mãe. Os serviços “COZINHAR” e “CONFIGURAR” foram

retirados da mãe enquanto o “HIGIENIZAR” foi retirado do pai. A criação desse fenótipo pode ser melhor entendida na Figura 6.

Figura 6: Fenótipo Filho após Crossover

O ciclo se fecha aplicando o algoritmo de mutação genética no filho. Os serviços “CONFIGURAR” e “HIGIENIZAR” foram colocados lado a lado e reduzidos em largura alterando a árvore de layouts. Desta forma todos os serviços estão dimensionados no espaço disponível pelo smartphone sem que haja perda de informação e atendendo o propósito do software desenvolvido nesse trabalho. A estrutura final do fenótipo filho é representada pela Figura 7.

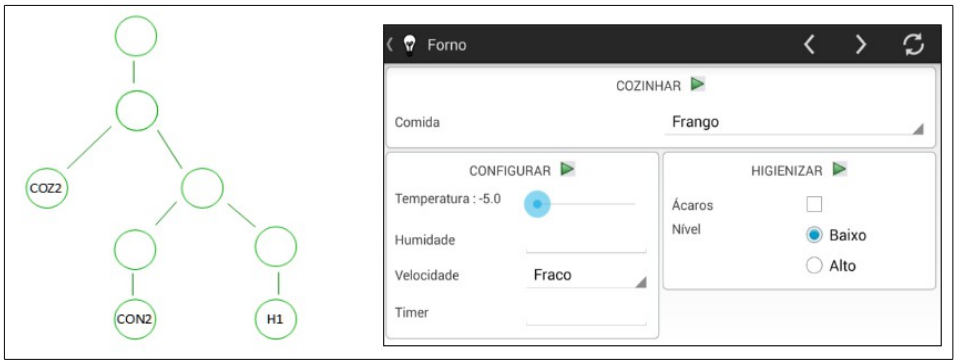

Figura 7: Fenótipo Final após Mutação

5.1 Interação com o Objeto Inteligente

Foram realizados testes com um EPOS-MOTE que simula um Ar Condicionado. Os testes respeitam a seguinte sequência:

1. Mostrar a tela com o serviço selecionado;
2. Requisitar à aplicação web SOServer que traduza os comandos recebidos para o padrão ASCII Modbus;
3. Enviar as instruções para o *gateway* via porta serial;
4. Verificar o resultado nos EPOS-MOTES.

As figuras dessa subseção representam a sequencia de aumentar e diminuir a temperatura do Ar Condicionado. O LED vermelho indica que a temperatura foi aumentada e o azul que foi diminuída.

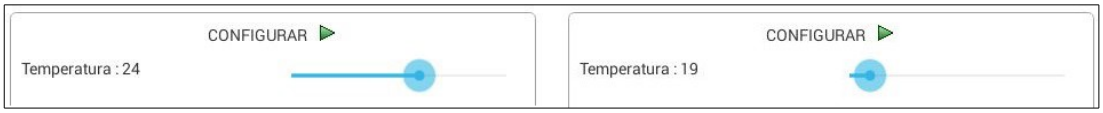

Figura 8: Serviço "Aumentar" e "Diminuir" temperatura do Ar Condicionado

Os logs são representados pela Figura 9.

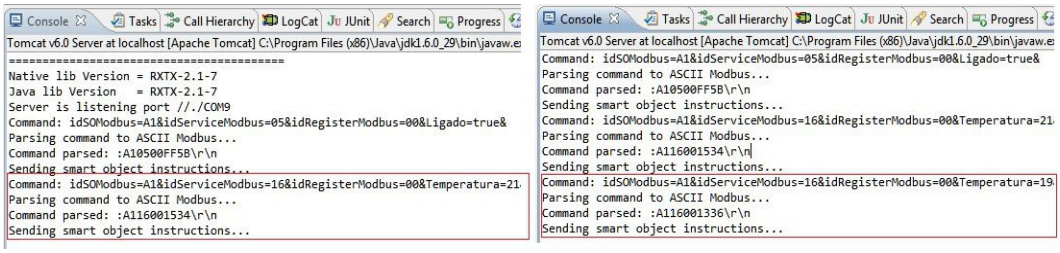

Figura 9: Logs de Aumentar e Diminuir a temperatura do Ar Condicionado

As cores do LED do Ar Condicionado indicam que os serviços foram executados com sucesso.

Figura 10: Ar Condicionado com a temperatura alterada

6 Conclusão

O presente trabalho apresentou o desenvolvimento de um sistema para dispositivos móveis capaz de gerar interfaces de interação com objetos inteligentes de forma automática, utilizando um algoritmo genético.

Com base no estudo realizado optou-se por utilizar dispositivos móveis com sistema operacional Android para fornecer a interface gráfica ao usuário, utilizar o EPOS-MOTE como equipamento embarcado de controle dos objetos inteligentes, utilizar o protocolo de comunicação HTTP para interação com os objetos inteligentes, e utilizar algoritmos genéticos para a criação automática da interface gráfica de interação com os objetos inteligentes.

O sistema desenvolvido auxilia na manipulação de objetos inteligentes centralizando toda a interação em um único aparelho; é capaz de gerar telas automáticas para qualquer

novo objeto inteligente encontrado, e dessa forma não é necessário desenvolver uma interface de controle para cada objeto inteligente, o que deve ser muito útil aos usuários. Entre os pontos fortes destaca-se que o layout das telas com maior aptidão geradas pelo algoritmo genético são salvas (tanto para o modo retrato quanto paisagem) e que o usuário pode escolher qual mais o agrada, o que não exige a geração de novas telas cada vez que o OI for acessado. Contudo, o usuário pode decidir quando gerar novas telas.

Também destaca-se que, apesar do foco deste trabalho ser a geração automática de interfaces gráficas para controle de objetos inteligentes (OI), foi desenvolvida também as funcionalidades de comunicação e acesso a objetos inteligentes reais, implementados com epos-motes, e também as aplicações de acesso ao banco de dados para a descoberta dos serviços. Assim, todo OI deve ser cadastrado em uma base de dados juntamente com seus serviços e usuários autorizados ao acesso e agora podem ser monitorados ou controlados a partir de qualquer dispositivo móvel com Android.

Referências

CASTRO, Miguel; JARA, Antonio; SKARMETA, Antonio. Smart Lighting solutions for Smart Cities. **International Conference on Advanced Information Networking and Applications Workshops**. 2013. p. 1374 – 1379.

VERMESAN, O, et al.. Strategic Research Roadmap. **Internet of Things**, Set. 2009.

WEISER, M. The Computer for the 21st Century. **Scientific American**, p. 94-104, Set. 1991.

BEIGL, M; GELLERSEN, Hans-W; SCHMIDT, A. **Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts**. 2001.

ALDRICH, F. Smart Homes: Past, Present and Future. **Inside the Smart Home**. 2003. p. 17 – 39.

MARCONDES, H, et al. EPOS: Um Sistema Operacional Portável para Sistemas Profundamente Embarcados. **III Workshop de Sistemas Operacionais**. Campo Grande, MS. Jul. 2006. p. 31 – 45.

Android, <http://developer.android.com/training/index.html>. Acessado em: 10 de Novembro de 2013. Às 16:10hs.

Internet Engineering Task Force, <http://datatracker.ietf.org/doc/draft-ietf-core-coap/>. Acessado em: 15 de Junho de 2013. Às 14:25hs.

LECHETA, R. **Google Android para Tablets**. São Paulo: Novatec, 2012. 448p.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.