Geração Automática de GUIs para Objetos Inteligentes em Dispositivos Móveis

Ercílio G. Nascimento, Bruno M. Crocomo, Rafael L. Cancian

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

{ercilio,bruno,cancian}@inf.ufsc.br

Abstract. *This paper presents the development of a mobile application with the intention of facilitating the interaction between humans and intelligent objects in the context of the Internet of things. The system uses a genetic algorithm to build GUIs in a progressive way, having the services provided by each smart object, and are also adaptable to the mobile screen. The system communicates using HTTP with a gateway that, in turn, communicates with embedded devices Epos-Motes through ModBus wireless network. Tests showed an effective generation of graphic interfaces and also functional monitoring and control on intelligent objects.*

Resumo. *Este artigo apresenta o desenvolvimento de um aplicativo para dispositivos móveis com a intenção de facilitar a interação entre homem e objetos inteligentes no contexto de internet das coisas. O sistema utiliza um 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 sistema se comunica via HTTP com um gateway que, por vez, se comunica com equipamentos embarcados Epos-Motes através de rede sem fio ModBus. Testes demonstraram uma efetiva geração de telas gráficas e também o monitoramento e atuação funcionais sobre os objetos inteligentes.*

1. Introdução

Numa rede de Internet das Coisas (IoT), os elementos ou nodos que fazem parte desta rede são chamados de Objetos Inteligentes (OI), e é através desses objetos inteligentes que a rede se comunica e a interação entre usuários e "coisas" é efetivada. Os objetos inteligentes precisam ocupar pouco espaço para que caibam nas "coisas" e também têm de ser baratos e com pouco consumo de energia elétrica. Esses objetos inteligentes, então, correspondem a sistemas computacionais embarcados, projetados para uma função específica. Neste artigo, nos referimos unicamente a Objetos Inteligentes (OI) como os equipamentos embarcados que precisam ser monitorados e/ou sobre os quais se atua.

Os OI geralmente não possuem uma interface gráfica para interação com o usuário, pois são pequenos, possuem baixa capacidade de processamento e memória, e precisam consumir pouca energia elétrica, pois geralmente estão ligados a uma bateria. A interação com esses objetos pode ocorrer de diversas formas, mas o mais comum ainda é o acesso remoto através de protocolos de comunicação de baixo nível, o que não é fácil para um usuário comum utilizar. Nos casos de acesso remoto, tais objetos disponibilizam seus serviços através da Internet, usando webservices por exemplo, o que faz necessária a utilização de aplicações externas que queiram acessar esses serviços de IoT (monitorar o estado das "coisas" ou atuar sobre elas).

Para a interação do usuário com os objetos inteligentes, os usuários normalmente desejam uma interface gráfica amigável e de fácil entendimento. Porém, segundo Izquierdo et al (2009), em geral, 50% do tempo de desenvolvimento de uma aplicação está na criação da interface gráfica, e é impraticável desenvolver uma interface gráfica para cada OI e, além disso, usuários não acessarão os OI a partir dos próprios objetos, mas a partir de seus computadores, e principalmente a partir de diferentes dispositivos móveis. Essa interação poderia ser beneficiada por um sistema que descobrisse serviços providos por Objetos Inteligentes de uma IoT e gerasse de forma dinâmica e automática uma interface gráfica (GUI) amigável para configurar e interagir com cada um dos OI encontrados, a partir do dispositivo em que esse sistema estiver executando, tipicamente um *smartphone* ou outro dispositivo móvel. Esse sistema ainda deveria ajustar os componentes da GUI (*labels, textboxes, radiobuttons, panels, etc*) para as características da tela e do dispositivo móvel em questão. Esta solução beneficiaria o usuário geral de uma IoT, como um sistema de automação residencial, que poderia utilizar um dispositivo móvel qualquer para controlar todos os equipamentos de uma rede de IoT, sem ter que implementar nenhuma GUI, e ainda poderia escolher a GUI mais adequada a cada OI.

Nesse contexto, este artigo apresenta um aplicativo que foi desenvolvido para dispositivos móveis, e que estabelece comunicação com os OI de uma rede por meio de requisições HTTP e que cria em sua tela uma interface gráfica dinâmica que se modela automaticamente de acordo com os serviços descobertos providos pelos OI e as características da tela do dispositivo móvel onde executa. A tela apresentada para cada OI é aquela considerada a mais apta por um algoritmo genético, após um processo de otimização, ou aquela escolhida pelo usuário, dentre as mais aptas. Por fim, esse sistema se comunica via HTTP com um *gateway* que, por sua vez, se comunica via rede sem fio modbus com os equipamentos (tipo EPOS-Motes) de uma rede de IoT.

2. Trabalhos Similares

Mori, Nonaka e Hassi (2010) abordam a criação de uma interface gráfica automática para 5 dispositivos de sistema audiovisual previamente cadastrados. Nesse caso, foi utilizada uma tela LCD que apresenta uma lista de equipamentos disponíveis para uso. Após selecionado, o sistema gera automaticamente, através de algoritmo genético, uma nova tela com todas as opções de manipulação possíveis separadas por páginas. Cada ícone é tratado como um gene e cada array de gene, por sua vez, é tratado como um

indivíduo. O indivíduo é o que consiste em todas as opções disponíveis para um determinado equipamento. O cálculo para a geração da interface gráfica consiste em duas etapas: (1) ocupar o menor número de espaços em branco possível em cada página e (2) gerar o menor número de páginas possível. A aplicação foi construída em um sistema embarcado, com um microcontrolador e um painel touch. Os autores conseguiram desenvolver uma aplicação que centraliza o controle dos equipamentos de áudio e vídeo, contudo, há a necessidade de cadastrar previamente os aparelhos, não possibilitando assim o controle à outros tipos de equipamentos.

Izquierdo et al (2009) criaram um *framework* para geração automática de interfaces gráficas em webservices, em que é possível importar o arquivo WSDL e para cada serviço disponível, gerar uma tela automática. Após a geração da tela, é possível que o usuário a modifique para melhor atender às necessidades e finalmente exportar a tela para um arquivo XML, o qual utilizará para integrar em alguma aplicação. O *framework* foi desenvolvido na linguagem Java em sua versão 1.5, utilizando os componentes *Swing* para GUI. Os autores conseguiram um método para gerar automaticamente telas para aplicativos que utilizam webservices. Apesar de ser bastante útil, esta prática está limitada apenas à área de sistemas webservices, não sendo possível para sistemas que utilizam outra tecnologia.

3. Desenvolvimento

3.1 Geração Automática de GUI

Com base em estudos realizados previamente, 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 EPOS-Mote foi escolhido por ser um equipamento pequeno, flexível e adequado ao cenário de IoT. Ele também foi desenvolvido pelo laboratório de pesquisa ao qual os autores estão vinculados e, portanto, está disponível, é passível de futuras adaptações e também já é utilizado noutros projetos de IoT.

A figura 1 mostra uma visão geral do ambiente desenvolvido. O sistema de geração de GUIs executa num dispositivo móvel e se comunica, via Wifi, com um repositório de serviços (em que os OI previamente registraram seus serviços). O sistema também se comunica com um servidor, no qual há um *gateway* conectado e que serve de ponto de acesso à IoT e aos Objetos Inteligentes sendo monitorados/controlados.

Neste artigo, focamos na geração de GUI, cujo objetivo é gerar automaticamente uma interface gráfica que melhor organize os serviços disponibilizados pelo objeto inteligente, o que é feito por um Algoritmo Evolucionário desenvolvido especificamente para a otimização da GUI, pois um OI pode conter dezenas de métodos com vários parâmetros cada e, portanto, a interface gráfica que permite acesso completo aos serviços desse objeto pode conter várias dezenas de elementos gráficos (*panels, radiobuttons, labels*, etc) que precisam ser adequadamente posicionados.

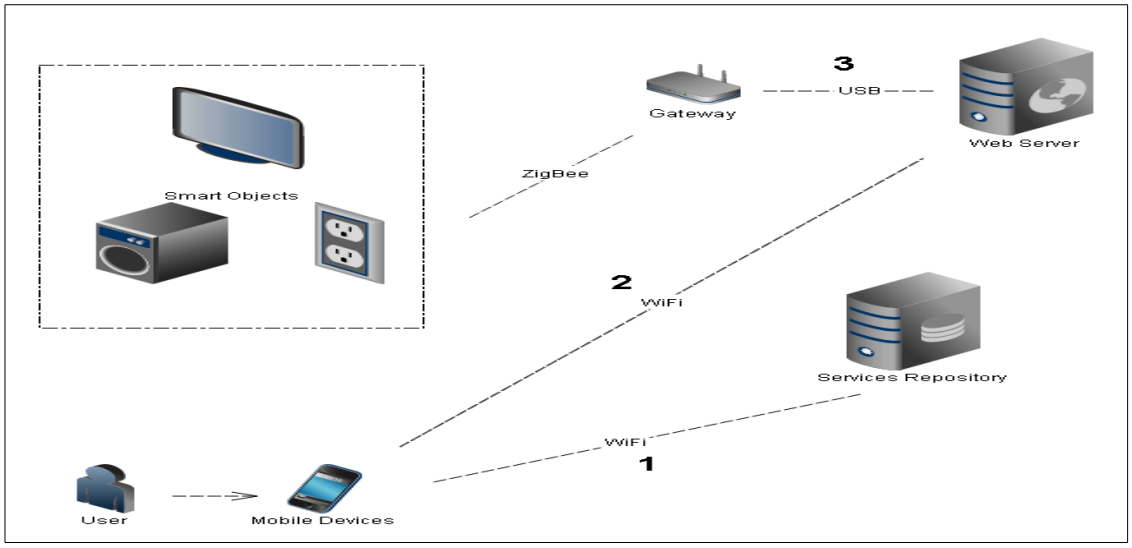

Figura 1: Visão geral do ambiente desenvolvido

No contexto do sistema desenvolvido, cada indivíduo representa uma possível distribuição gráfica dos serviços disponibilizados pelo objeto inteligente na interface, o que é conseguido por um genótipo em forma de árvore (e não linear, como é quase sempre o caso), para representar o layout dos *panels* que agregam os elementos gráficos de um serviço específico. Como exemplo, suponha um OI que controla um condicionador de ar e que tem um método "`void configurar(int temperatura)`", que corresponde a um dos serviços que esse OI executa. Esse serviço deve gerar um *panel* na GUI com título "*Configurar*", com um *label* "*Temperatura:*" e com um elemento gráfico de definição de inteiros, como um *textbox*, um *spinbox* ou um *slider*.

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 mais aptos sofrem recombinação genética (alguns ramos da árvore do genótipo são trocados), mutação genética e suas aptidões são avaliadas. O processo se repete geração após geração até que um critério de parada seja atingido: tempo máximo atingido, ou aptidões não melhoram mais. Ao final do processo de otimização, os indivíduos que tiverem os melhores graus de aptidão são guardados para que o usuário possa alternar entre outras possíveis gerações de telas até que encontre uma que o agrade.

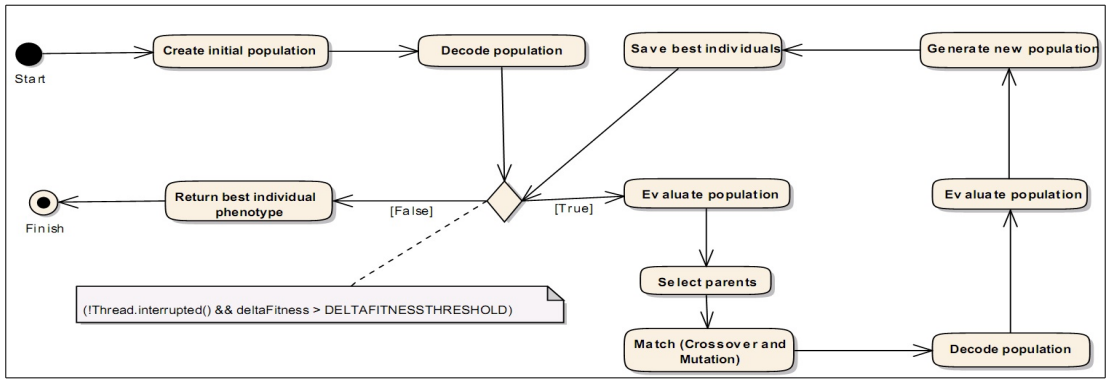

Figura 2: Diagrama de Atividades da Geração de Tela

De maneira geral, o algoritmo evolucionário implementado possui as seguintes atividades principais, das quais descrevemos apenas algumas de suas características.

Create initial population: etapa que cria a população inicial com seus indivíduos. Cada indivíduo possui seu genótipo, fenótipo e aptidão e representa uma tela com os serviços disponibilizados pelo objeto inteligente (para modos retrato e paisagem). Para criar uma população, para cada OI, para cada serviço e para cada parâmetro é criado um Gene com seus atributos e adicionado ao Genótipo, que é então adicionado ao indivíduo.

Decode population: método que decodifica o genótipo de cada indivíduo em fenótipo. No contexto do sistema desenvolvido, um fenótipo é a própria tela gerada, e mais especificamente, a disposição de seus elementos gráficos ((x1,y1),(x2,y2)) e os métodos para interagir com eles, ou seja, invocar remotamente os serviços do OI. Esta é a etapa que mais demanda processamento, pois consiste em criar componentes gráficos, texturas, cores, layouts, elementos de interação com o usuário e eventos de toque, entre outros. O principal método da decodificação percorre a árvore de genes de forma recursiva e cria os componentes gráficos (*swing*) por eles representados.

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, que varia de 0 a 1 e quanto mais próximo de 1, melhor; Sua fórmula não é apresentada aqui por limitações de espaço, mas é calculada a proporção entre o retângulo que representa a área da tela do dispositivo (tanto para modo paisagem como modo retrato) e a área que é a soma dos retângulos ocupados pelos elementos gráficos. Quanto mais próximo de 1, melhor é a disposição dos elementos na tela.

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 para reprodução. 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 selecionados (pais), gerando assim um indivíduo-filho contendo material genético de ambos os pais. A troca de material genético (que é uma estrutura de árvore) implementada nessa etapa consiste em clonar o genótipo do pai e da mãe, então os genes da mãe são adicionados de forma aleatória no genótipo clonado do pai. Assim temos um filho com parte do material genético erroneamente duplicados. Um caminhamento *preOrderTree* é realizado recursivamente, retirando os genes duplicados. Essa forma de *crossover* não foi encontrada em nenhum outro trabalho pesquisado.

Mutation: etapa em que o genótipo é submetido a uma mutação de seus genes. Cada gene que representa um layout poderá ter seus atributos (alinhamento vertical ou horizontal, posição, tamanho, etc) alterados aleatoriamente. Assim, 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. Mutação aleatória não garante que todos os genótipos sejam bons, porém os piores indivíduos têm menor probabilidade de seleção para reprodução.

Generate new population: método que adiciona a nova população decorrente do crossover e da mutação (os filhos) na população anterior, aumentado o número de indivíduos. Depois disso os elementos são ordenados pelas taxas de aptidão e os indivíduos menos aptos são 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 boas soluções ao longo das gerações e também entre execuções do aplicativo. Os melhores genótipos da população gerada são adicionados à representação computacional do objeto inteligente, que é posteriormente persistido em memória não-volátil para que possa ser reutilizado em futuras interações com o usuário. O mapa contendo os melhores genótipos também utilizado para alternar entre diferentes telas sem que haja a necessidade de aplicar novamente o algoritmo genético a cada execução, melhorando o desempenho do sistema. O usuário pode gerar novas telas sempre que desejar, porém as melhores telas são mantidas para comodidade do usuário.

3.2 Interação com Objetos Inteligentes

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 1, 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 (Epos-Mote) selecionado.

4 Teste e Validação

4.1 Geração automática de GUI

As figuras a seguir ilustram o ciclo completo do algoritmo genético desenvolvido. A partir de dois genótipos (pai e 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 3 representa o fenótipo do pai, à esquerda a árvore de layouts com seus serviços e à direita tela por ela representada. Os nodos da árvore (genótipo) podem representar layouts (verticais ou horizontais) ou os elementos gráficos (t*ext boxes, combos, radio buttoms, etc*) dentro desses layouts.

Figura 3: Fenótipo Pai

O fenótipo da mãe possui os mesmos serviços, porém estão dispostos de forma diferente. A Figura 4 ilustra a árvore e a tela desse fenótipo.

Figura 4: 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 5.

O processo completa 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 visível na tela e atendendo o propósito do software desenvolvido nesse trabalho. A estrutura final do fenótipo filho é representada pela Figura 6. Com isso, verifica-se que o processo de otimização de tela realizado pelo algoritmo evolucionário é funcional, e gera automaticamente interfaces com diferentes disposições e elementos gráficos, sendo (esse processo) adequado aos objetivos da pesquisa.

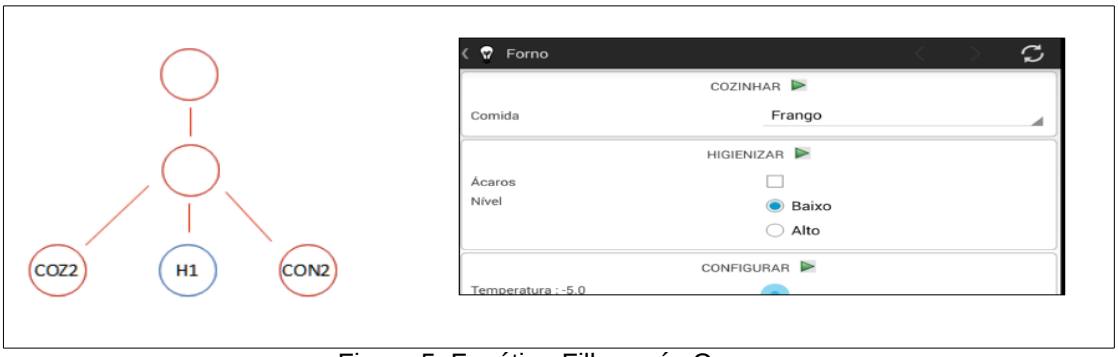

Figura 5: Fenótipo Filho após Crossover

Figura 6: Fenótipo Final após Mutação

4.2 Interação com Objetos Inteligentes

Foram realizados também diferentes testes com equipamentos reais do tipo Epos-Mote, utilizados em aplicações de IoT e redes de sensores sem fio, e que se comunicam via rede sem fio ModBus. O objetivo desses testes foi comprovar o correto monitoramento e atuação sobre os OI reais, utilizando a GUI gerada automaticamente pelo sistema. Nesse sentido, foram avaliadas principalmente questões de descoberta dos serviços e comunicação remota com servidores e com os OI, em diferentes cenários de teste. Especificamente neste artigo, apresentamos os testes realizados sobre um Epos-Mote que comandava um condicionador de ar. 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 a seguir representam a sequência de aumentar e diminuir a temperatura do condicionador de ar. Para efeitos de teste, o equipamento Epos-Mote foi desconectado do condicionador de ar real, e o acionamento foi enviado a LEDs. O LED

vermelho indica que foi comandado o aumento da temperatura, enquanto o LED azul indica que foi comandada a diminuição da temperatura.

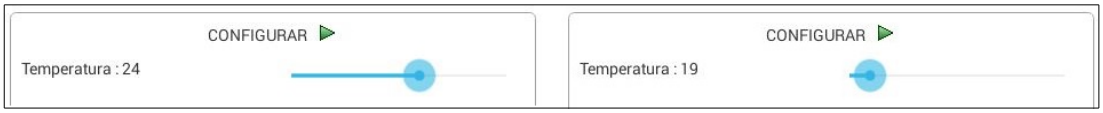

Figura 8: Serviço "Aumentar" e "Diminuir" temperatura do Ar Condicionado

Após a temperatura desejada ter sido ajustada na GUI gerada no dispositivo móvel, o comando correspondente deve ser transmitido ao OI. Parte dos logs dos comandos de envio de mensagens aos OI reais são apresentados pela Figura 9, e as mensagens ModBus são destacadas, demonstrando tanto o recebimento de requisições realizadas a partir do dispositivo móvel quanto o envio de mensagens ao gateway (Epos-mote) que dá acesso aos OI. Portanto, a comunicação sem fios com os equipamentos reais é funcional e os métodos corretos devem ser invocados.

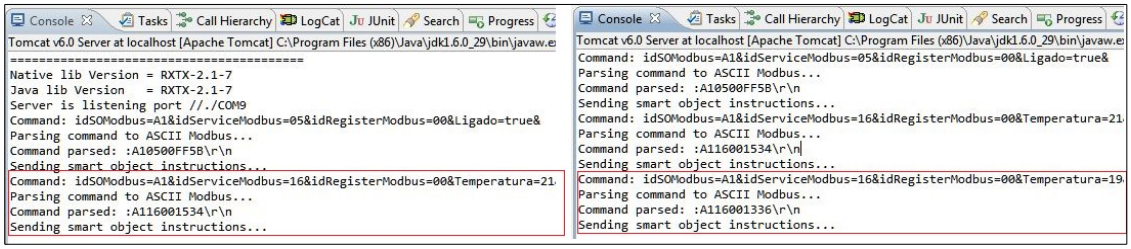

Figura 9: Logs demonstram comunicação com OI usando Wifi e ModBus

Um segundo Epos-Mote representa o acesso ao condicionador de ar e se comunica (diretamente ou por multi-*hops*) com o *gateway* via *ModBus*. As cores do LED desse Epos-Mote indicam que as mensagens foram recebidas com sucesso pela rede de “coisas”, e que os serviços solicitados pela GUI do dispositivo móvel foram efetivamente executados no OI, completando com sucesso o teste.

Figura 10: Epos-mote do condicionador de ar indicando que a temperatura alterada

6 Conclusão

Este artigo apresentou o desenvolvimento de um sistema para dispositivos móveis capaz de gerar interfaces gráficas de interação com objetos inteligentes de forma automática. O sistema desenvolvido auxilia na monitoração e atuação de objetos inteligentes numa IoT, a partir de um dispositivo móvel qualquer com Android. Ele é capaz de gerar telas

gráficas automaticamente para qualquer novo objeto inteligente encontrado pelo serviço de descoberta, e dessa forma não é necessário desenvolver uma interface gráfica de controle para cada objeto inteligente, o que deve ser muito útil aos usuários de redes de IoT, incluindo diferentes sistemas de automação. 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 artigo ter sido dado à geração automática de interfaces gráficas para controle de objetos inteligentes (OI), foram desenvolvidos também os sistemas de comunicação e acesso a objetos inteligentes reais, implementados com o equipamentos embarcados Epos-Motes para redes de sensores sem fio, 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.