

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

**Geração Automática de GUIs para Objetos Inteligentes em
Dispositivos Móveis**

Ercílio Gonçalves Nascimento

Prof. Rafael Luiz Cancian, Dr. Eng.
Orientador

Florianópolis (SC), Junho de 2014

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Geração Automática de GUIs para Objetos Inteligentes em
Dispositivos Móveis

Ercílio Gonçalves Nascimento

Relatório apresentado à Banca
Examinadora do Trabalho de
Conclusão do Curso de Sistemas
de Informação para análise e
aprovação.

Orientador: Prof. Rafael Luiz
Cancian, Dr. Eng.

Florianópolis (SC), Junho de 2014

Ercílio Gonçalves Nascimento

Geração Automática de GUIs para Objetos Inteligentes em Dispositivos Móveis

Este Trabalho de Conclusão de Curso foi julgado, adequado e aprovado em sua forma final para obtenção do grau de Bacharel em Sistemas de Informação da Universidade Federal de Santa Catarina.

Florianópolis, ____ de ____ de ____.

Orientador:

Prof. Dr. Eng. Rafael Luiz Cancian

Banca avaliadora:

Prof. Dr. Antônio Augusto Fröhlich

Prof. M.Sc. Arliones Hoeller Jr.

Aos exemplos da minha vida,
Maria, Gil, Darlene e Bianca
Dedico

AGRADECIMENTOS

À minha mãe, Maria, a qual me espelho para fazer o melhor de mim e que me ensinou a conquistar meus objetivos mesmo não morando juntos. Obrigado por me ajudar em tudo. Muito obrigado mãe!

Aos meus irmãos do coração Gil e Darlene, que me ajudaram nos momentos difíceis da minha vida sempre dando apoio e conselhos me guiando em mais uma jornada de sucesso. Obrigado pelo apoio e carinho!

À minha namorada, Bianca, pela compreensão, ensinamentos, companhia e tudo que foi agregado para a conclusão deste trabalho. Tenho muito orgulho de você e de estar com você. Muito obrigado, amo você!

À banca, professores Antônio Augusto Fröhlich e Arliones Hoeller Jr., pela paciência, ensinamentos, pelas contribuições ao desenvolvimento desse trabalho e também por disponibilizar materiais que facilitaram a conclusão e a defesa do projeto. Muito Obrigado.

Ao meu orientador, Rafael Luiz Cancian, pelo privilégio de ser seu orientado e pela sua dedicação e colaboração neste trabalho. Meus sinceros agradecimentos!

RESUMO

Este trabalho de conclusão de curso teve como finalidade o desenvolvimento de um aplicativo para dispositivos móveis para a criação automática de GUIs para a interação com os objetos inteligentes de uma rede de IoT (Internet of Things). Com base no estudo realizado, optou-se por utilizar dispositivos com Android para fornecer a interface gráfica ao usuário, o EPOS-MOTE como equipamento embarcado de controle dos objetos inteligentes, o protocolo de comunicação HTTP para interação com os objetos inteligentes, e algoritmos genéticos para a criação automática da interface gráfica. Os resultados foram satisfatórios, havendo a descoberta dos serviços providos por Objetos Inteligentes, a geração automática de uma Interface Gráfica para cada novo Objeto descoberto, e a interação correta via rede sem fio com os Objetos Inteligentes, que agora podem ser monitorados e/ou controlados a partir de qualquer dispositivo móvel com Android.

ABSTRACT

This completion of course work aimed to develop a mobile application for the automatic creation of GUIs for interacting with smart objects in a network of IoT (Internet of Things). Based on this study, we chose to use devices with Android to provide the graphical user interface, the EPOS-MOTE equipment as embedded control smart objects, the HTTP communication protocol for interacting with smart objects, and genetic algorithms for the automatic creation of graphical user interface. The results were satisfactory, with the discovery of the services provided by Smart Objects and the automatic generation of a graphical user interface for each newly discovered object, and the correct interaction with Smart Objects using a wireless network, which can now be monitored and controlled from any mobile device with Android.

Lita de Figuras

Figura 1: Computação ubíqua e pervasiva na caneca inteligente - Beigl, Gelerssen e Schmidt 2001.....	6
Figura 2: Painel de controle residencial.....	9
Figura 3: Esquema de um sistema inteligente residencial - Liu et al, 2011.....	14
Figura 4: Esquema de estações de carregamento de veículos elétricos - Liu et al, 2011.....	15
Figura 5: Topologia do ambiente inteligente.....	20
Figura 6: Casos de uso do sistema.....	22
Figura 7: Rastreamento dos casos de uso.....	23
Figura 8: Diagrama de Pacotes.....	24
Figura 9: Classes do Pacote de Descoberta de Objetos Inteligentes.....	25
Figura 10: Classe do Pacote de Gerenciamento.....	26
Figura 11: Classes do Pacote de Interação com Objetos Inteligentes.....	27
Figura 12: Classes do Pacote de Geração Automática de Interface.....	28
Figura 13: Classes do Pacote de Interface Gráfica.....	30
Figura 14: Classes do Pacote POJO.....	31
Figura 15: GUIGenerating Use Case.....	33
Figura 16: GUIGenerating código.....	36
Figura 17: Modelo Relacional de Banco de Dados.....	37
Figura 18: SORespository - Descoberta de Objetos Inteligentes.....	38
Figura 19: SOServer - Tradução HTTP para ASCII ModBus.....	39
Figura 20: Fenótipo Pai.....	40
Figura 21: Fenótipo Mãe.....	41
Figura 22: Fenótipo Filho após Crossover.....	41
Figura 23: Fenótipo Final após Mutação.....	42
Figura 24: Tela inicial com dados incorretos.....	42
Figura 25: Log de usuário inválido.....	42
Figura 26: Mensagem de usuário inválido.....	43
Figura 27: Tela inicial com dados corretos.....	43
Figura 28: Log de descoberta de objetos inteligentes.....	43

Figura 29: Resultado do banco de dados.....	44
Figura 30: Lista de Ols.....	44
Figura 31: Telas geradas para o Ar Condicionado.....	45
Figura 32: Telas geradas para a Lâmpada.....	45
Figura 33: Serviço "Ligar" do Ar Condicionado.....	46
Figura 34: Log do comando "Ligar" do Ar Condicionado.....	46
Figura 35: Ar Condicionado ligado.....	46
Figura 36: Serviço "Aumentar" e "Diminuir" temperatura do Ar Condicionado.....	47
Figura 37: Logs de Aumentar e Diminuir a temperatura do Ar Condicionado.....	47
Figura 38: Ar Condicionado com a temperatura alterada.....	47
Figura 39: Serviços "Desligar" e "Ligar" Lâmpada.....	48
Figura 40: Logs de Desligar e Ligar Lâmpada.....	48
Figura 41: Lâmpada desligada e ligada.....	48

LISTA DE ABREVIATURAS

- AAHB – American Association of House Builders
AMR – Automatic Meter Reading
AWT – Abstract Window Toolkit
CoAP – Constrained Application Protocol
CSIT – Costumer Smart Interactive Terminal
DOM – Document Object Model
GUI – Graphic User Interface
HTTP – Hypertext Transfer Protocol
IETF – Internet Engineering Task Force
IoT – Internet of Things
LAN – Local Area Network
LCD – Liquid-crystal Display
LED – Light-emitting diode
M2M – Machine-to-Machine
OI – Objetos Inteligentes
PLC – Power Line Carrier
SOAP – Simple Object Access Protocol
URI – Uniform Resource Identifier
VoIP – Voice over Internet Protocol
WLAN – Wireless Local Area Network
WSDL – Web Service Definition Language
XML – eXtensible Markup Language
MIT – Massachusetts Institute of Technology
RFID – Radio-frequency Identification
RPC – Remote Procedure Call
WS4D – Web Services For Devices
ITEA – Information Technology for European Advancement

Sumário

1.INTRODUÇÃO.....	1
1.2.PROBLEMATIZAÇÃO.....	2
1.2.1.Formulação do Problema.....	2
1.2.1.Solução Proposta.....	2
1.2.OBJETIVOS.....	3
1.2.1.Objetivo Geral.....	3
1.2.2.Objetivos Específicos.....	3
1.3.ESTRUTURA DO TRABALHO.....	4
2. FUNDAMENTAÇÃO TEÓRICA.....	5
2.1FUNDAMENTOS RELACIONADOS E PROBLEMA TRATADO.....	5
2.2 TRABALHOS SIMILARES.....	12
2.3TECNOLOGIAS RELACIONADAS.....	15
2.3.1 Webservices.....	15
2.3.2 ZigBee.....	16
2.3.3 Dispositivos Móveis.....	17
2.3.4 Android e Geração Automática de GUI.....	17
3.PROJETO.....	19
3.1. Requisitos do Software Controlador de OI.....	19
3.2. Topologia de um Ambiente equipado com OI.....	20
3.3. Interação Entre o Usuário e o Sistema.....	21
3.4. Pacotes do Sistema.....	23
3.4.1.Pacote de Descoberta de Objetos Inteligentes.....	24
3.4.2. Pacote de Gerenciamento.....	26
3.4.3. Pacote de Interação com o Objeto Inteligente.....	27
3.4.4. Pacote de Geração Automática de GUI.....	28
3.4.5. Pacote de Interface Gráfica.....	29
3.4.6. Pacote de Classes POJO.....	31
4.DESENVOLVIMENTO.....	32
4.1.Algoritmo Genético.....	32
4.2.Demais Aplicações Desenvolvidas.....	36
4.2.1.Banco de Dados.....	37
4.2.2.SORespository.....	37
4.2.3.SOServer.....	38
5.TESTE E VALIDAÇÃO.....	40
5.1.Algoritmo Genético.....	40
5.2.Autenticação de Usuário.....	42
5.3.Descoberta de Objetos Inteligentes.....	43
5.4.Geração de Telas.....	44

5.5.Interação com o Objeto Inteligente.....	45
5.5.1.Ar Condicionado.....	46
5.5.1.1.Ligar.....	46
5.5.1.2.Aumentar e Diminuir a Temperatura.....	47
5.5.2.Lâmpada.....	47
5.5.2.1.Ligar e Desligar.....	47
6.CONCLUSÕES E TRABALHOS FUTUROS.....	49
REFERÊNCIAS BIBLIOGRÁFICAS.....	51
ANEXOS.....	53

1. INTRODUÇÃO

Atualmente utilizamos a informática para auxiliar as atividades diárias de forma consciente e muitas vezes também quando nem percebemos. O campo que estuda o uso da computação “escondida” é chamado de Computação Ubíqua e Pervasiva.

O conceito de computação ubíqua e pervasiva vem sendo estudado ao longo dos anos com o intuito de facilitar as atividades realizadas diariamente por qualquer pessoa, desde efetuar uma simples compra de um produto até automatizar processos antes feito por humanos.

A computação ubíqua e pervasiva está intimamente ligada com a Internet das Coisas (IoT), no sentido que as “coisas” conectadas correspondem aos sistemas computacionais imperceptíveis com os quais se interage. Essas “coisas” devem se tornar participantes ativos no mundo dos negócios, informação e processos sociais, onde elas são capazes de interagir e se comunicar entre si e com o ambiente através da troca de dados e informações “captadas” sobre o ambiente, reagindo de forma autônoma a estes eventos captados (Vermesan et al, 2009).

A IoT pode ser considerada uma infraestrutura de rede dinâmica que possui capacidade de autoconfiguração e utiliza padrões e protocolos de comunicação para estabelecer a interoperabilidade entre as “coisas”. Nessa rede todos os dispositivos possuem identificadores, atributos físicos e interfaces inteligentes, normalmente projetadas especificamente para esse domínio.

Numa rede de IoT, os elementos que fazem parte desta rede são chamados de Objetos Inteligentes (OI), e é através desses objetos que a rede se comunica e a interação é efetivada. Os objetos inteligentes precisam ocupar pouco espaço para que caibam nos componentes residenciais e também têm de ser baratos e com pouco consumo de energia elétrica. Esses objetos inteligentes, então, correspondem a um sistema computacional embarcado, projetado para uma função específica.

1.2.PROBLEMATIZAÇÃO

1.2.1.Formulação do Problema

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 (fala, gestos, expressões faciais, etc), 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. 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 consumir esses serviços de IoT.

Para a interação do usuário com os objetos inteligentes, os usuários desejam uma interface gráfica amigável e de fácil entendimento. Porém, é impraticável desenvolver uma interface gráfica para cada OI e, além disso, usuários não acessarão os OI a partir deles, mas a partir de seus próprios computadores, e principalmente a partir de diferentes dispositivos móveis.

1.2.1.Solução Proposta

As áreas de computação ubíqua e pervasiva e Internet das Coisas poderiam ser beneficiadas por um sistema que descobrisse serviços de Objetos Inteligentes de uma rede e gerasse de forma dinâmica e automática uma interface gráfica amigável para configurar e interagir com um OI escolhido a partir do dispositivo em que esse sistema estiver executando, seja um laptop, um smartphone ou outro dispositivo móvel, ajustando os componentes da GUI para as características da tela e do dispositivo em questão. Esta solução beneficiaria o usuário, que poderia utilizar um dispositivo qualquer para controlar todos os elementos de uma rede de IoT, e poderia escolher a GUI mais adequada a cada elemento.

Neste trabalho foi desenvolvido um aplicativo para dispositivo móvel 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 e as características do dispositivo móvel. 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.

1.2.OBJETIVOS

1.2.1.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.

1.2.2.Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Pesquisar e analisar as diferentes soluções para geração automática de interface gráfica;
- 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.

1.3. ESTRUTURA DO TRABALHO

O trabalho está estruturado em 6 capítulos:

- Capítulo 1: introduz o tema tratado no trabalho;
- Capítulo 2: revisão bibliográfica sobre o tema demonstrando os trabalhos similares, estado da arte e o que ainda está faltando;
- Capítulo 3: definição do projeto da solução computacional;
- Capítulo 4: desenvolvimento da solução computacional definida na etapa de projeto;
- Capítulo 5: teste e validação do sistema desenvolvido; e
- Capítulo 6: conclusão e considerações finais do projeto.

2. FUNDAMENTAÇÃO TEÓRICA

Antigamente os computadores eram enormes e ocupavam diversas salas. Com o passar do tempo, a tecnologia foi evoluindo e novos designs e técnicas de arquitetura computacional foram sendo explorados. A partir de então surgiram os computadores pessoais, desktops e notebooks, e mais recentemente os dispositivos móveis, hoje representados principalmente por smartphones e tablets, disponíveis virtualmente para qualquer pessoa.

Pode-se dizer que hoje existe uma dependência do computador por parte da população mundial em geral, e diversas tarefas são executadas diariamente utilizando a tecnologia. Hoje é possível pagar contas, comprar passagens de avião, alugar um filme, tudo com apenas um clique e sem sair de casa. A informática é utilizada também na área da saúde e educação, engenharia e segurança; enfim, está completamente imersa no cotidiano das pessoas mesmo que às vezes não sendo percebida. Os fundamentos nesse sentido e no contexto do trabalho desenvolvido são apresentados na próxima seção.

2.1 FUNDAMENTOS RELACIONADOS E PROBLEMA TRATADO

O conceito de computação ubíqua e pervasiva é antigo e foi mencionado pela primeira vez por Weiser (1991), e dizia que “elementos especializados de hardware e software, conectados por fios, ondas de rádio e infravermelho, serão tão ubíquos que ninguém notará a presença deles”. A computação ubíqua e pervasiva mencionada por Weiser tem a ideia de integrar totalmente a relação homem-máquina para que seja feita essa interação de forma invisível, no sentido de que o uso seja imperceptível. A comunicação entre o ser humano e esses tipos de computadores seria natural, através da fala, gestos, expressões faciais, entre outros. Ao sacar dinheiro por exemplo, poderia ser inserida a senha apenas com o movimento dos olhos, aumento assim a segurança pelo fato de não ter que escolher fisicamente os botões. Isto se expande para outras áreas além da segurança.

Um estudo de caso feito por Beigl, Gellersen e Schmidt (2001) apresenta uma caneca inteligente. A base da caneca é feita de borracha e dentro dela é integrada toda a parte eletrônica com os sensores e outros hardware necessários para o funci-

onamento. Os sensores detectam dados como movimento da caneca, a temperatura e a quantidade de café, avisando a cafeteira de que o café acabou para que ela possa fazer mais. Outra funcionalidade da caneca inteligente é que após a caneca ter ficado parada e o café esfriado, ao se fazer o movimento para tomar café, a caneca avisa que o líquido deverá ser aquecido. A caneca inteligente denominada pelos autores de Mediacups é ilustrada na figura 1.



Figura 1: Computação ubíqua e pervasiva na caneca inteligente
- Beigl, Gelerssen e Schmidt 2001

Este tipo de computação pode ser expandido para praticamente qualquer outro elemento comum nas residências, para proporcionar mais conforto e comodidade aos moradores: lâmpadas que ligam automaticamente, geladeiras que informam a existência de comidas com prazo de validade vencidos, a máquina de lavar louças que é ligada automaticamente quando estiver cheia.

Este tipo de computação em que os objetos são inteligentes e a interação homem-máquina é transparente está intimamente ligada com a Internet das Coisas (IoT). A Internet das Coisas é uma revolução tecnológica que representa o futuro da computação e comunicação e que está instalada em tudo: carros, interruptores de luz, mesas, canetas, tudo conectado por algum meio, seja ele físico ou não, tendo a função de observar alterações no ambiente desejado e disparar algum tipo de ação.

A IoT foi mencionada pela primeira vez pelo britânico Kevin Ashton em 1999. Ashton foi cofundador do Auto-ID Center do Instituto de Tecnologia de Massachusetts (MIT), instituto este responsável por criar o sistema de padronização global para a identificação por radiofrequência (RFID) e outros sensores. Na literatura, a respeito de IoT Vermesan et al dizem que:

Na IoT, “as coisas” devem se tornar participantes ativos no mundo dos negócios, informação e processos sociais, onde elas são capazes de interagir e se comunicar entre si e com o ambiente através da troca de dados e informações “captadas” sobre o ambiente, reagindo e influenciando de forma autônoma aos eventos do mundo “real/físico” executando processos que disparam ações e criam serviços com ou sem interação humana direta (Vermesan et al, 2009).

A IoT pode ser considerada uma infraestrutura de rede dinâmica que possui capacidade de autoconfiguração e utiliza padrões e protocolos de comunicação para estabelecer a interoperabilidade entre as “coisas”. Nessa rede todos os dispositivos possuem identificadores, atributos físicos e interfaces inteligentes.

Uma rede de IoT é muito útil no controle de infecções, no registros de animais e até mesmo na prevenção de desastres naturais, tema este muito estudado devido aos abalos ocasionados pelas grandes indústrias, desmatamentos e outras ações provocadas pelo homem contra a natureza. IoT também pode ser encontrada em Smart Cities (Cidades Inteligentes, equipadas com Objetos Inteligentes que auxiliam no controle do transporte público, monitoração do trânsito, entre outras tarefas).

Smart Grids (Redes Inteligentes) possuem um conjunto de sensores instalados na rede de energia elétrica que têm entre suas funções o monitoramento dos componentes elétricos. Os Ols que compõem a rede IoT se comunicam por radiofrequência e podem disponibilizar seus serviços na internet através de uma das tecnologias disponíveis no mercado como WiFi, 3g, GPRS, Zigbee, entre outros, e quando tais serviços não são disponibilizados é necessário realizar uma intervenção via linguagem baixo nível. Pelo fato de serem pequenos, possuírem pouca capacidade de memória e processamento, baixa disponibilidade de energia e pela grande quantidade de Ols existente numa rede IoT, fica impraticável desenvolver uma GUI para cada elemento inteligente, por isso é interessante ter em mãos um dispositivo de controle que possa modelar em seu visor uma interface gráfica dinâmica de acordo com cada funcionalidade dos objetos, sendo deste modo capaz de interagir com tais objetos completamente.

O problema de interação entre usuário e objetos inteligentes também pode ser encontrado numa rede IoT em Smart Cities (Cidades Inteligentes) e deve ser tra-

tado para promover a facilidade e a praticidade ao usarmos a computação a nosso favor.

Cidades inteligentes são cidades que possuem um sistema integrado de serviços computacionais e de comunicação fundamental para alcançar sustentabilidade e facilitar o desenvolvimento sustentável das indústrias e residências. Os líderes possuem as ferramentas necessárias para analisar os dados obtidos e tomar melhores decisões, antecipar problemas e saná-los proativamente e coordenar os recursos da cidade. O primeiro ponto para alcançar sustentabilidade é reduzir o consumo de energia e a emissão de gases poluentes que incrementam o efeito estufa. Cidades inteligentes oferecem um suporte eficiente para comunicação e acesso aos serviços e informação (Castro, Jara e Skarmeta, 2013).

Uma cidade inteligente pode ser muito útil no caso onde a quantidade populacional é elevada, nas questões de segurança, água, energia, lixo, etc. Baseado em padrões de consumo pode-se saber em que parte da cidade a distribuição de água e energia é mais necessária. Através da IoT pode-se prevenir gargalos do trânsito, predizer em quanto tempo o rio irá transbordar em dias de chuva e o controle de epidemias pode ser mais eficiente. A ideia de cidades inteligentes também pode ser aplicada às casas inteligentes.

O conceito de casas inteligentes foi oficialmente introduzido pela Associação Americana de Construtoras Residenciais em 1984 (LIU et al., 2011) e diz que a automação residencial não trata somente a comunicação entre elementos áudio/visuais mas também ar-condicionado, luzes, janelas, enfim, qualquer objeto que possa ser controlado de forma dinâmica e fácil. Tais objetos estão interconectados com outros por algum tipo de protocolo estabelecendo comunicação entre si e podem ser acessados de qualquer lugar da casa. Segundo Aldrich (2003) uma casa inteligente é definida como “um lugar equipado com tecnologia computacional e de informação que responde às requisições solicitadas pelos ali presentes”.

Um dos propósitos das casas inteligentes é promover ao morador o conforto, conveniência e segurança através de um gerenciamento das várias tecnologias encontradas dentro de casa e fora dela, reduzindo o consumo de energia e diminuindo a emissão de carbono na natureza.

Existem diversas utilidades em uma smart home. Um exemplo dessa utilidade é quando o sistema de incêndio detecta sinal de fumaça ou fogo e automaticamente dispara mensagens para os celulares dos proprietários da residência e bombeiros. Detectores de movimento podem fazer com que as luzes externas sejam ligadas apenas quando forem necessárias, de modo que haja uma diminuição do desperdício de energia elétrica e consequentemente uma economia financeira. Pode-se também controlar televisores, aparelhos de ar-condicionado, janelas e home theater apenas com um único controle, como smartphones e tablets.

A figura 2 abaixo demonstra um painel de controle residencial capaz de controlar as luzes, segurança, termostato, fechaduras e recursos de entretenimento.



Figura 2: Painel de controle residencial

Nas cidades e casas inteligentes os objetos que fazem parte da infraestrutura e que estão interconectados são chamados de Objetos Inteligentes. É por meio deles que todo o sistema inteligente se comunicará.

Os objetos inteligentes precisam ocupar pouco espaço para que caibam nos componentes residenciais e também têm de ser baratos e com pouco consumo de energia elétrica. Esses dispositivos contêm um sistema computacional embarcado projetado apenas para a função de cada um. Por serem pequenos, eles possuem pouca capacidade de processamento, com memórias curtas e geralmente sem interface gráfica para a interação humana.

A construção de um ambiente inteligente a partir destes objetos pode fazer com que a interação homem-máquina seja transparente, pois os objetos inteligentes podem ser criados para “pensar” através de microcontroladores e sistemas embarcados, estabelecendo uma comunicação entre humanos, espaços e outros objetos por meio de uma linguagem natural através da voz, gestos ou expressões as quais as pessoas já estão acostumadas quando interagem umas com as outras (FAREL-LA, 2010). Tais objetos se comunicam através de redes sem fio, infravermelho, bluetooth e podem disponibilizar seus serviços na internet através de webservices (serviços disponibilizados na rede por linguagem XML que tem a principal função estabelecer a interoperabilidade entre aplicações de linguagens diferentes).

A facilidade de utilizarmos a computação a nosso favor sem que percebamos é ótima, e também seria se quiséssemos utilizar de forma controlada, em outras palavras, controlar os elementos inteligentes e acessar seus serviços.

O controle dos OI é importante para dar autonomia aos moradores ao efetuar algumas tarefas simples e diárias como ligar a luz, levantar e abaixar a persiana, entre outras. Imagine dezenas de objetos inteligentes instalados nos cômodos residenciais os quais queremos controlar. Poderia ser utilizado um controle remoto para cada aparelho, codificação baixo nível, ou talvez uma central fixa em algum ponto da casa para controlá-los. Mori, Nonaka e Hassi (2010) desenvolveram um sistema centralizado capaz de controlar os aparelhos audiovisuais da casa. A grande motivação neste quesito é a tentativa de diminuir ao máximo o número de controles remotos, desenvolver apenas um que seja possível controlar todos os dispositivos residenciais. Outro fator não menos importante é a segurança: necessita-se fazer o uso de uma proteção caso alguém mal intencionado e não autorizado tente se conectar em algum elemento da casa e manipulá-lo, como por exemplo, ao abrir a porta.

Uma alternativa para o problema de inúmeros controles remotos é utilizar dispositivos móveis levando-se em consideração a grande disponibilidade de mercado, acessibilidade, locomoção dentro de casa, conforto e comodidade. Desta forma seria possível utilizar apenas um dispositivo controlador em um sistema inteligente.

Os dispositivos móveis estão em alta no mercado. Uma pesquisa revelada pela consultoria Nielsen (<http://br.nielsen.com/site/index.shtml>) realizada no primeiro semestre de 2013 diz que em cada 10 brasileiros que possuem celular, 3 são

smartphones. O acesso de compra facilitado pelo governo promove um aumento significativo na quantidade destes dispositivos a cada dia. Por esse motivo é interessante criar algum mecanismo de controle dos objetos inteligentes através destes aparelhos. Tal mecanismo deve possuir uma interface gráfica amigável e de fácil entendimento do usuário já que na maioria dos casos eles não possuem conhecimento técnico dos OI.

Ao desenvolver um controle universal para a manipulação dos objetos inteligentes deve-se levar em consideração a interface gráfica, construí-la de modo que consiga atender todas as opções dos equipamentos, e se for utilizar dispositivos móveis, desenvolver uma aplicação que se modele para todos os diferentes tipos de aparelhos, com tamanhos e resoluções de telas variados. Caso esse controle universal seja sem fio, o consumo de bateria entre a comunicação do controle com os objetos deverá ser minimizado para que não seja necessário recarregar o controle frequentemente.

É imprescindível que haja um estudo acerca das resoluções, quantidade de pixels e aspectos das telas dos equipamentos para alcançar uma uniformidade da interface gráfica independente dos modelos de aparelhos, desenvolver elementos gráficos baseados em densidade de tela é uma ideia interessante ao trabalhar neste quesito.

Apesar das novas tecnologias dentro da IoT estarem em constante evolução auxiliando nos problemas enfrentados diariamente por todos nós, há de se considerar alguns problemas ainda encontrados no uso desta tecnologia. Os dispositivos controladores dos objetos inteligentes precisam ter uma interface gráfica amigável e intuitiva, pois o usuário não precisa ter conhecimento profundo do que ele quer controlar. Os elementos inteligentes devem estabelecer uma comunicação com o mundo externo através de serviços webservices, disponibilizando suas funcionalidades para a interação com outros dispositivos. Estes objetos precisam consumir pouca energia elétrica quando ligados a uma bateria e serem bem projetados por possuírem pouca capacidade de processamento e memória. Outra dificuldade encontrada é o desenvolvimento de um mecanismo de segurança para acesso destes objetos na eventualidade de ataques virtuais de pessoas maliciosas.

Pode-se encontrar na literatura pesquisas relacionadas a esses assuntos mencionados na tentativa de resolver estes problemas, mas não são totalmente sa- nados.

2.2 TRABALHOS SIMILARES

Mori, Nonaka e Hassi (2010) abordam a criação de uma interface gráfica au- tomática para 5 dispositivos de sistema audiovisual previamente cadastrados. Nesse caso, foi utilizada uma tela LCD que apresenta uma lista de equipamentos disponí- veis para uso. Após selecionado, o sistema gera automaticamente, através de algo- ritmo genético, uma nova tela com todas as opções de manipulação possíveis sepa- radas 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 op- ções disponíveis para um determinado equipamento. O cálculo para a geração da in- terface 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 microcontrola- dor operando na frequência de 100MHz e um painel touch. Como requisito não-fun- cional principal, foi estabelecido um tempo máximo de 30 segundos em cada gera- ção de tela.

Os autores conseguiram desenvolver uma aplicação que centraliza o controle dos equipamentos de áudio e vídeo, contudo, há a necessidade de cadastrar previa- mente os aparelhos, não possibilitando assim o controle à outros tipos de equipa- mentos.

Segundo Izquierdo et al (2009), em geral, 50% do tempo de desenvolvimento de uma aplicação está na criação da interface gráfica. A partir disto, os mesmos au- tores tiveram a ideia de criar um framework para geração automática de interface gráfica em webservices. É um aplicativo escrito na linguagem Java onde é possível importar o arquivo WSDL e para cada serviço disponível, gerar uma tela automática (arquivo este baseado na linguagem XML que estabelece as especificações de ser- viço como métodos disponíveis, parâmetros necessários, como acessá-los entre ou-

etros, funcionando como um contrato de serviço). 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, o qual segundo os autores, oferece mais sofisticação do que os componentes AWT. Mesmo tratando do WSDL ser um arquivo XML, optou-se por utilizar não somente as classes nativas do java DOM, mas também uma biblioteca externa de manipulação deste tipo de arquivo, a biblioteca WSDL4J. 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.

Em relação à outras funcionalidades das aplicações de IoT, Liu et al (2011) expõem as mais relevantes a seguir:

1 - Monitoramento Online de Linha de Transmissão de Energia Elétrica: Segundo os autores, este tipo de aplicação é um dos mais importantes na área de smart grids. Nos últimos anos, desastres naturais vem trazendo desafios nas linhas de transmissão de alta tensão, dentre elas a segurança, estabilidade e confiança. Como o monitoramento nessas instalações são feitos principalmente por operações manuais, viu-se a necessidade de criar um monitoramento em tempo-real. Sensores foram instalados nos cabos de uma rede experimental, os quais medem vibração do vento, congelamento dos cabos, temperatura dos condutores entre outros. Este monitoramento é composto por sensores instalados nas torres de transmissão, equipados com conexão wireless para transferência dos dados obtidos, e outra parte onde os dados são visualizados.

2 - Sistema de Casa Inteligente: Uma casa totalmente integrada com um sistema inteligente capaz de assegurar conforto, diminuição no consumo de energia e segurança. A figura 3 abaixo ilustra a estrutura do sistema inteligente. Existe uma rede de energia elétrica com componentes inteligentes instalados para monitoramento de aparelhos elétricos tais como aquecedores, máquinas de lavar, ar

condicionado e células fotoelétricas e um Access Point PLC disponibilizando a transmissão de uma rede sem fio do tipo ZigBee para detectores de fumaça e botões de emergência. Um cabeamento de LAN ligando computadores pessoais, televisores, telefones VoIP e uma central de controle (CSIT) dos medidores AMR de água e gás.

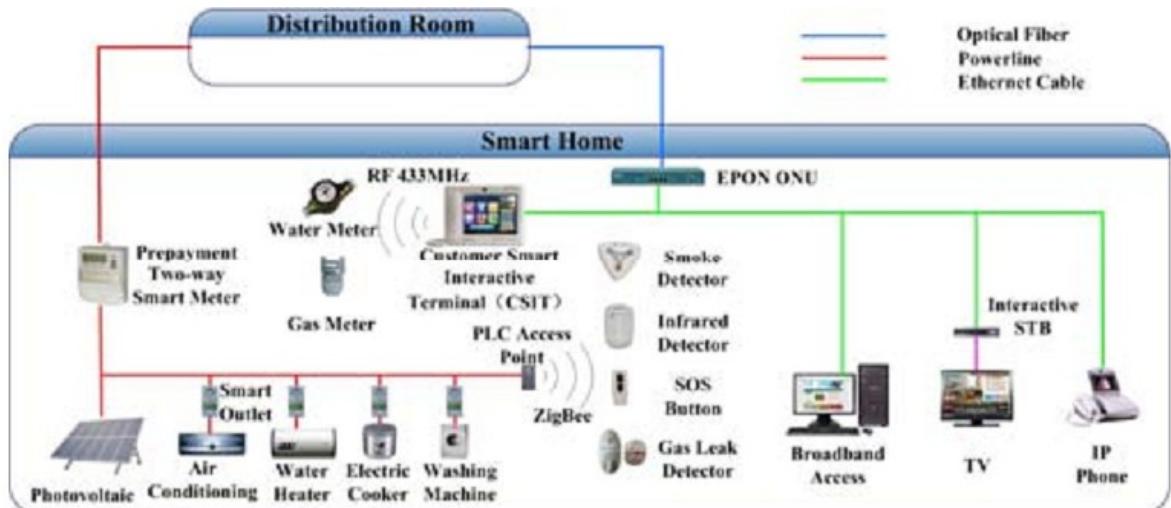


Figura 3: Esquema de um sistema inteligente residencial - Liu et al, 2011

3 - Sistema de Gerenciamento de Informação para Veículos Elétricos: Veículos elétricos oferecem uma grande oportunidade em smart grid e contribuem para a conscientização em relação à poluição do meio ambiente.

Estações de recarregamento dos veículos elétricos estão ilustrados na figura 4. Os equipamentos responsáveis pelo recarregamento dos veículos podem operar em corrente alternada, geralmente instalado nas residências e fornecem um recarregamento lento colaborando com a natureza, ou podem operar em corrente contínua, este último nos lugares onde é necessária a eficiência e agilidade. O sistema de monitoramento é responsável pela segurança e obtenção de dados em tempo-real do ambiente de recarga. Segundo Liu et al (2011), veículos elétricos podem ser considerados sistemas móveis de armazenamento de energia em smart grids.



Figura 4: Esquema de estações de carregamento de veículos elétricos - Liu et al, 2011

Trivedi, Lai e Zhang (2001) apresentam um desenvolvimento automatizado de janelas utilizando algoritmos genéticos. Esse algoritmo calcula a disposição das janelas na tela de forma que ocupem o menor espaço possível dependendo do tamanho de cada uma, e considera ainda que tais janelas não podem se sobrepor. O algoritmo genético se baseia na teoria da evolução de Darwin, na qual diz que os indivíduos precisam se adaptar ao ambiente para sobreviverem. Essa adaptação é alcançada através de mutação genética e operações de *crossover*, fazendo com que as gerações evoluam com o passar do tempo.

Em outra pesquisa na área de Android e webservices, Hsieh et al (2012) apresentam um framework na plataforma Android no qual implementam um widget que consome um webservice para apresentação de dados de turismo. O sistema foi desenvolvido para dispositivos móveis como smartphones e tablets.

2.3 TECNOLOGIAS RELACIONADAS

2.3.1 Webservices

Os objetos inteligentes podem estar ligados a uma bateria para alimentação elétrica, localizados em diversas partes numa casa inteligente, podem utilizar redes sem fio para troca de informações e implementar protocolos que facilitem a comunicação entre eles.

Ol possuem uma diversidade muito grande de serviços disponíveis. A publicação desses serviços na rede inteligente pode ser feita pela Internet para que ou-

tos dispositivos possam se comunicar através de requisições HTTP, e também por webservices. Os webservices têm a funcionalidade principal de interoperabilidade entre aplicações, ou seja, uma aplicação escrita numa linguagem pode trocar informações com outra aplicação escrita em outra linguagem através da Internet, isto é possível pois a comunicação entre os dois sistemas se dá através de uma linguagem universal padronizada, a XML.

Os webservices possuem um identificador chamado URI (Uniform Resource Identifier) e utilizam protocolos de acesso específicos, entre eles o SOAP (Simple Object Access Protocol). Este protocolo é baseado em XML para troca de mensagens e chamadas de procedimentos remotos (RPC) e pode trabalhar através de requisições HTTP. Sua estrutura básica é composta por um cabeçalho e um corpo (Curbura et al 2002). Tanto a chamada quanto a resposta das requisições é feita pela linguagem XML, e todos os métodos com seus argumentos estão definidos no arquivo WSDL, que serve como um contrato de serviço.

Quando o assunto é IoT, pode-se utilizar o protocolo CoAP (Constrained Application Protocol) que é um protocolo destinado à transferência de informações pela web em redes inteligentes. Segundo a Internet Engineering Task Force (IETF) o CoAP é um protocolo da camada de aplicação e possui um baixo consumo de energia, suporte a multicast e foi desenvolvido para sensores de baixo consumo e componentes que podem ser controlados remotamente através do padrão da internet. A maior parte da padronização deste protocolo foi feita pela IETF.

A ideia inicial desse Trabalho de Conclusão de Curso era utilizar protocolos *webservices* para descobrir os serviços disponibilizados pelos objetos inteligentes e fazer a comunicação entre dispositivo móvel e OI, porém devido a descontinuidade de um projeto paralelo, essa ideia foi descartada.

2.3.2 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.

2.3.3 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.

Marcondes et al (2006) apresentam um sistema operacional portável para sistemas profundamente embarcados, o EPOS. Este sistema operacional é orientado à aplicação e baseado em componentes, sendo desenvolvido facilitando a portabilidade do sistema. Outros sistemas operacionais também podem ser utilizados no desenvolvimento de aplicativos controladores de objetos inteligentes, entre eles está o Android, uma plataforma baseada em Linux desenvolvida pela Google.

Devido à grande facilidade de desenvolvimento de aplicativos pra esta plataforma, os maiores esforços para controle de objetos inteligentes se concentram neste tema.

2.3.4 Android e Geração Automática de GUI

Segundo uma pesquisa realizada pela equipe de consultoria Kantar Worldpanel ComTech (<http://www.kantarworldpanel.com/Global/News/Soaring-iPhone-5-sales-in-US-knock-Android-into-second-place>), empresa experiente no ramo de conhecimento de consumidores, 56,7% dos smartphones vendidos no Brasil no final de 2012 possuem o Android como sistema operacional, contra apenas 0,4% do seu maior rival, o iPhone. O grande fator disto é a variedade de preços, o Android está presente tanto em aparelhos básicos que possuem acesso à internet quanto em aparelhos mais modernos que competem diretamente com os iPhones. Isso torna os

dispositivos móveis com este sistema operacional um grande atrativo na integração dos usuários com os sistemas inteligentes de IoT.

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. Existe também uma série de ferramentas úteis que combinadas com as Activities facilitam na geração automática de telas nos aplicativos Android, é o caso da classe Fragment. Esta classe tem como principal função representar o comportamento ou uma porção da GUI dentro de uma Activity. Na prática é possível combinar várias Fragments dentro de uma única Activity construindo um painel de GUI's. Segundo o site do próprio fabricante (<http://developer.android.com/guide/components/fragments.html>) “você pode imaginar uma Fragment como uma seção modular de uma Activity, a qual possui seu próprio ciclo de vida, recebe seus próprios eventos de entrada, e pode ser adicionada ou removida em tempo de execução”.

O Android possui em sua etapa de criação de projeto uma estrutura de pastas que possibilita ao desenvolvedor criar um aplicativo capaz de se modelar a todos os tipos e tamanhos de aparelhos Android existentes, seja ele smartphone ou tablet.

Devido às vantagens citadas acerca deste sistema operacional, este trabalho foi desenvolvido na plataforma Android.

3. PROJETO

A materialização do software controlador de objetos inteligentes foi guiada pelo conjunto de artefatos descritos nesta seção. O projeto buscou, através da UML, capturar as características estruturais, comportamentais e de interação do software. Para a modelagem estrutural foi utilizado o diagrama de classe, e para capturar o comportamento do software, os diagramas de caso de uso e sequência.

As seções que seguem abordam as funcionalidades necessárias a este software através de uma listagem de requisitos (Seção 3.1), a topologia de um ambiente equipado com objetos inteligentes pra interação (Seção 3.2), a modelagem das interações entre o usuário e o sistema através de casos de uso (Seção 3.3) e a modelagem dos pacotes por diagramas de classe e sequência (Seção 3.4). Os detalhes pertinentes ao desenvolvimento do projeto são descritos no Capítulo 4 e as técnicas para avaliação da solução no Capítulo 5.

3.1. Requisitos do Software Controlador de OI

O desenvolvimento da aplicação destinada ao controle de objetos inteligentes através de dispositivos móveis foi realizado de acordo com requisitos previamente definidos e que serviram de insumo para o modelo dessa aplicação. Deste modo, foram identificados os requisitos funcionais e não funcionais dispostos na lista abaixo.

- RF01 - O sistema deve ser capaz de pesquisar por objetos inteligentes e seus serviços disponíveis através de redes WiFi utilizando padrões de projeto que facilitem o desenvolvimento de outras tecnologias no futuro;
- RF02 – O sistema deve, ao selecionar um objeto inteligente, gerar uma interface gráfica automaticamente para a interação com o usuário;
- RF03 – O sistema deve enviar comandos ao objeto inteligente previamente selecionado para que seja efetuada a operação desejada;
- RNF01 – A geração da tela não pode levar mais do que 30 segundos;
- RNF02 – O algoritmo de geração de tela deve ser implementado utilizando Algoritmos Genéticos; e

- RNF03 – O sistema deve guardar os 5 melhores indivíduos gerados pelo AG para acelerar a geração de novas telas.

3.2. Topologia de um Ambiente equipado com OI

Um ambiente inteligente é composto por diversos equipamentos e dispositivos interligados por algum meio para que haja comunicação e troca de informação. Esse ambiente é um espaço com sistemas embarcados e tecnologia da informação e comunicação que cria ambientes interativos capaz de trazer a computação para o mundo físico.

Para ilustrar este ambiente, uma possível topologia é demonstrada pela Figura 5 a seguir:

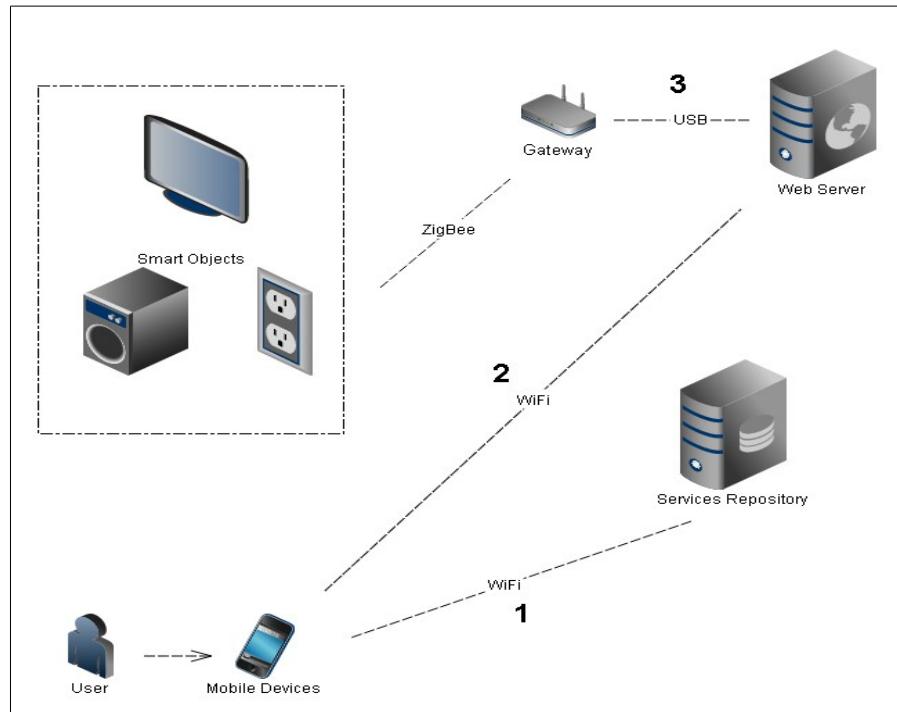


Figura 5: Topologia do ambiente inteligente.

Os objetos inteligentes disponibilizam seus serviços através da tecnologia Zigbee. O *Gateway* é responsável por fazer a comunicação entre Objetos Inteligentes e servidor web através da porta USB. O *Services Repository* possui uma aplicação web e um banco de dados com os registros de todos os objetos inteligentes, contendo também informações necessárias como o endereço IP do servidor onde os OI's estão disponíveis.

Para a manipulação dos Objetos Inteligentes a aplicação desenvolvida neste trabalho obedece a seguinte ordem de tarefas:

1. O usuário digita o seu *login* e senha no aplicativo e seleciona a opção “Pesquisar”. O sistema efetua uma requisição HTTP para o servidor repositório o qual retorna uma lista dos Objetos Inteligentes encontrados e permitidos para o usuário contendo seus serviços e o endereço IP do servidor onde ele está;
2. Para a manipulação do objeto inteligente escolhido ser efetuada, o sistema solicita uma requisição HTTP para o endereço IP do servidor onde o OI está disponível encaminhando os parâmetros necessários no método POST;
3. O servidor transmite essa informação para o *Gateway* através da porta USB, que encaminha via *ZigBee* o comando para o Objeto Inteligente escolhido.

3.3. Interação Entre o Usuário e o Sistema

Os usuários fazem uso do sistema através dos seus próprios dispositivos móveis. O sistema fica responsável pela descoberta e interação com os objetos inteligentes e geração automática da interface gráfica contendo os serviços oferecidos.

Os elementos que interagem com o sistema nesta seção são modelados na UML por atores, que é descrito como “alguém ou algo que interage com o sistema” (Eriksson et al., 2004). Cada interação é capturada por um caso de uso responsável pela descrição da troca de mensagens entre esses elementos. O ator envia uma mensagem ao sistema que responde ao ator com outra. Essa troca continua até que algum critério de parada seja atingido.

Os casos de uso são derivações dos requisitos. A listagem de casos de uso é guiada pelas funcionalidades que devem estar presentes no sistema, onde cada caso de uso busca detalhar a comunicação entre os usuários e o software. Assim,

tomando a listagem de requisitos como base, os casos de uso são apresentados na Figura 6.



Figura 6: Casos de uso do sistema

O sistema é composto por 3 casos de uso e 3 atores, que são:

- *UserActor*: representa o usuário que utiliza o sistema e efetua as requisições;
- *SORepositoryActor*: servidor com todos os serviços dos objetos inteligentes cadastrados; e
- *SOServerActor*: servidor responsável por enviar os comandos para os objetos inteligentes através do gateway.

Já os casos de uso, obedecendo a ordem de pré e pós execução, são:

- *SmartObjectDiscovery*: deve ser possível realizar a busca por objetos inteligentes e seus serviços através de WiFi sempre que o usuário desejar;
- *SmartObjectGUIGenerating*: com um objeto inteligente previamente escolhido a partir de uma lista, o sistema deverá ser capaz de gerar uma interface

gráfica dinamicamente utilizando algoritmo genético e apresentá-la ao usuário; e

- *SmartObjectServicesManipulation*: o usuário, através deste caso de uso, poderá interagir com o objeto inteligente manipulando seus serviços disponibilizados.

Os casos de uso apresentados nesta seção derivam dos requisitos explicitados na Seção 3.1. É possível então mapear quais requisitos originaram que casos de uso como mostra a Figura 7.

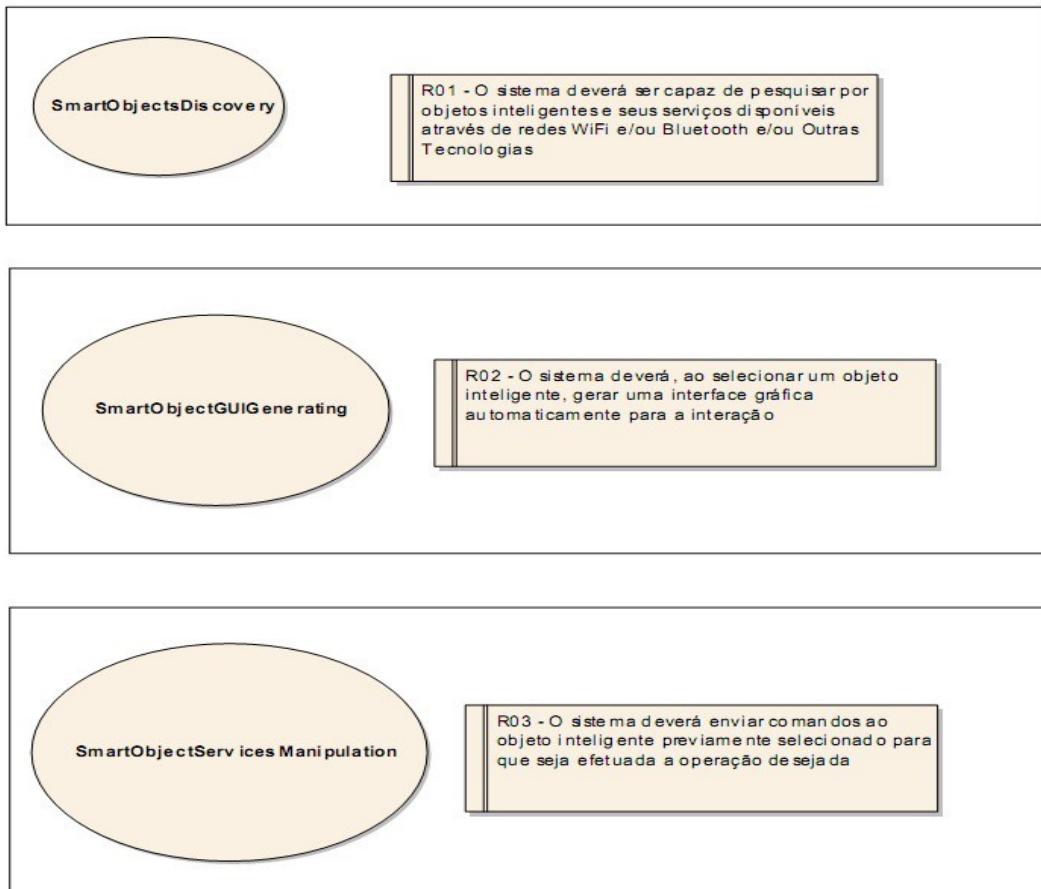


Figura 7: Rastreamento dos casos de uso.

3.4. Pacotes do Sistema

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 for-

mam um fundamental conjunto de ferramentas para a elaboração de sistemas visando os aspectos de confiabilidade, legibilidade, manutenção e flexibilidade.

Uma das vantagens ao se desenvolver sistemas separados em pacotes é o de software reusável, onde é possível chamar métodos e funções específicas a partir de diversos locais do código, descartando assim a possibilidade de ter que codificar um outro algoritmo que realize a mesma lógica. Por consequência disso a manutenção se torna eficiente pois é necessário alterar linhas de código em apenas um lugar.

Para facilitar o entendimento do trabalho, chamaremos de OI os objetos inteligentes físicos e OJ os objetos de classes escritas na linguagem Java. O diagrama representado pela Figura 8 mostra a estruturação em pacotes da aplicação desenvolvida neste trabalho e suas responsabilidades. Cada pacote será detalhado nas seções seguintes.

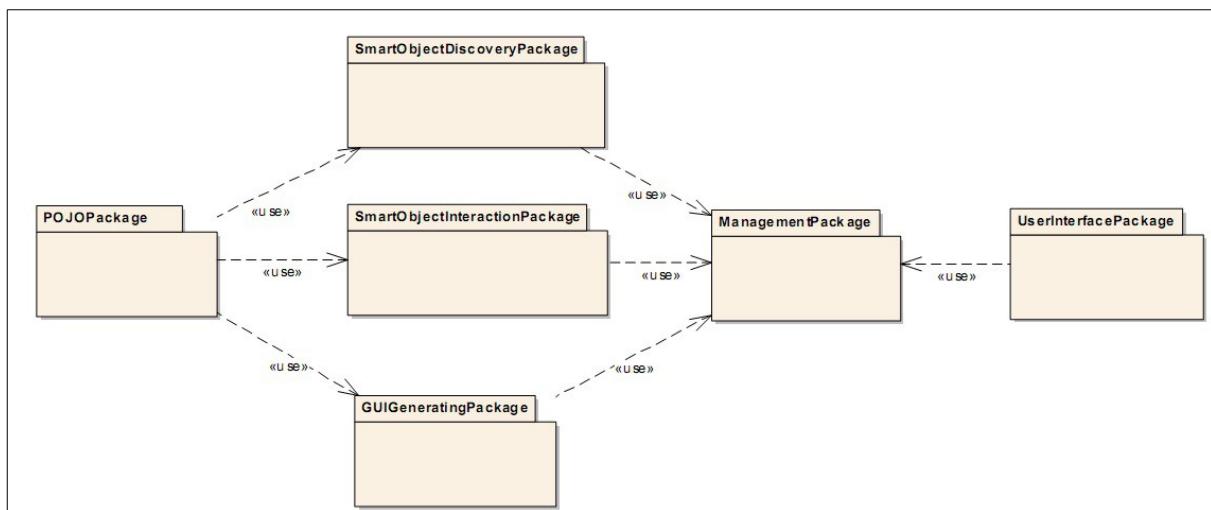


Figura 8: Diagrama de Pacotes

3.4.1. Pacote de Descoberta de Objetos Inteligentes

O *SmartObjectDiscoveryPackage* é responsável pela descoberta dos objetos inteligentes através das tecnologias WiFi, Bluetooth, entre outros (neste trabalho os esforços estão apenas no desenvolvimento de descoberta via WiFi, deixando a aplicação preparada para novas implementações). Este pacote efetua a busca por objetos inteligentes (OI) e retorna para a interface gráfica um objeto java (OJ) contendo

uma lista de OI encontrados. A Figura 9 ilustra as classes que compõem este pacote com seus respectivos métodos e atributos.

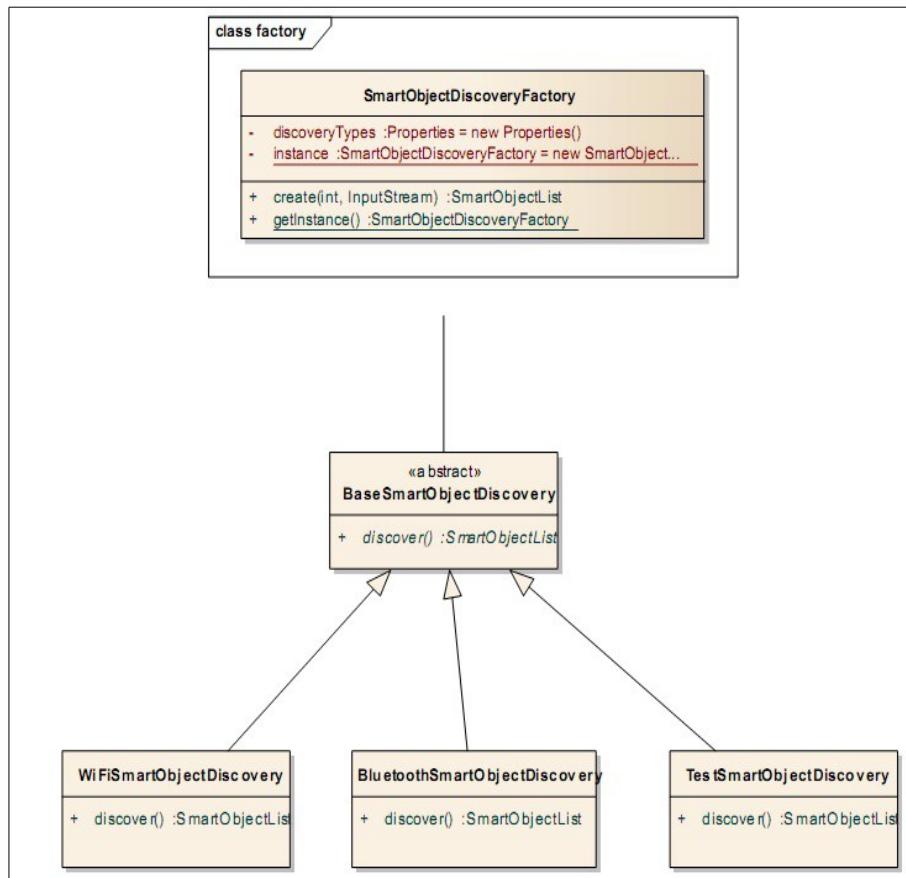


Figura 9: Classes do Pacote de Descoberta de Objetos Inteligentes

A classe inicial do sistema, `SmartObjectDiscoveryFactory`, é uma classe *Singleton*, ou seja, apenas uma instância será criada durante toda a execução da aplicação facilitando assim a reutilização dos atributos quando o usuário efetuar novas buscas.

Na invocação do seu método `discover`, será criada uma instância de `BaseSmartObjectDiscovery` que através da herança invocará o método `discover` da classe responsável pelo tipo de tecnologia selecionada (WiFi, etc.) pelo usuário para efetuar a busca por objetos inteligentes.

3.4.2. Pacote de Gerenciamento

A aplicação desenvolvida neste trabalho possui um pacote de gerenciamento, o *ManagementPackage*. Esse pacote é responsável por fazer a mediação dos dados vindos do pacote de interface do usuário com os demais pacotes do sistema. Toda requisição solicitada pelo usuário do sistema é redirecionada através deste pacote para as classes responsáveis por executar tal requisição. A classe que faz parte do pacote de gerenciamento da aplicação é ilustrada pela Figura 10.

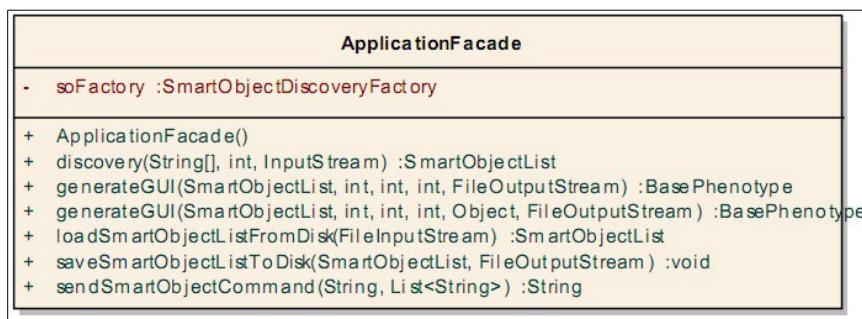


Figura 10: Classe do Pacote de Gerenciamento

A classe *ApplicationFacade* também é do tipo *Singleton*, possuindo seu método estático *getInstance*, e delega as responsabilidades para os respectivos pacotes.

O método *discovery* tem como parâmetro um array de String com os dados da tela inicial, um *int* com a tecnologia de busca utilizada (WiFi) e um *InputStream* que passa o stream do arquivo de configuração da aplicação. Esse arquivo contém diversas informações importantes para o funcionamento dinâmico e reuso de código do software. Este método solicita a instância de *SmartObjectDiscoveryFactory* e efetua a busca retornando um OJ do tipo *SmartObjectList* contendo os objetos inteligentes encontrados e seus serviços para a interface gráfica.

Os métodos *generateGUI* é utilizado para gerar a interface gráfica dos objetos inteligentes. Apenas um deles é utilizado na plataforma Android, o outro serve para outras plataformas Java.

Os métodos *saveSmartObjectToDisk* e *loadSmartObjectFromDisk* são responsáveis pelo salvamento e carregamento da lista de objetos inteligentes.

Por último, o método `sendSmartObjectCommand` pode ser chamado para enviar instruções do usuário ao objeto inteligente alterando assim o seu estado. Os campos obrigatórios são a URL e a lista contendo os parâmetros necessários para a realização do comando.

3.4.3. Pacote de Interação com o Objeto Inteligente

Para facilitar o desenvolvimento e a manutenção do sistema, o pacote de interação com os objetos inteligentes foi criado delegando às classes participantes desse pacote as responsabilidades de comunicação entre dispositivo móvel e objeto inteligente, com suas requisições e respostas, centralizando aqui a codificação acerca desse pacote. A Figura 11 apresenta as classes que compõem o pacote.

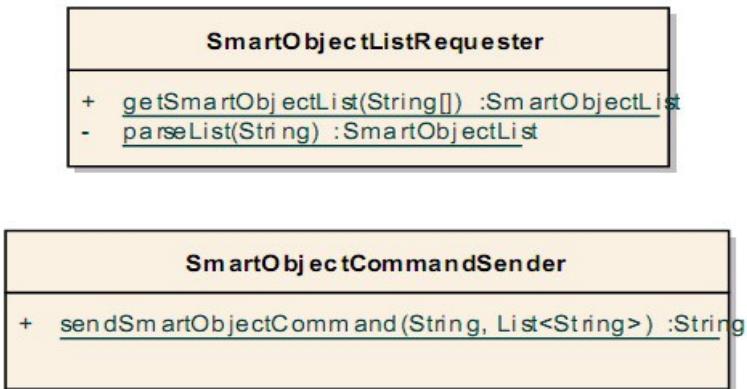


Figura 11: Classes do Pacote de Interação com Objetos Inteligentes

A classe `SmartObjectListRequester` é responsável por requisitar via protocolo HTTP os objetos inteligentes e seus serviços cadastrados em uma base de dados. A solicitação é feita enviando o `login` e senha do usuário para que somente seja apresentada uma lista com os OI que ele tiver acesso. A responsabilidade da classe `SmartObjectCommandSender` é de encaminhar os comandos ao objeto inteligente escolhido, enviando em sua requisição HTTP a URL e os parâmetros necessários.

3.4.4. Pacote 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 12. As demais classes podem ser consultadas nos apêndices desse documento.

GUIGeneratingMain	
- ARCHIEVE_SIZE :int = 5 {readOnly}	
- bestIndividual :Individual	
- bestOfAllGenerations :Population	
- context :Object	
- CROSSINGOVERRATE :double = 0.99 {readOnly}	
- DELTAFITNESS_THRESHOLD :double = 0.001 {readOnly}	
- executor :ExecutorService = Executors.newFi...	
- MUTATIONRATE :double = 0.1 {readOnly}	
- OFFSPRINGSIZE :int = 8 {readOnly}	
- PARENTSSIZE :int = 8 {readOnly}	
- POPULATIONSIZE :int = 16 {readOnly}	
- saveStream :FileOutputStream	
- screenHeight :int	
- screenWidth :int	
- so :SmartObject	
- TIMEOUTSECS :int = 900 {readOnly}	
- applyGeneticAlgorithm() :void	
+ generateGUI() :BasePhenotype	
- generateNewPopulation(Population, Population) :Population	
+ GUIGeneratingMain(SmartObject, int, int, Object)	
- removeLessEffectiveIndividuals(Population, Population) :void	
- saveToArchive(Population) :void	

Figura 12: Classes do Pacote de Geração Automática de Interface

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 in-

indivíduo possui seu grau de aptidão (*fitness*), que diz o quanto 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 os requisitos não funcionais estabelecidos anteriormente. Quando uma condição de parada for 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.

3.4.5. Pacote de Interface Gráfica

A aplicação foi desenvolvida obedecendo padrões de projeto que facilitam o reuso de código e que possibilite a utilização em outros ambientes da linguagem Java como páginas web e/ou aplicações *desktop*. Para que isso ocorra, o pacote de interface gráfica é “destacável”, ou seja, para a implementação em outro ambiente basta substituir este pacote por outro e invocar os métodos do pacote de gerenciamento passando os parâmetros necessários. As descrições a seguir são de exclusividade do desenvolvimento Android.

Todas as telas da aplicação são representadas por classes herdadas da classe *Activity*, esta classe faz parte da API do Android e é responsável pela apresentação gráfica da aplicação. Tais classes são representadas pela Figura 13.

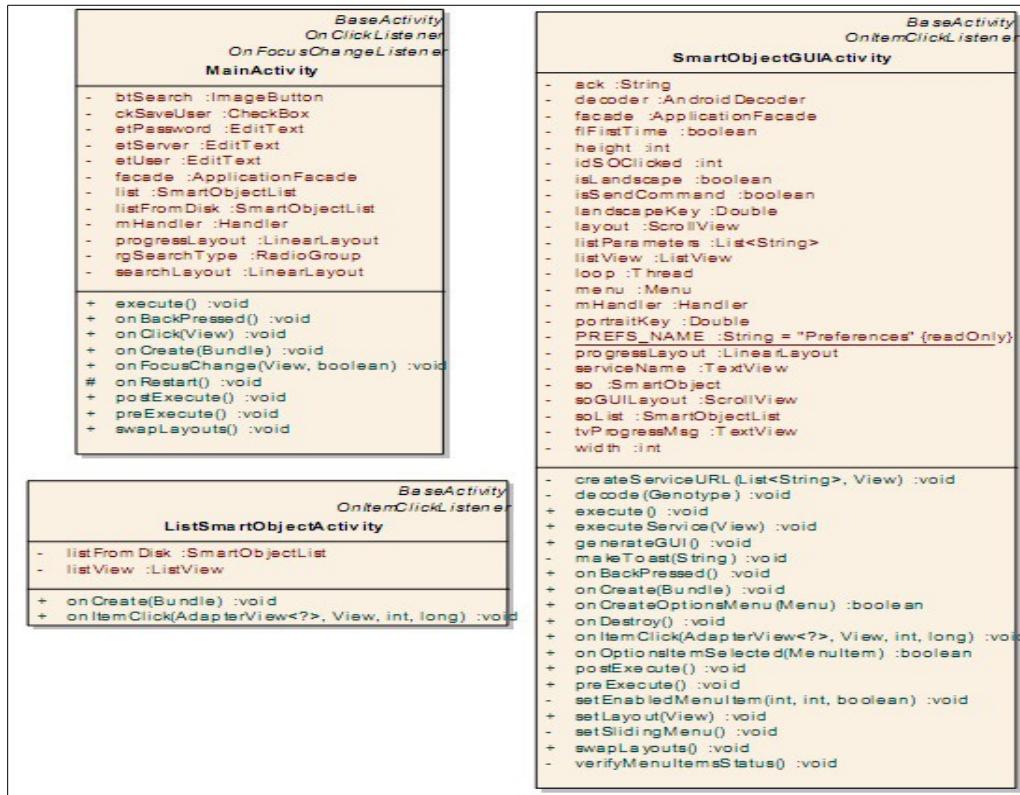


Figura 13: Classes do Pacote de Interface Gráfica

A principal tela do sistema e pela qual o aplicativo iniciará é a *MainActivity*. Esta classe sobrescreve o método da sua classe-mãe (*Activity*) *onCreate*, que tem a principal função de instanciar os objetos que compõem a tela, como os *Buttons*, *TextViews* e *Labels*, entre outros e seus eventos quando necessário. Esta tela apresenta campos de usuário e senha, URL de consulta dos objetos inteligentes, assim como um botão “Pesquisar”, que ao pressionado, aciona o seu evento *onClick* que chama o método *discoverSmartObjects* da classe gerenciadora *ApplicationFacade* e retornando um OI contendo a lista de OI encontrados.

Após a busca por OI, a aplicação é redirecionada para a classe *ListSmartObjectActivity*, que através do seu método *onCreate*, lista todos os OI encontrados e adiciona um evento de clique nos itens da lista. Ao selecionar um OI da lista, o evento *onItemClick* é invocado levando a lista de objetos inteligentes para a próxima tela.

A terceira e última tela da aplicação é a *SmartObjectGUIActivity*. A responsabilidade dessa classe é apresentar na tela todos os serviços disponíveis pelo objeto inteligente selecionado e enviar comandos através dos seus eventos.

3.4.6. Pacote de Classes POJO

A sigla POJO representa a definição *Plain Old Java Objects*. São classes que possuem apenas atributos e seus respectivos métodos *getters* e *setters* e seu construtor-padrão, não havendo nenhuma regra de negócio implementada. Em vez de utilizarmos os tipos primitivos ou objetos e seus atributos do próprio java, podemos criar classes POJO que representam objetos com os atributos específicos para a aplicação, facilitando a implementação e a manipulação dos dados. A Figura 14 representa as classes POJO criadas no presente trabalho.

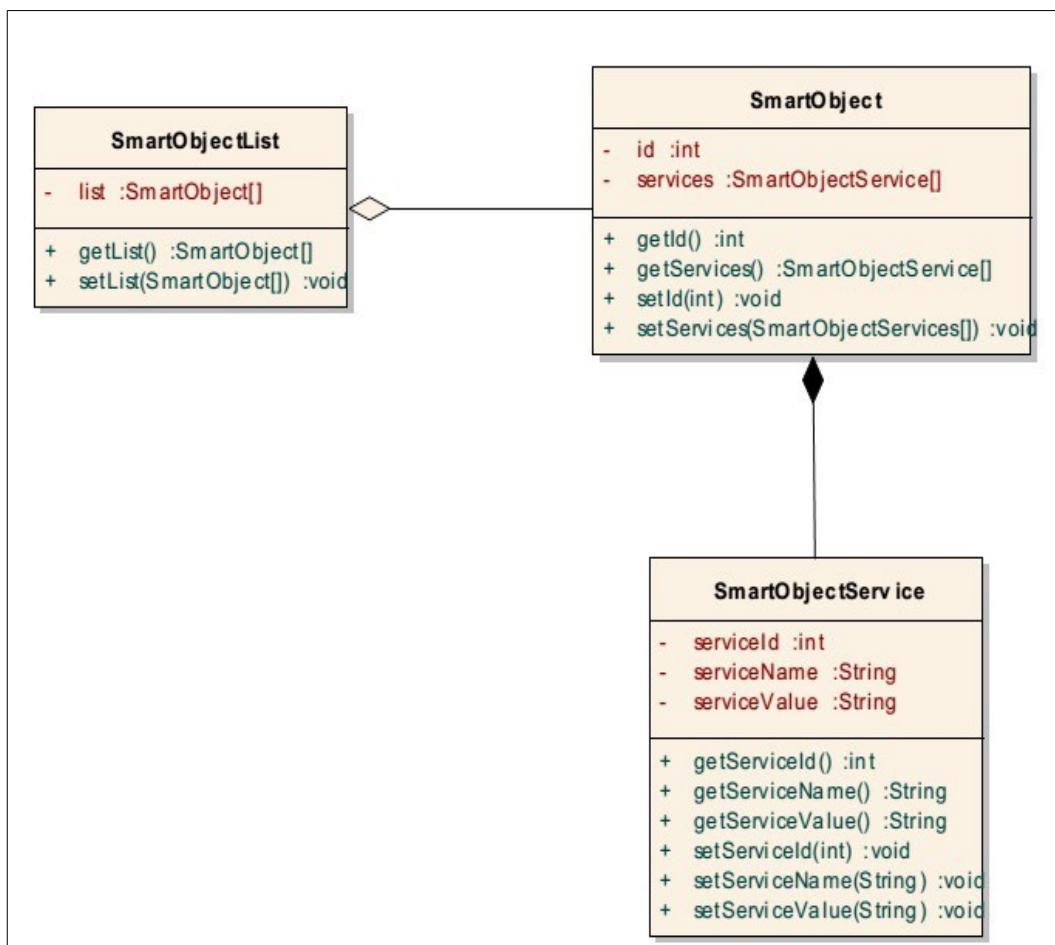


Figura 14: Classes do Pacote POJO

4. DESENVOLVIMENTO

Para o desenvolvimento do sistema proposto foi utilizada a interface de desenvolvimento Eclipse, ferramenta *open source* e recomendada para desenvolvedores Android pela própria Google. A escolha dessa IDE (*Integrated Development Environment*) foi feita pelo autor devido a experiência prévia na utilização da ferramenta e também por apresentar todas as características e funcionalidades necessárias no desenvolvimento desse sistema computacional.

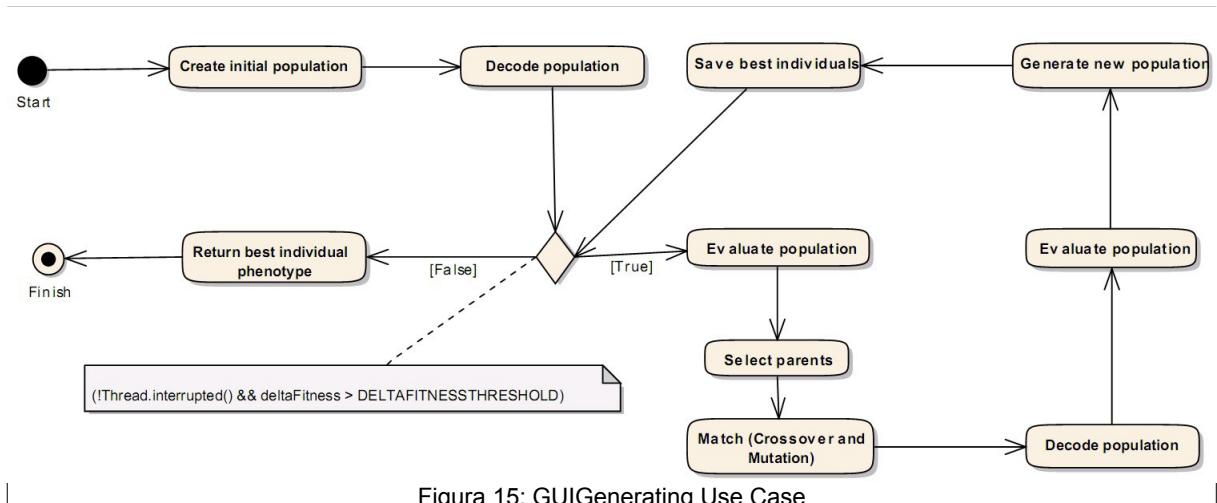
Foram utilizadas também bibliotecas *open source* de terceiros para facilitar o desenvolvimento, tornando assim possível o foco na solução do problema tratado nesse trabalho e abstraindo outras codificações que tomariam um tempo demasia-damente longo para serem implementadas. A lista abaixo indica as bibliotecas utilizadas no desenvolvimento e seus links se encontram na seção de referências bibliográficas.

- **actionbarsherlock.jar**, é uma extensão da biblioteca de suporte projetada para facilitar o uso de padrões de projeto na *action bar* (barra de menu superior utilizada nos aplicativos Android) em todas as versões do Android com uma única API.
- **library.jar**, o projeto chamado de SlidingMenu é uma biblioteca de código aberto Android que permite aos desenvolvedores criar facilmente aplicações com menus laterais deslizantes muito comuns em aplicativos móveis.
- **RXTXcomm.jar**, é uma biblioteca de suporte para implementação de comunicação via portas seriais, paralelas e USB. Baseada na API Javacomm distribuída pela própria Sun.

4.1. Algoritmo Genético

O maior foco no desenvolvimento da aplicação é a geração automática de interfaces gráficas para os objetos inteligentes. Essa geração 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 pro-

blema em questão, que no contexto desse trabalho correspondem a telas gráficas melhor dispostas. A Figura 15 exibe o diagrama de atividades que ilustra as etapas que fazem parte do algoritmo codificado.



- **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, aumentando 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.

A Figura 16 apresenta o trecho de código que corresponde ao laço principal do algoritmo genético, ou seja, ao processo de geração automática de telas, progressivamente mais adequadas. A partir desse laço, as demais funções descritas acima são invocadas.

```

while (!Thread.interrupted() && deltaFitness > DELTAFITNESSTHRESHOLD) {
    nextAverageFitness = evaluator.evaluate(population);
    deltaFitness = nextAverageFitness - averageFitness;
    averageFitness = nextAverageFitness;
    parents = matcher.selectParents(population, PARENTSSIZE);
    offspring = matcher.match(parents, OFFSPRINGSIZE, CROSSINGOVERRATE,
MUTATIONRATE);
    decoder.decode(offspring, this.context);
    evaluator.evaluate(offspring);
    population = generateNewPopulation(population, offspring);
    this.bestIndividual = population.getBestIndividual();
    saveToArchive(population);
}
Log.i("best deltafitness", " " + this.bestIndividual.getFitness());
}

```

Figura 16: GUIGenerating código

4.2. Demais Aplicações Desenvolvidas

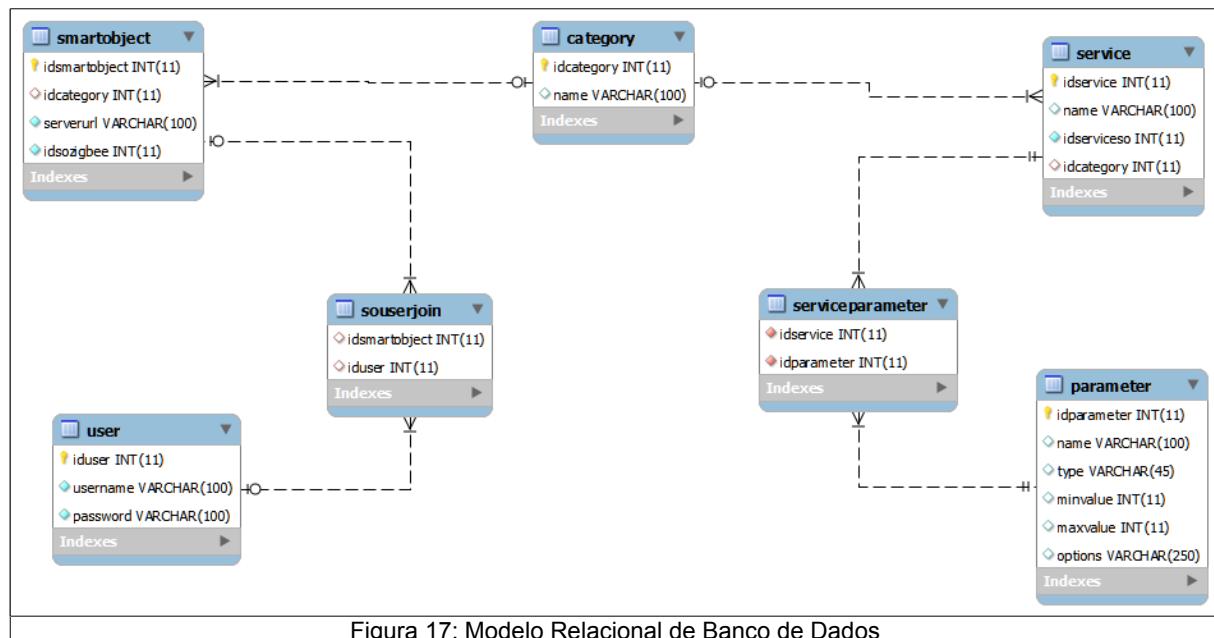
A descoberta de objetos inteligentes é feita através de uma requisição HTTP para uma aplicação web JSP utilizando método POST para maior segurança, ou seja, os parâmetros não ficam visíveis na URL, apenas no corpo da mensagem. O ambiente de desenvolvimento possui o Apache Tomcat v6.0 como servidor de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

4.2.1. Banco de Dados

O banco de dados relacional foi modelado em Oracle MySQL e normalizado de forma que diversos usuários possam acessar o mesmo objeto inteligente assim como os seus serviços também podem compartilhar parâmetros. O modelo relacional de banco de dados é representado pela Figura 17.



4.2.2. SORespository

A função principal dessa aplicação é receber requisições HTTP com métodos POST, validar usuário e senha e, se autorizado, retornar a lista de objetos inteligentes disponíveis ao usuário e os serviços de cada um. O trecho de código que retorna os objetos inteligentes é ilustrado pela Figura 18.

```

public SmartObjectList getSOList(String user) throws SQLException {
    SmartObjectList solist = new SmartObjectList();
    StringBuilder sql = new StringBuilder();
    sql.append("SELECT `SO`.`IDSMARTOBJECT`, `SO`.`IDSOGIGBEE`, `SO`.`SERVERURL`, ");
    sql.append("`CA`.`NAME`, `SER`.`IDSERVICESO`, `SER`.`NAME`, `P`.`NAME`, `P`.`TYPE`, ");
    sql.append("`P`.`MINVALUE`, `P`.`MAXVALUE`, `P`.`OPTIONS` ");
    sql.append("FROM `SODB`.`USER` U");
    sql.append("JOIN `SODB`.`SouserJoin` SOUJ ON `SOUJ`.`IDUSER` = `U`.`IDUSER` ");
    sql.append("JOIN `SODB`.`SMARTOBJECT` SO ON `SO`.`IDSMARTOBJECT` = `SOUJ`.`IDSMARTOBJECT` ");
    sql.append("JOIN `SODB`.`CATEGORY` CA ON `CA`.`IDCATEGORY` = `SO`.`IDCATEGORY` ");
    sql.append("JOIN `SODB`.`SERVICE` SER ON `SER`.`IDCATEGORY` = `CA`.`IDCATEGORY` ");
    sql.append("JOIN `SODB`.`SERVICEPARAMETER` SERPAR ON `SERPAR`.`IDSERVICE` = `SER`.`IDSERVICE` ");
    sql.append("JOIN `SODB`.`PARAMETER` P ON `P`.`IDPARAMETER` = `SERPAR`.`IDPARAMETER` ");
    sql.append("WHERE `U`.`USERNAME` = ? ");
    sql.append("ORDER BY `SO`.`IDSMARTOBJECT` ");
    sql.append("`SER`.`IDSERVICE` ");
    sql.append("`P`.`IDPARAMETER` ");
    PreparedStatement prst = conn.prepareStatement(sql.toString());
    prst.setString(1, user);
    ResultSet rs = prst.executeQuery();

    while (rs.next()) {
        solist.getList().add(getConcatValues(rs));
    }

    return solist;
}

```

Figura 18: SORepository - Descoberta de Objetos Inteligentes

4.2.3. SOServer

Os comandos enviados aos objetos inteligentes são capturados por essa aplicação web que os traduz para ASCII Modbus e envia via porta serial para o *gateway*.

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 que é utilizado na implementação dessa aplicação web. A tradução e o envio do comando para a porta serial é representada pela Figura 19.

```

private String parseCommandToASCIIModBus(String command) {
    StringBuilder modBus = new StringBuilder(":");
    List<String> params = Arrays.asList(command.split("&"));
    String AA = StringUtils.substringAfter(params.get(0), "=");
    String BB = StringUtils.substringAfter(params.get(1), "=");
    String CC = StringUtils.substringAfter(params.get(2), "=");
    String DD = "";
    int AAhex = Integer.valueOf(AA, 16);
    int BBhex = Integer.valueOf(BB, 16);
    int CChex = Integer.valueOf(CC, 16);
    int DDhex = 0x00;
    if (BBhex != 0x03) {
        DD = StringUtils.substringAfter(params.get(3), "=");
        if (DD.equalsIgnoreCase("false")) {
            DD = "00";
        } else if (DD.equalsIgnoreCase("true") && AA.equals("A1")) {
            DD = "FF";
            DDhex = 0xFF;
        } else if (DD.equalsIgnoreCase("true") && AA.equals("A2")) {
            DD = "01";
            DDhex = 0x01;
        } else {
            DDhex = Integer.valueOf(DD, 16);
        }
    }
    String irc = Integer.toHexString(((AAhex + BBhex + DDhex + CChex) ^ 0xFF) + 1) & 0xFF;
    modBus.append(AA);
    modBus.append(BB);
    modBus.append(CC);
    modBus.append(DD);
    modBus.append(irc.toUpperCase());
    return modBus.toString() + "\r\n";
}

```

Figura 19: SOServer - Tradução HTTP para ASCII ModBus

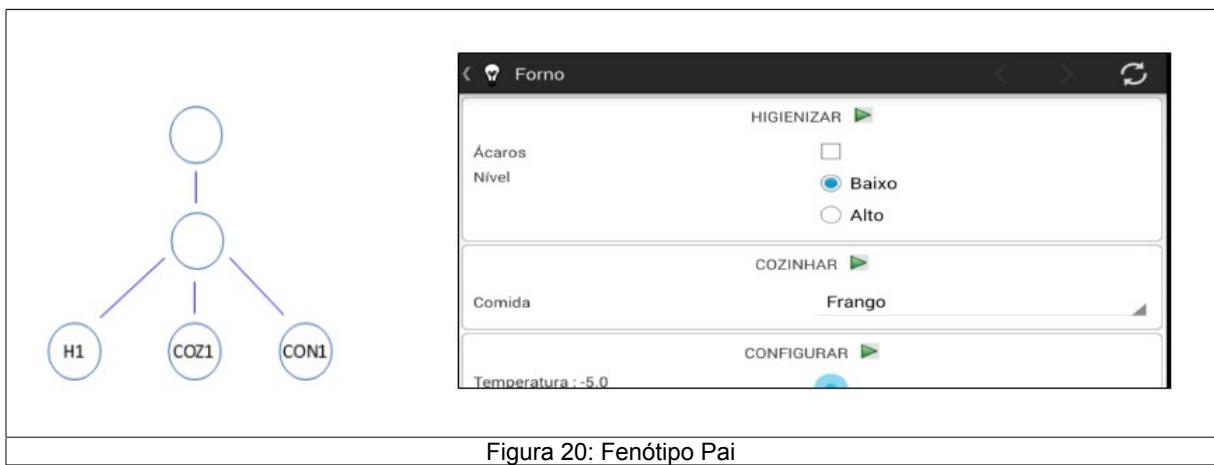
5. TESTE E VALIDAÇÃO

Foram realizados testes do algoritmo genético e também com EPOS-MOTE simulando aparelhos utilizados na automação residencial.

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 20 demonstra o fenótipo do pai, à esquerda a árvore de layouts com seus serviços e ao lado a tela por ela representada.



O fenótipo da mãe possui os mesmos serviços, porém estão dispostos de forma diferente. A Figura 21 ilustra a árvore e a tela desse fenótipo.

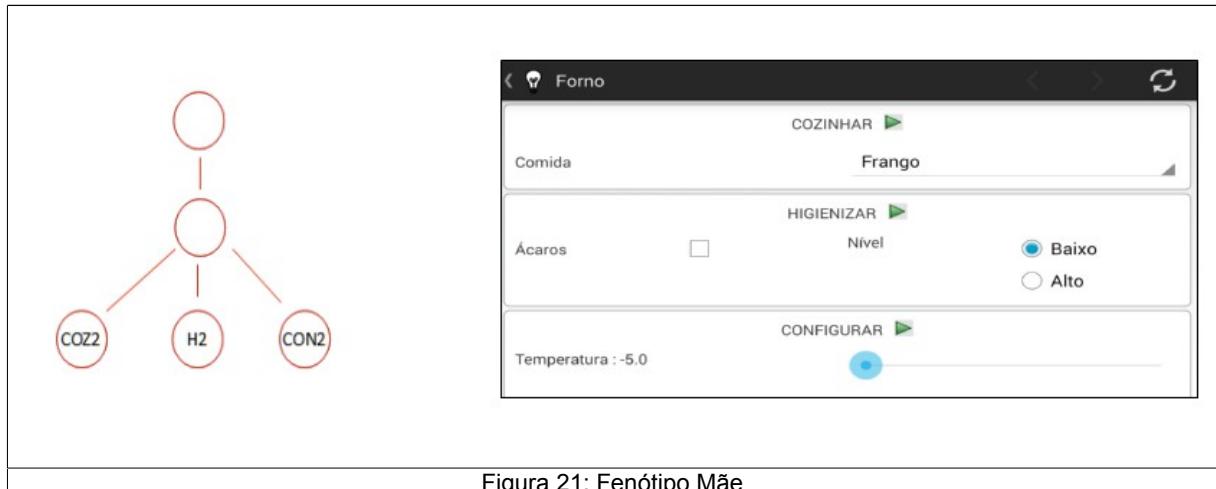


Figura 21: Fenótipo Mãe

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 22.



Figura 22: 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 23.



Figura 23: Fenótipo Final após Mutação

5.2. Autenticação de Usuário

A tela representada pela Figura 24 mostra os dados iniciais para descoberta de objetos inteligentes. Neste teste a senha está propositalmente errada para verificar a autenticação de usuário.



Figura 24: Tela inicial com dados incorretos

Ao clicar no botão de pesquisar, a aplicação web SORespository é chamada consultando o banco de dados para verificação do usuário. O log é ilustrado pela Figura 25.

```

Problems @ Javadoc Declaration Console Pending Changes Call Hierarchy Progress LogCat Search History Synchronize
Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw.exe (07/06/2014 10:23:33)
INFO: Starting Coyote HTTP/1.1 on http-8080
07/06/2014 10:23:34 org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
07/06/2014 10:23:34 org.apache.jk.server.JkMain start
INFO: JK running ID=0 time=0/12 config=null
07/06/2014 10:23:34 org.apache.catalina.startup.Catalina start
INFO: Server startup in 455 ms
ercilio:batatinha2
Validating user...
07/06/2014 10:23:46 SQL - VALIDATE USER: SELECT `U`.`USERNAME` FROM `SODB`.`USER` U WHERE `U`.`USERNAME` = 'ercilio' AND `U`.`PASSWORD` = 'batatinha2'

```

Figura 25: Log de usuário inválido

Feita a verificação, o usuário é notificado de que os dados estão incorretos.

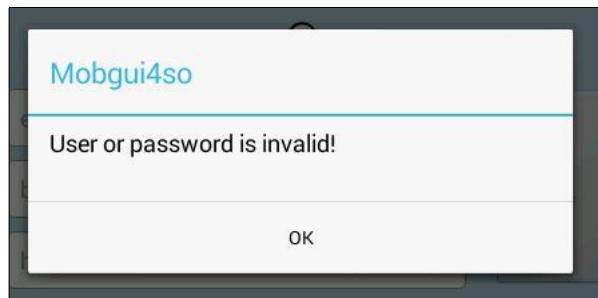


Figura 26: Mensagem de usuário inválido

5.3. Descoberta de Objetos Inteligentes

Nesse teste, os dados foram colocados corretamente para que seja efetuada a busca por objetos inteligentes. A Figura 27 mostra os dados corretos.

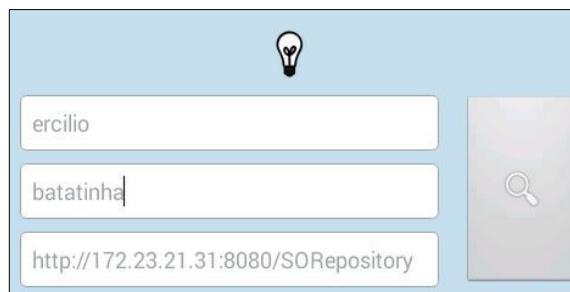


Figura 27: Tela inicial com dados corretos

A aplicação SORepository faz a validação dos dados e efetua uma pesquisa no banco de dados dos objetos inteligentes cadastrados para o usuário solicitado.

```

INFO: JK: ajp13 listening on /0.0.0.0:8009
07/06/2014 10:24:17 org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/11 config=null
07/06/2014 10:24:17 org.apache.catalina.startup.Catalina start
INFO: Server startup in 471 ms
erciliobatatinha
Validating user...
07/06/2014 10:24:26 SQL - VALIDATE USER: SELECT `U`.`USERNAME` FROM `SODB`.`USER` U WHERE `U`.`USERNAME` = 'ercilio' AND `U`.`PASSWORD` = 'batatinha'
Getting smart object list...
07/06/2014 10:24:26 SQL - GET SO LIST: SELECT `SO`.`IDSMARTOBJECT`, `SO`.`IDSOMODBUS`, `SO`.`SERVERURL`, `CA`.`NAME`, `SER`.`IDSERVICEMODBUS`, `SER`.`IDR

```

Figura 28: Log de descoberta de objetos inteligentes

A consulta retornada pelo banco de dados representa todos os objetos inteligentes cadastrados para o usuário e seus respectivos serviços.

	IDSMAR	IDSMODBL	SERVERURL	NAME	IDSERVICEMC	IDREGISTERMC	NAME	NAME	TYPE	MINVAL	MAXVAL	OPTIONS
►	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	05	00	Ligar	Ligado	boolean	NULL	NULL	NULL
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	16	00	Configurar	Temperatura	double	18	28	NULL
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	16	00	Configurar	Humidade	double	0	100	NULL
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	16	00	Configurar	Velocidade	combo	NULL	NULL	Fraco Médio Forte
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	16	00	Configurar	Timer	double	0	7	NULL
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	0	00	Tipo de Função	Função	combo	NULL	NULL	Circular Refrigerar Aquecer
	1	A1	http://192.168.43.138:8080/SOserver/...	Ar Condicionado	03	00	Pegar Temperatura	Temperatura	get	NULL	NULL	NULL
	3	A2	http://192.168.43.138:8080/SOserver/...	Lâmpada	05	00	Acender	Aceso	boolean	NULL	NULL	NULL
	5	A3	http://192.168.43.138:8080/SOserver/...	CO2	03	00	Quantidade de CO2	CO2	get	NULL	NULL	NULL

Figura 29: Resultado do banco de dados

Os objetos inteligentes encontrados são mostrados em uma lista para o usuário.

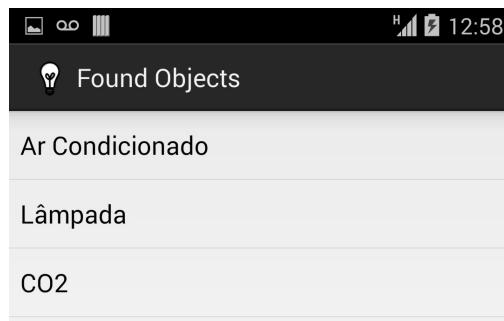


Figura 30: Lista de OIs

5.4. Geração de Telas

Diferentes telas são geradas de acordo com a posição do dispositivo móvel (Retrato ou Paisagem), as setas possibilitam ao usuário navegar entre diferentes telas e o ícone de “atualizar” permite gerar novas telas. As Figuras 31 e 32 representam as telas geradas para o Ar Condicionado e a Lâmpada respectivamente.

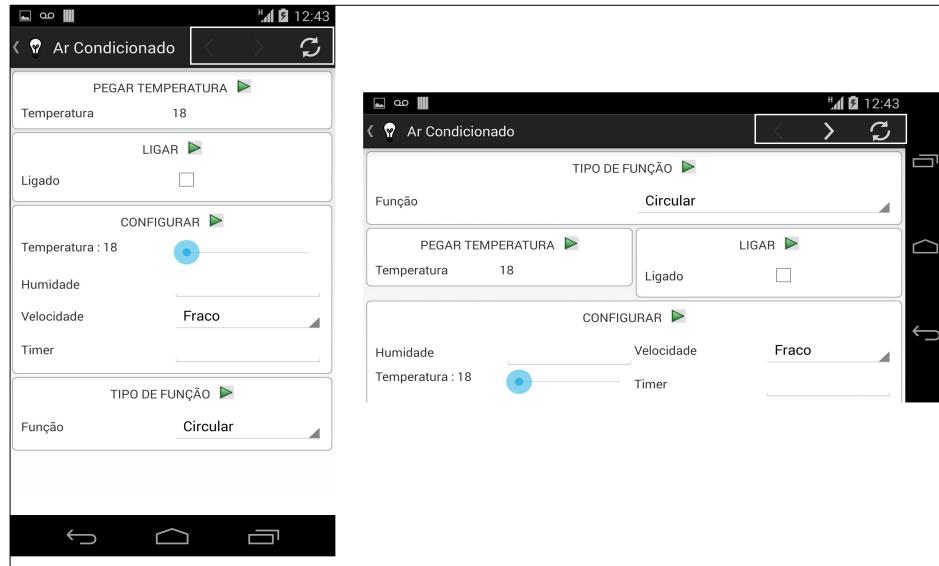


Figura 31: Telas geradas para o Ar Condicionado

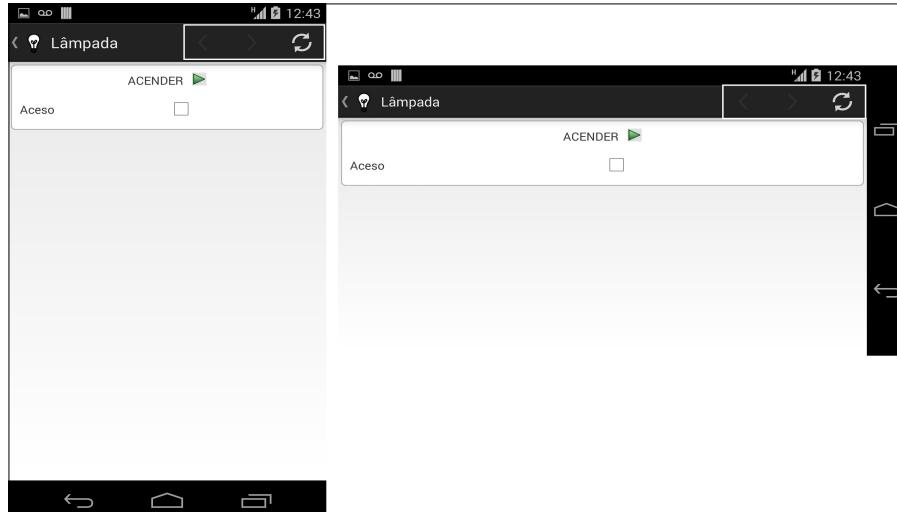


Figura 32: Telas geradas para a Lâmpada

5.5. Interação com o Objeto Inteligente

Foram realizados testes com um EPOS-MOTE que simula um Ar Condicionado e outro que simula uma Lâmpada. 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.

5.5.1. Ar Condicionado

5.5.1.1. Ligar

A Figura 41 ilustra o serviço “Ligar” do Ar Condicionado.



O *gateway* é instanciado na porta COM9; Os comandos são traduzidos pelo SOSServer para ASCII Modbus e enviados via porta serial para o *gateway*, que por sua vez envia as instruções via rede sem fio para o Ar Condicionado. A Figura 5 na seção de Projeto relembra a topologia utilizada.

```
Console X Tasks Call Hierarchy LogCat JUnit Search Progress History
Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw.exe (0)
INFO: Jk running ID=0 time=0/12 config=null
07/06/2014 10:36:32 org.apache.catalina.startup.Catalina start
INFO: Server startup in 473 ms
Stable Library
=====
Native lib Version = RXTX-2.1-7
Java lib Version = RXTX-2.1-7
Server is listening port //./COM9
Command: idSOModbus=A1&idServiceModbus=05&idRegisterModbus=00&Ligado=true&
Parsing command to ASCII Modbus...
Command parsed: :A10500FFSB\r\n
Sending smart object instructions...
```

Figura 34: Log do comando "Ligar" do Ar Condicionado

O LED verde indica que o Ar Condicionado está ligado.



Figura 35: Ar Condicionado ligado

5.5.1.2. Aumentar e Diminuir a Temperatura

As figuras dessa subseção estabelecem a mesma sequência, porém para aumentar e diminuir a temperatura do Ar Condicionado. O LED vermelho indica que a temperatura foi aumentada e o azul que foi diminuída.



Figura 36: Serviço "Aumentar" e "Diminuir" temperatura do Ar Condicionado

Os logs são representados pela Figura 45.

```
Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw.exe
=====
Native lib Version = RXTX-2.1-7
Java lib Version = RXTX-2.1-7
Server is listening port //./COM9
Command: idSMODbus=A1&idServiceModbus=05&idRegisterModbus=00&Ligado=true&
Parsing command to ASCII Modbus...
Command parsed: :A10500F5FB\r\n
Sending smart object instructions...
Command: idSMODbus=A1&idServiceModbus=16&idRegisterModbus=00&Temperatura=21.
Parsing command to ASCII Modbus...
Command parsed: :A116001534\r\n
Sending smart object instructions...
Command: idSMODbus=A1&idServiceModbus=16&idRegisterModbus=00&Temperatura=19
Parsing command to ASCII Modbus...
Command parsed: :A116001336\r\n
Sending smart object instructions...
```

Figura 37: 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 38: Ar Condicionado com a temperatura alterada

5.5.2. Lâmpada

5.5.2.1. Ligar e Desligar

Os passos se repetem para a Lâmpada. O LED vermelho indica que a Lâmpada está desligada e o verde que está ligada.

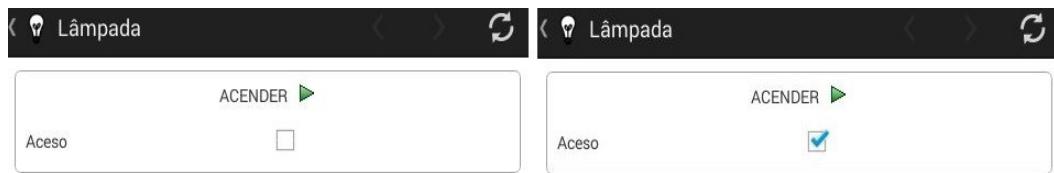


Figura 39: Serviços "Desligar" e "Ligar" Lâmpada

Os logs abaixo indicam que tanto a comunicação quanto a tradução para a sintaxe ASCII Modbus estão sendo feitos da forma correta.

```

Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw
Command: idSMModbus=A2&idServiceModbus=05&idRegisterModbus=008Aceso=false&
Parsing command to ASCII Modbus...
Command parsed: :A205000059\r\n
Sending smart object instructions...

Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw
Command: idSMModbus=A2&idServiceModbus=05&idRegisterModbus=008Aceso=true&
Parsing command to ASCII Modbus...
Command parsed: :A205000158\r\n
Sending smart object instructions...

```

Figura 40: Logs de Desligar e Ligar Lâmpada

Os LEDs indicam que o comando foi recebido e executado corretamente, explicitando assim que há verdadeiramente uma comunicação entre *smartphone* e EPOS-MOTES atendendo os objetivos do trabalho.



Figura 41: Lâmpada desligada e ligada

6. CONCLUSÕES E TRABALHOS FUTUROS

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.

Os objetivos propostos nesse trabalho de conclusão de curso foram completamente alcançados e o desenvolvimento foi concluído seguindo todos os aspectos descritos durante o trabalho, constatando-se a eficiência de um sistema pra dispositivos móveis capaz de gerar interfaces gráficas amigáveis aos usuários no âmbito da automação residencial.

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.

Os resultados obtidos pelo presente trabalho podem ser consideravelmente ampliados através de uma busca por novas tecnologias e inclusão de novas funcionalidades. Com base nas limitações da versão atual desenvolvida, sugere-se a realização dos seguintes trabalhos futuros:

Realizar *streaming* de áudio e vídeo dos objetos inteligentes tais como câmeras e microfones; Implementar uma consulta automática dos serviços dos objetos inteligentes para casos necessários como detecção de variação de temperatura, luminosidade, pressão entre outros; Internacionalizar o software para as línguas mais comuns, deixando assim a aplicação personalizável e de agrado ao usuário; e Alterar a *skin* do aplicativo para padrões utilizados por aplicativos consolidados e de grande porte, entre eles Google, Facebook e Twitter.

REFERÊNCIAS BIBLIOGRÁFICAS

- 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.
- HSIEH, Meng-Yen, et al. A Mobile Application Framework For Rapid Integration of Ubiquitous Web Services. **International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing**. 2012. p. 136 – 142.
- TRIVEDI, Nihar; LAI, Wei; ZHANG, Zhongwei. **Optimizing Windows Layout by Applying a Genetic Algorithm**. 2001. p. 431 – 435.
- HAN, Dae-Man; LIM, Jae-Hyun. **Smart Home Energy Management System using IEEE 802.15.4 and ZigBee**. 2010. p. 1403 – 1410.
- FARELLA, Elisabetta. Ambient Intelligence, Smart Objects and Sensor Networks: practical experiences. **International Conference on Embedded and Ubiquitous Computing**. 2010. p. 416 – 421.
- LIU, Jianmimg, et al. **Applications of Internet of Things on Smart Grids in China**. 2011. p. 13 – 17.
- 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.
- IZQUIERDO, Paoli, et al. An Annotation Tool for Enhancing The User Interface Generation Process for Services. **International Crimean Conference “Microwave & Telecommunication Technology”**. Ucrânia, 2009. p. 372 – 374.
- MORI, Takanori; NONAKA, Takako; HASE, Tomohiro. Automatic GUI Generation on AV Remote Control Using Genetic Algorithm. **International Symposium on Consumer Electronics**. 2010.
- 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.

CURBERA, F, et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. **Internet Computing.** Abr. 2002. p. 86 – 93.

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.

ERIKSSON, Hans-Erik et al. **UML 2 toolkit.** Wiley. com, 2004.

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.

Wesley Moraes. Android Domina Vendas no Brasil, <<http://www.guiaopc.com.br/noticias/29476/android-domina-vendas-brasil.html>>. Acessado em: 05 de Maio de 2013. Às 15:48hs.

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

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

SILVA, R. **UML2: em modelagem orientada a objetos.** Florianópolis: Visual Books, 2007. 232p.

SILVA, R. **Como modelar com UML 2.** Florianópolis, SC: Visual Books, 2009. 320p.

BORATTI, I. **Programação Orientada a Objetos em Java.** Florianópolis: VisualBooks. 2007.

BORATTI, I; OLIVEIRA, A. **Introdução a Programação – Algoritmos.** VisualBooks. 3 Ed. 2007.

DEITEL, H; DEITEL, P. **Java: como programar.** 3. ed. Porto Alegre: Bookman, 2001.

ANEXOS

Anexo A - Artigo

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.

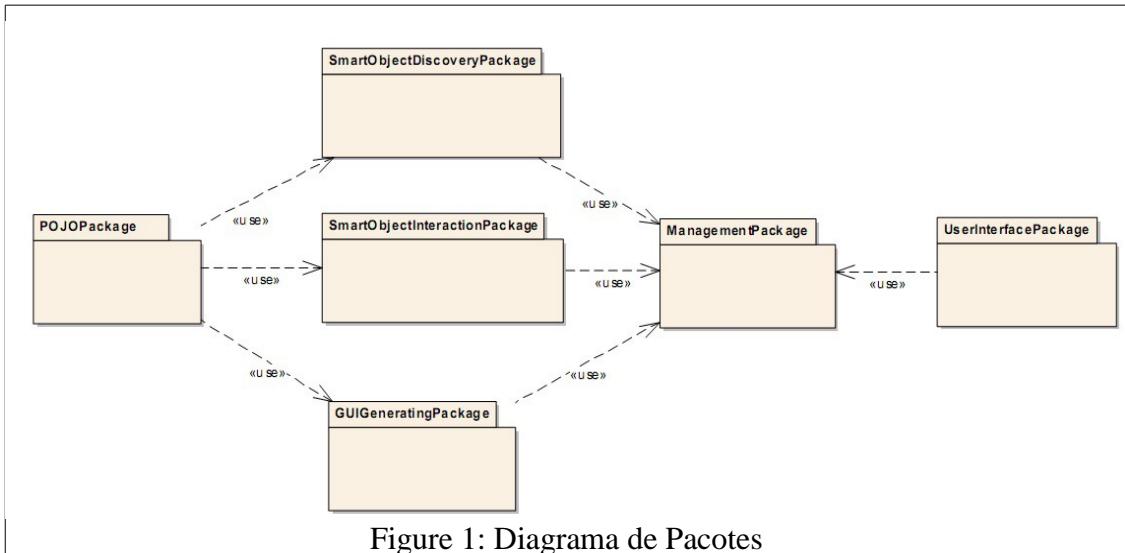


Figure 1: Diagrama de Pacotes

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-las no documento, a classe mais importante desse pacote é demonstrada pela Figura 2.

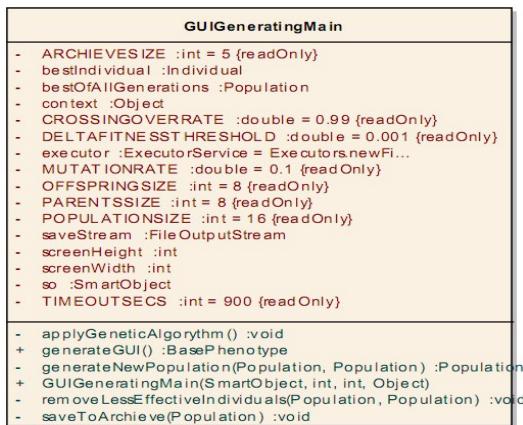


Figure 2: Classes do Módulo de Geração Automática de GUI

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 quanto 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.

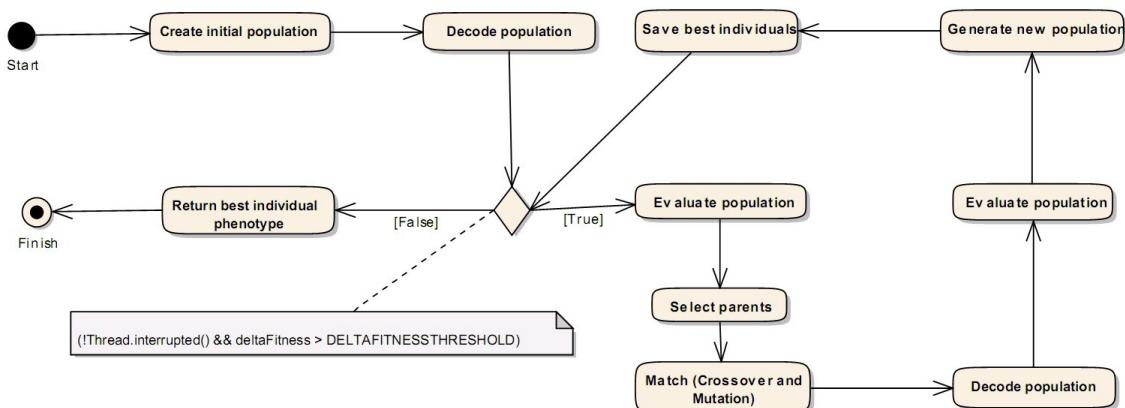


Figure 3: Diagrama de Atividades da Geração de Tela

- **Create initial population:** etapa que cria a população inicial com seus indivíduos. Cada indivíduo possui seu genótipo, fenótipo e aptidão (taxa que diz quanto 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, aumentando 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.

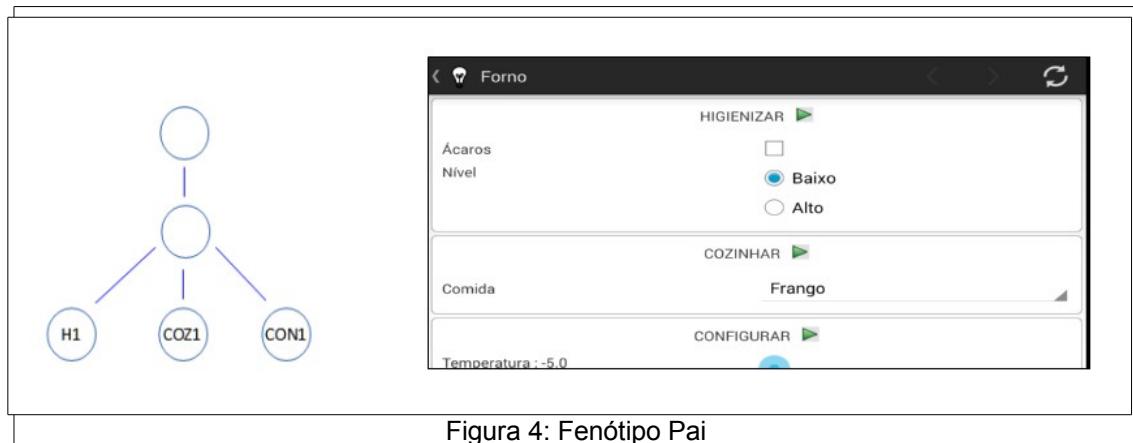


Figura 4: Fenótipo Pai

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.

```

Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files (x86)\Java\jdk1.6.0_29\bin\javaw.exe
=====
Native lib Version = RXTX-2.1-7
Java lib Version = RXTX-2.1-7
Server is listening port //./COM9
Command: id$Modbus=A1&idServiceModbus=05&idRegisterModbus=00&Ligado=true&
Parsing command to ASCII Modbus...
Command parsed: :A10500FF5B\r\n
Sending smart object instructions...
Command: id$Modbus=A1&idServiceModbus=16&idRegisterModbus=00&Temperatura=21
Parsing command to ASCII Modbus...
Command parsed: :A116001534\r\n
Sending smart object instructions...
Command: id$Modbus=A1&idServiceModbus=16&idRegisterModbus=00&Temperatura=19
Parsing command to ASCII Modbus...
Command parsed: :A116001336\r\n
Sending smart object instructions...

```

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.

Anexo B – Código Fonte Mobgui4so

ApplicationFacade.java

```
1 /**
4 package br.com.mobgui4so.controller;
5
6 import java.io.InputStream;
20
21 /**
22 * @author Ercilio Nascimento
23 */
24 public class ApplicationFacade {
25
26     private SmartObjectDiscoveryFactory soFactory;
27
28     public ApplicationFacade() {
29         soFactory = SmartObjectDiscoveryFactory.getInstance();
30     }
31
32     public SmartObjectList discovery(String[] form, int discoveryType, InputStream properties) throws IOException, ClassNotFoundException {
33         BaseSmartObjectDiscovery discover = soFactory.create(discoveryType, properties);
34         SmartObjectList list = discover.execute(form);
35         // this.saveSmartObjectListToDisk(form, list, saveStream);
36         // SmartObjectList so = this.loadSmartObjectList(save);
37         return list;
38     }
39
40     public void saveSmartObjectListToDisk(SmartObjectList list, FileOutputStream saveStream) {
41         /*list.setServerURL(form[2]);
42         if (Boolean.parseBoolean(form[3])) {
43             list.setUser(form[0]);
44             list.setPassword(form[1]);
45             list.setCKSaveUser(true);
46         }*/
47         ObjectOutputStream os;
48         try {
49             os = new ObjectOutputStream(saveStream);
50             os.writeObject(list);
51             os.close();
52         } catch (IOException e) {
53             e.printStackTrace();
54         }
55     }
56
57     public SmartObjectList loadSmartObjectListFromDisk(FileInputStream loadStream) throws IOException, ClassNotFoundException {
58         ObjectInputStream is = new ObjectInputStream(loadStream);
59         SmartObjectList soList = (SmartObjectList) is.readObject();
60         is.close();
61         return soList;
62     }
63
64 /**
65 * Use it for Non-Android platforms
66 */
67 public BasePhenotype generateGUI(SmartObjectList soList, int soIndex, int screenWidth, int screenHeight, FileOutputStream saveStream) {
68     return generateGUI(soList, soIndex, screenWidth, screenHeight, null, saveStream);
69 }
70
71 /**
72 * Use it just for Android platforms
73 */
74 public BasePhenotype generateGUI(SmartObjectList soList, int soIndex, int screenWidth, int screenHeight, Object context, FileOutputStream saveStream) {
75     BasePhenotype phenotype = new GUIGeneratingMain(soList.getList().get(soIndex), screenWidth, screenHeight, context).generateGUI();
76     saveSmartObjectListToDisk(soList, saveStream);
77
78     return phenotype;
79 }
80
81     public String sendSmartObjectCommand(String url, List<String> params) {
82         return SmartObjectCommandSender.sendSmartObjectCommand(url, params);
83     }
84 }
85 }
```

BaseSmartObjectDiscovery.java

```
1 /**
4 package br.com.mobgui4so.model.discovery;
5
6 import java.io.IOException;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public abstract class BaseSmartObjectDiscovery {
14     public abstract SmartObjectList execute(String[] form) throws IOException;
16
17 }
18
```

WiFiSmartObjectDiscovery.java

```
1 /**
4 package br.com.mobgui4so.model.discovery;
5
6 import java.io.IOException;
10
11 /**
12 * @author Ercilio Nascimento
13 */
14 public class WiFiSmartObjectDiscovery extends BaseSmartObjectDiscovery {
15
16     @Override
17     public SmartObjectList execute(String[] form) throws IOException {
18         return SmartObjectListRequester.getSmartObjectList(form);
19     }
20
21 }
22
```

SmartObjectDiscoveryFactory.java

```
1 /**
2  * package br.com.mobgui4so.model.factory;
3  *
4  * import java.io.InputStream;
5  *
6  * /**
7  * * @author Ercilio Nascimento
8  */
9  *
10 public class SmartObjectDiscoveryFactory {
11
12     private Properties discoveryTypes = new Properties();
13
14     private static SmartObjectDiscoveryFactory instance = new SmartObjectDiscoveryFactory();
15
16     public static SmartObjectDiscoveryFactory getInstance() {
17         return instance;
18     }
19
20     public BaseSmartObjectDiscovery create(int discoveryType, InputStream properties) {
21         String typeName = "";
22         switch (discoveryType) {
23             case 1:
24                 typeName = "WiFi";
25                 break;
26             default:
27                 typeName = "WiFi";
28                 break;
29         }
30         try {
31             discoveryTypes.load(properties);
32             String stringClass = discoveryTypes.getProperty(typeName);
33             Class<?> discoveryClass = Class.forName(stringClass);
34             return (BaseSmartObjectDiscovery) discoveryClass.newInstance();
35         } catch (Exception e) {
36             e.printStackTrace();
37         }
38     }
39
40     return null;
41 }
42 }
43 }
44 }
```

ChildrenList.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating;
5
6 import java.io.Serializable;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 @SuppressWarnings("serial")
13 public class ChildrenList extends ArrayList<Gene> implements Serializable {
14
15     public ChildrenList clone() {
16         ChildrenList list = new ChildrenList();
17         for (Gene gene : this) {
18             list.add(gene.clone());
19         }
20
21         return list;
22     }
23 }
24
```

```

1 /**
4 package br.com.mobgui4so.model.guigenerating;
5
6 import java.io.Serializable;
12
13 /**
14 * @author Ercilio Nascimento
15 */
16 public class Gene implements Serializable {
17     private int id;
18     private ParamType paramType;
19     private BaseCodon codon;
20     private int lastChildRead;
21     private boolean isParamLayout = false;
22     private ChildrenList children = new ChildrenList();
23
24     public enum ParamType {
25         BUTTON,
26         COMBO,
27         LAYOUT,
28         BOOLEAN,
29         STRING,
30         INT,
31         DOUBLE,
32         VOID,
33         GET
34     }
35
36
37     public Gene() {
38         this.lastChildRead = -1;
39     }
40
41     public Gene(ParamType att) {
42         this(att, null, 0, 0, null);
43     }
44
45     public Gene(ParamType att, String name) {
46         this(att, name, 0, 0, null);
47     }
48
49     public Gene(ParamType att, String name, int minValue, int maxValue, String options) {
50         this.lastChildRead = -1;
51         this.paramType = att;
52         switch (att) {
53             case LAYOUT:
54                 this.codon = new LayoutCodon(name);
55                 break;
56             case COMBO:
57                 this.codon = new ComboCodon(name, options);
58                 break;
59             case BOOLEAN:
60                 this.codon = new BaseCodon(name, minValue, maxValue);
61                 break;
62             case DOUBLE:
63                 this.codon = new BaseCodon(name, minValue, maxValue);
64                 break;
65             case INT:
66                 this.codon = new BaseCodon(name, minValue, maxValue);
67                 break;
68             case GET:
69                 this.codon = new BaseCodon(name, minValue, maxValue);
70                 break;
71             default:
72                 break;
73         }
74     }
75
76     public void setBaseCodon(BaseCodon codon) {
77         this.codon = codon;
78     }
79
80     public Gene clone() {
81         Gene gene = new Gene();
82         gene.setId(this.id);
83         gene.setParamType(this.paramType);
84         gene.setBaseCodon(this.codon.clone());
85         gene.setLastChildRead(this.lastChildRead);
86         gene.setParamLayout(this.isParamLayout);
87         gene.setChildren(children.clone());
88
89         return gene;
90     }
91
92     public boolean isParamLayout() {
93         return isParamLayout;
94     }
95
96     public void setParamLayout(boolean isParamLayout) {
97         this.isParamLayout = isParamLayout;
98     }
99
100    public void removeChild(int index) {
101        this.children.remove(index);
102    }
103
104    public Gene getChild(int index) {
105        return this.children.get(index);
106    }
107
108    public int getNumberOfChildren() {
109        return this.children.size();
110    }
111}

```

Gene.java

```
112     public ParamType getParamType() {
113         return paramType;
114     }
115
116     public void setParamType(ParamType paramType) {
117         this.paramType = paramType;
118     }
119
120     public BaseCodon getCodon() {
121         return codon;
122     }
123
124     public void setCodon(BaseCodon codon) {
125         this.codon = codon;
126     }
127
128     public void addChild(Gene son) {
129         this.children.add(son);
130     }
131
132     public void removeChild(Gene son) {
133         this.children.remove(son);
134     }
135
136     public Gene getNextChild() {
137         return this.children.get(++lastChildRead);
138     }
139
140     public int getLastChildRead() {
141         return lastChildRead;
142     }
143
144     public void setLastChildRead(int lastChildRead) {
145         this.lastChildRead = lastChildRead;
146     }
147
148     public int getId() {
149         return id;
150     }
151
152     public void setId(int id) {
153         this.id = id;
154     }
155
156     public List<Gene> getChildren() {
157         return children;
158     }
159
160     public void setChildren(ChildrenList children) {
161         this.children = children;
162     }
163
164     private boolean isLeaf() {
165         boolean isLeaf = false;
166         if (this.children.isEmpty()) {
167             isLeaf = true;
168         }
169
170         return isLeaf;
171     }
172 }
```

Genotype.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating;
5
6 import java.io.Serializable;
10
11 /**
12 * @author Ercilio Nascimento
13 */
14 public class Genotype implements Serializable {
15     private Gene root;
16
17     public Gene getRoot() {
18         return root;
19     }
20
21     public void setRoot(Gene root) {
22         this.root = root;
23     }
24
25     public Gene createRoot() {
26         Gene root = new Gene(ParamType.LAYOUT, "");
27         LayoutCodon codon = (LayoutCodon) root.getCodon();
28         codon.setMutable(true);
29         root.setId(0);
30         this.root = root;
31
32         return this.root;
33     }
34
35     public Genotype clone() {
36         Genotype genotype = new Genotype();
37         genotype.setRoot(root.clone());
38
39         return genotype;
40     }
41 }
42
```

```

1 /**
4 package br.com.mobgui4so.model.guigenerating;
5
6 import java.io.FileOutputStream;
28
29 /**
30 * @author Ercilio Nascimento
31 */
32 public class GUIGeneratingMain {
33
34     private ExecutorService executor = Executors.newFixedThreadPool(1);
35     private final int TIMEOUTSECS = 900;
36     private final int POPULATIONSIZE = 16;
37     private final double MUTATIONRATE = 0.1;
38     private final double CROSSINGOVERRATE = 0.99;
39     private final int PARENTSSIZE = 8;
40     private final int ARCHIEVESIZE = 5;
41     private final int OFFSPRINGSIZE = 8;
42     private final double DELTAFITNESSTHRESHOLD = 0.001;
43     private SmartObject so;
44     private Individual bestIndividual;
45     private int screenWidth;
46     private int screenHeight;
47     private Object context;
48     private Population bestOfAllGenerations;
49     private FileOutputStream saveStream;
50
51     public GUIGeneratingMain(SmartObject so, int screenWidth, int screenHeight, Object context) {
52         this.so = so;
53         this.screenHeight = screenHeight;
54         this.screenWidth = screenWidth;
55         this.context = context;
56     }
57
58     public BasePhenotype generateGUI() {
59         final Future<?> future = executor.submit(new Runnable() {
60             public void run() {
61                 try {
62                     applyGeneticAlgorythm();
63                 } catch (Exception e) {
64                     throw new RuntimeException(e);
65                 }
66             }
67         });
68         try {
69             future.get(this.TIMEOUTSECS, TimeUnit.SECONDS);
70         } catch (TimeoutException e) {
71             future.cancel(true);
72         } catch (InterruptedException e) {
73         } catch (ExecutionException e) {
74         }
75
76         return this.bestIndividual.getPhenotype();
77     }
78
79     private void applyGeneticAlgorythm() {
80         Population parents = null, offspring = null;
81         double deltaFitness = 1e6, nextAverageFitness = 1e6, averageFitness = 0;
82
83         Decoder decoder = new Decoder(new AndroidDecoder());
84         Evaluator evaluator = new Evaluator(new AreaEvaluator(this.screenHeight, this.screenWidth));
85         Creator creator = new Creator();
86         Matcher matcher = new Matcher();
87         Population population = creator.create(POPULATIONSIZE, this.so);
88         decoder.decode(population, this.context);
89         while (!Thread.interrupted() && deltaFitness > DELTAFITNESSTHRESHOLD) {
90             nextAverageFitness = evaluator.evaluate(population);
91             deltaFitness = nextAverageFitness - averageFitness;
92             averageFitness = nextAverageFitness;
93             parents = matcher.selectParents(population, PARENTSSIZE);
94             offspring = matcher.match(parents, OFFSPRINGSIZE, CROSSINGOVERRATE, MUTATIONRATE);
95             decoder.decode(offspring, this.context);
96             evaluator.evaluate(offspring);
97             population = generateNewPopulation(population, offspring);
98             this.bestIndividual = population.getBestIndividual();
99             saveToArchive(population);
100        }
101        Log.i("best deltafitness", " " + this.bestIndividual.getFitness());
102    }
103
104    private Population generateNewPopulation(Population population, Population offspring) {
105        removeLessEffectiveIndividuals(population, offspring);
106
107        return population;
108    }
109
110    private void removeLessEffectiveIndividuals(Population population, Population offspring) {
111        population.addPopulation(offspring);
112        List<Individual> individuals = population.getIndividualsOrderedByWorstFitnesses();
113        for (int i = offspring.size() - 1; i >= 0; i--) {
114            population.removeIndividual(individuals.get(i));
115        }
116    }
117
118    private void saveToArchive(Population population) {
119        NavigableMap<Double, Genotype> map = null;
120        if (this.screenWidth > this.screenHeight) {
121            map = so.getLandscapeGenotypes();
122        } else {
123            map = so.getPortaitGenotypes();
124        }
125        for (Individual individual : population.getIndividuals()) {
126            map.put(individual.getFitness(), individual.getGenotype());
127        }

```

GUIGeneratingMain.java

```
128     List<Double> keys = new ArrayList<Double>();
129     keys.addAll(map.keySet());
130     for (Double key : keys) {
131         if (map.size() > ARCHIEVESIZE) {
132             map.remove(key);
133         } else {
134             break;
135         }
136     }
137 }
138
139 }
140
```

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.codon;
5
6 import java.io.Serializable;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class BaseCodon implements Serializable {
12
13     protected String text;
14     protected float weight;
15     protected boolean isMutable = false;
16     protected int width;
17     protected int height;
18     protected int minValue;
19     protected int maxValue;
20
21     public BaseCodon() {
22
23     }
24
25     public BaseCodon(String text, int minValue, int maxValue) {
26         this.text = text;
27         this.minValue = minValue;
28         this.maxValue = maxValue;
29     }
30
31     public BaseCodon clone() {
32         BaseCodon codon = new BaseCodon();
33         codon.setText(text);
34         codon.setWeight(weight);
35         codon.setMutable(isMutable);
36         codon.setWidth(width);
37         codon.setHeight(height);
38         codon.setMinValue(minValue);
39         codon.setMaxValue(maxValue);
40
41         return codon;
42     }
43
44     public int getMinValue() {
45         return minValue;
46     }
47
48     public void setMinValue(int minValue) {
49         this.minValue = minValue;
50     }
51
52     public int getMaxValue() {
53         return maxValue;
54     }
55
56     public void setMaxValue(int maxValue) {
57         this.maxValue = maxValue;
58     }
59
60     public int getWidth() {
61         return width;
62     }
63
64     public void setWidth(int width) {
65         this.width = width;
66     }
67
68     public int getHeight() {
69         return height;
70     }
71
72     public void setHeight(int height) {
73         this.height = height;
74     }
75
76     public boolean isMutable() {
77         return isMutable;
78     }
79
80     public void setMutable(boolean isMutable) {
81         this.isMutable = isMutable;
82     }
83
84     public String getText() {
85         return text;
86     }
87
88     public void setText(String text) {
89         this.text = text;
90     }
91
92     public float getWeight() {
93         return weight;
94     }
95
96     public void setWeight(float weight) {
97         this.weight = weight;
98     }
99
100 }
```

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.codon;
5
6 import java.util.Arrays;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 public class ComboCodon extends BaseCodon {
13
14     private List<String> options;
15
16     public ComboCodon() {
17
18     }
19
20     public ComboCodon(String text, String options) {
21         super(text, 0, 0);
22         this.options = Arrays.asList(options.split("\\|"));
23     }
24
25     public ComboCodon clone() {
26         ComboCodon codon = new ComboCodon();
27         codon.setText(text);
28         codon.setWeight(weight);
29         codon.setMutable(isMutable);
30         codon.setWidth(width);
31         codon.setHeight(height);
32         codon.setMinValue(minValue);
33         codon.setMaxValue(maxValue);
34         codon.setOptions(options);
35
36         return codon;
37     }
38
39     public List<String> getOptions() {
40         return options;
41     }
42
43     public void setOptions(List<String> options) {
44         this.options = options;
45     }
46
47 }
48
```

```
1 /**
2 package br.com.mobgui4so.model.guigenerating.codon;
3
4 /**
5 * @author Ercilio Nascimento
6 */
7 public class LayoutCodon extends BaseCodon {
8
9     private int orientation;
10    private boolean isFrame;
11    private boolean isGetLayout = false;
12
13    public LayoutCodon() {
14
15    }
16
17    public LayoutCodon(String text) {
18        super(text, 0, 0);
19        this.isFrame = false;
20    }
21
22    public LayoutCodon clone() {
23        LayoutCodon codon = new LayoutCodon();
24        codon.setText(text);
25        codon.setWeight(weight);
26        codon.setMutable(isMutable);
27        codon.setWidth(width);
28        codon.setHeight(height);
29        codon.setMinValue(minValue);
30        codon.setMaxValue(maxValue);
31        codon.setOrientation(orientation);
32        codon.setFrame(isFrame);
33
34        return codon;
35    }
36
37    public boolean isGetLayout() {
38        return isGetLayout;
39    }
40
41    public void setGetLayout(boolean isGetLayout) {
42        this.isGetLayout = isGetLayout;
43    }
44
45    public int getOrientation() {
46        return orientation;
47    }
48
49    public void setOrientation(int orientation) {
50        this.orientation = orientation;
51    }
52
53    public boolean isFrame() {
54        return isFrame;
55    }
56
57    public void setFrame(boolean isFrame) {
58        this.isFrame = isFrame;
59    }
60
61 }
62
63 }
```

```

Creator.java

1 /**
4 package br.com.mobgui4so.model.guigenerating.creator;
5
6 import java.util.List;
18
19 /**
20 * @author Ercilio Nascimento
21 */
22 public class Creator {
23
24     public Population create(int populationSize, SmartObject so) {
25         Population population = new Population();
26         for (int i = 0; i < populationSize; i++) {
27             population.addIndividual(create(so));
28         }
29
30         return population;
31     }
32
33     private Individual create(SmartObject so) {
34         Individual individual = new Individual();
35         Genotype genotype = new Genotype();
36         Gene root = genotype.createRoot();
37         int id = 0;
38
39         for (SmartObjectService service : so.getServices()) {
40             Gene frameLayout = new Gene(ParamType.LAYOUT, service.getServiceName());
41             frameLayout.setId(++id);
42             LayoutCodon codon = (LayoutCodon) frameLayout.get��();
43             codon.setFrame(true);
44             codon.setWeight(1f);
45             if (service.getIdServiceModbus().equals("03")) {
46                 codon.setLayout(true);
47             }
48             codon.setMutable(true);
49             root.addChild(frameLayout);
50             for (SOServiceParam param : service.getParams()) {
51                 Gene paramLayout = new Gene(ParamType.LAYOUT);
52                 paramLayout.setId(++id);
53                 paramLayout.setParamLayout(true);
54                 BaseCodon layoutCodon = paramLayout.get��();
55                 layoutCodon.setWeight(1f);
56                 frameLayout.addChild(paramLayout);
57                 Gene paramGene = new Gene(param.getType(), param.getName(), param.getMinValue(), param.getMaxValue(), param.getOptions());
58                 paramGene.setId(++id);
59                 BaseCodon paramCodon = paramGene.get��();
56                 paramCodon.setWeight(.5f);
57                 paramLayout.addChild(paramGene);
58             }
59         }
60         individual.setGenotype(genotype);
61
62         return individual;
63     }
64
65     private Individual createTestGenotype(SmartObject so) {
66         Individual ind = new Individual();
67         Genotype test = new Genotype();
68         Gene root = test.createRoot();
69         int id = 0;
70         SmartObjectService service = so.getServices().get(0);
71         List<SOServiceParam> params = service.getParams();
72
73         Gene layout = new Gene(ParamType.LAYOUT, service.getServiceName());
74         layout.setId(++id);
75         LayoutCodon codon = (LayoutCodon) layout.get��();
76         codon.setOrientation(1);
77         codon.setFrame(true);
78         root.addChild(layout);
79
80         Gene layout2 = new Gene(ParamType.LAYOUT, "");
81         layout2.setId(++id);
82         layout.addChild(layout2);
83
84         Gene param1 = new Gene(params.get(0).getType(), params.get(0).getName());
85         param1.setId(++id);
86         layout2.addChild(param1);
87
88         Gene layout3 = new Gene(ParamType.LAYOUT, "");
89         layout3.setId(++id);
90         layout.addChild(layout3);
91
92         Gene param2 = new Gene(params.get(1).getType(), params.get(1).getName());
93         param2.setId(++id);
94         layout3.addChild(param2);
95
96         Gene layout4 = new Gene(ParamType.LAYOUT, "Limpar");
97         layout4.setId(++id);
98         LayoutCodon codon2 = (LayoutCodon) layout4.get��();
99         codon2.setOrientation(1);
100        codon2.setFrame(true);
101        root.addChild(layout4);
102
103        Gene layout5 = new Gene(ParamType.LAYOUT, "");
104        layout5.setId(++id);
105        layout4.addChild(layout5);
106
107        Gene param3 = new Gene(ParamType.DOUBLE, "Temperatura");
108        param3.setId(++id);
109        layout5.addChild(param3);
110
111        Gene layout6 = new Gene(ParamType.LAYOUT, "Explodir");
112        layout6.setId(++id);
113        layout5.addChild(layout6);
114
115        Gene layout7 = new Gene(ParamType.LAYOUT, "");
116        layout7.setId(++id);
117        layout6.addChild(layout7);

```

Creator.java

```
118 LayoutCodon codon3 = (LayoutCodon) layout6.getCodon();
119 codon3.setOrientation(1);
120 codon3.setFrame(true);
121 root.addChild(layout6);
122
123 Gene layout7 = new Gene(ParamType.LAYOUT, "");
124 layout7.setId(++id);
125 layout6.addChild(layout7);
126
127 Gene param4 = new Gene(ParamType.BOOLEAN, "Emoï¿½ï¿½o");
128 param4.setId(++id);
129 layout7.addChild(param4);
130
131 ind.setGenotype(test);
132
133 return ind;
134 }
135 }
136 }
```

Decoder.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.decoder;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
10
11 /**
12 * @author Ercilio Nascimento
13 */
14 public class Decoder {
15
16     private IDecodable decoderImpl;
17
18     public Decoder(IDecodable decoder) {
19         this.decoderImpl = decoder;
20     }
21
22     public void decode(Population population, Object context) {
23         for (Individual individual : population.getIndividuals()) {
24             decode(individual, context);
25         }
26     }
27
28     private void decode(Individual individual, Object context) {
29         Genotype genotype = individual.getGenotype();
30         BasePhenotype phenotype = decoderImpl.decode(genotype, context);
31         individual.setPhenotype(phenotype);
32     }
33 }
34
```

IDecodable.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.decoder;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 public interface IDecodable {
13
14     public BasePhenotype decode(Genotype gen, Object context);
15
16 }
17
```

AreaEvaluator.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.evaluator;
5
6 import br.com.mobgui4so.model.guigenerating.phenotype.BasePhenotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class AreaEvaluator implements IEvaluatable {
12
13     private int screenHeight, screenWidth;
14
15     public AreaEvaluator(int screenHeight, int screenWidth) {
16         super();
17         this.screenHeight = screenHeight;
18         this.screenWidth = screenWidth;
19     }
20
21     /* (non-Javadoc)
22      * @see br.com.mobgui4so.model.guigenerating.evaluator.IEvaluatable#evaluate(br.com.mobgui4so.model.guigenerating.phenotype.BasePhenotype)
23      */
24     @Override
25     public double evaluate(BasePhenotype phenotype) {
26         double areaBoundingFrames = (phenotype.getMaxFrameHeight() * phenotype.getMaxFrameWidth());
27         double blankArea = (areaBoundingFrames - phenotype.getFramesArea());
28         double value = 0.0;
29         int screenArea = this.screenHeight * this.screenWidth;
30         if (!phenotype.isBadPhenotype() && (blankArea < 0.20 * areaBoundingFrames)) {
31             value = phenotype.getMaxFrameHeight();
32             double area = phenotype.getFramesArea();
33             value = area < screenArea ? area / screenArea : screenArea / area;
34         }
35     }
36
37     return value;
38 }
39
40 }
41
```

Evaluator.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.evaluator;
5
6 import br.com.mobgui4so.model.guigenerating.pojo.Individual;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 public class Evaluator {
13
14     private IEvaluatable evaluatorImpl;
15
16     public Evaluator(IEvaluatable evaluator) {
17         this.evaluatorImpl = evaluator;
18     }
19
20     public double evaluate(Population population) {
21         double sumFitness = 0.0;
22         for (Individual individual : population.getIndividuals()) {
23             sumFitness += evaluate(individual);
24         }
25         double average = sumFitness / population.size();
26         population.setFitnessAverage(average);
27
28         return average;
29     }
30
31     private double evaluate(Individual ind) {
32         double fitness = this.evaluatorImpl.evaluate(ind.getPhenotype());
33         ind.setFitness(fitness);
34         return fitness;
35     }
36 }
37
```

IEvaluatable.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.evaluator;
5
6 import br.com.mobgui4so.model.guigenerating.phenotype.BasePhenotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public interface IEvaluatable {
12     public double evaluate(BasePhenotype phenotype);
13 }
14
15 }
16
```

```

2 * SelectParents(population)
4 package br.com.mobgui4so.model.guigenerating.matcher;
5
6 import java.util.HashMap;
16
17 /**
18 * @author Ercilio Nascimento
19 */
20 public class Matcher {
21
22     public Population selectParents(Population toSelect, int parentSize) {
23         Map<Individual, Double> mapIndividualProb = new HashMap<Individual, Double>();
24         Population parents = new Population();
25         double reproductionProbability = 0.0;
26         int i = 0;
27
28         Population population = Population.clone(toSelect);
29         double sumFitness = population.getFitnessesSum();
30         for (Individual individual : population.getIndividuals()) {
31             reproductionProbability = individual.getFitness() / sumFitness;
32             mapIndividualProb.put(individual, reproductionProbability);
33         }
34         while (parents.size() < parentSize) {
35             Individual individual = population.getIndividual(i);
36             reproductionProbability = mapIndividualProb.get(individual);
37             if (Math.random() < reproductionProbability) {
38                 parents.addIndividual(individual);
39                 population.removeIndividual(individual);
40             }
41             if (++i >= population.size()) {
42                 i = 0;
43             }
44         }
45
46         return parents;
47     }
48
49     public Population match(Population parents, int offspringSize, double crossoverRate, double mutationRate) {
50         Population offspring = new Population();
51         Crossover crossover = new Crossover(new BasicCrossover(), crossoverRate);
52         Mutation mutation = new Mutation(new BasicMutation(), mutationRate);
53         for (int i = 0; i < offspringSize; i++) {
54             int index = (int) Math.random() * parents.size();
55             Individual father = parents.getIndividual(index);
56             parents.removeIndividual(father);
57             index = (int) Math.random() * parents.size();
58             Individual mother = parents.getIndividual(index);
59             parents.addIndividual(father);
60             Genotype fatherGen = father.getGenotype();
61             Genotype motherGen = mother.getGenotype();
62             Genotype sonGen = crossover.crossover(fatherGen, motherGen);
63             mutation.mutate(sonGen);
64             Individual son = new Individual();
65             son.setGenotype(sonGen);
66             offspring.addIndividual(son);
67         }
68
69         return offspring;
70     }
71 }
72

```

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import java.util.ArrayList;
13
14 /**
15 * @author Ercilio Nascimento
16 */
17 public class BasicCrossover implements ICrossovable {
18
19     private Gene toErase;
20     private List<Integer> paramLayoutsRead = new ArrayList<Integer>();
21
22     @Override
23     public Genotype crossover(Genotype father, Genotype mother, double crossingoverRate) {
24         Genotype son = father.clone();
25         if (Math.random() <= crossingoverRate) {
26             Genotype fatherClone = father.clone();
27             Genotype motherClone = mother.clone();
28             ChildrenList sonChildren = new ChildrenList();
29             sonChildren.addAll(fatherClone.getRoot().getChildren());
30             Random r = new Random();
31             for (int i = 0; i < motherClone.getRoot().getNumberOfChildren(); i++) {
32                 int index = r.nextInt(sonChildren.size() + 1);
33                 sonChildren.add(index, motherClone.getRoot().getChild(i));
34             }
35             son = new Genotype();
36             son.createRoot();
37             son.getRoot().setChildren(sonChildren);
38             preOrderTree(son.getRoot());
39             paramLayoutsRead.clear();
40         }
41
42         return son;
43     }
44
45     private void preOrderTree(Gene root) {
46         if (root.isParamLayout()) {
47             if (paramLayoutsRead.contains(root.getId())) {
48                 toErase = root;
49             } else {
50                 paramLayoutsRead.add(root.getId());
51             }
52         } else {
53             for (int i = 0; i < root.getNumberOfChildren(); i++) {
54                 preOrderTree(root.getChild(i));
55                 if (toErase != null) {
56                     root.removeChild(toErase);
57                     i--;
58                     toErase = null;
59                 }
60             }
61             if (root.getNumberOfChildren() == 0) {
62                 toErase = root;
63             }
64         }
65     }
66
67     return;
68 }
69 }
70 }
```

BasicMutation.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import java.util.Random;
13
14 /**
15 * @author Ercilio Nascimento
16 */
17 public class BasicMutation implements IMutable {
18
19     @Override
20     public void mutate(Genotype son, double mutationRate) {
21         preOrderTree(son.getRoot(), mutationRate);
22
23     }
24
25     private void preOrderTree(Gene root, double mutationRate) {
26         BaseCodon codon = root.get��();
27         if (��.isMutable() && root.getNumberOfChildren() > 1) {
28             if (Math.random() <= mutationRate) {
29                 raffleGenesToMutate(root);
30             }
31             root.setLastChildRead(-1);
32             while (root.getLastChildRead() != root.getChildren().size() - 1) {
33                 preOrderTree(root.getNextChild(), mutationRate);
34             }
35         }
36
37         return;
38     }
39
40     private void raffleGenesToMutate(Gene root) {
41         Random r = new Random();
42         int numberofChildren = r.nextInt(root.getNumberOfChildren() - 1) + 2;
43         Gene toMutate = new Gene();
44         for (int i = 0; i < numberofChildren; i++) {
45             int sorteio = r.nextInt(root.getNumberOfChildren());
46             toMutate.addChild(root.getChild(sorteio));
47             root.removeChild(root.getChild(sorteio));
48         }
49         int numberofLines = r.nextInt(toMutate.getNumberOfChildren() - 1) + 1;
50         mutate(numberofLines, toMutate, root);
51     }
52
53
54     private void mutate(int numberofLines, Gene toMutate, Gene root) {
55         int size = toMutate.getNumberOfChildren();
56         Gene group = new Gene(ParamType.LAYOUT, "");
57         LayoutCodon group�� = (LayoutCodon) group.get��();
58         group��.setWeight(1f);
59         group.setId(-1);
60         for (int i = 0; i < size;) {
61             Gene gene = new Gene(ParamType.LAYOUT, "");
62             gene.setId(-1);
63             LayoutCodon�� codon = (LayoutCodon) gene.get��();
64             codon.setOrientation(1);
65             codon.setWeight((float) (1 / (Math.ceil((double) size / numberofLines))));
66             for (int j = 0; j < numberofLines; j++) {
67                 if (i < size) {
68                     gene.addChild(toMutate.getChild(0));
69                     toMutate.removeChild(0);
70                     i++;
71                 }
72             }
73             group.addChild(gene);
74         }
75         root.addChild(group);
76     }
77
78 }
79 }
```

Crossover.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class Crossover {
12
13     private ICrossovable crossoverImpl;
14     private double crossingoverRate;
15
16     public Crossover(ICrossovable crossover, double crossingoverRate) {
17         this.crossoverImpl = crossover;
18         this.crossingoverRate = crossingoverRate;
19     }
20
21     public Genotype crossover(Genotype father, Genotype mother) {
22         return this.crossoverImpl.crossover(father, mother, crossingoverRate);
23     }
24
25 }
26
```

ICrossoverable.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public interface ICrossoverable {
12     public Genotype crossover(Genotype father, Genotype mother, double crossingoverRate);
13 }
14
15 }
16
```

IMutable.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public interface IMutable {
12     public void mutate(Genotype son, double mutationRate);
13 }
14
15 }
16
```

Mutation.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.operator;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class Mutation {
12
13     private IMutable mutationImpl;
14     private double mutationRate;
15
16     public Mutation(IMutable mutation, double mutationRate) {
17         this.mutationImpl = mutation;
18         this.mutationRate = mutationRate;
19     }
20
21     public void mutate(Genotype son) {
22         this.mutationImpl.mutate(son, this.mutationRate);
23     }
24
25 }
26
```

AndroidPhenotype.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.phenotype;
5
6 import android.widget ScrollView;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class AndroidPhenotype extends BasePhenotype {
12
13     private String text;
14     private ScrollView layout;
15
16     public ScrollView getLayout() {
17         return layout;
18     }
19
20     public void setLayout(ScrollView layout) {
21         this.layout = layout;
22     }
23
24     public String getText() {
25         return text;
26     }
27
28     public void setText(String text) {
29         this.text = text;
30     }
31
32 }
33
```

BasePhenotype.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.phenotype;
5
6 /**
7 * @author Ercilio Nascimento
8 */
9 public class BasePhenotype {
10
11     protected double fitness;
12     private int framesArea;
13     private int maxFrameHeight;
14     private int maxFrameWidth;
15     private boolean badPhenotype = false;
16
17     public boolean isBadPhenotype() {
18         return badPhenotype;
19     }
20
21     public void setBadPhenotype(boolean badPhenotype) {
22         this.badPhenotype = badPhenotype;
23     }
24
25     public double getFitness() {
26         return fitness;
27     }
28
29     public void setFitness(double fitness) {
30         this.fitness = fitness;
31     }
32
33     public int getFramesArea() {
34         return framesArea;
35     }
36
37     public void setFramesArea(int framesArea) {
38         this.framesArea = framesArea;
39     }
40
41     public int getMaxFrameHeight() {
42         return maxFrameHeight;
43     }
44
45     public void setMaxFrameHeight(int maxFrameHeight) {
46         this.maxFrameHeight = maxFrameHeight;
47     }
48
49     public int getMaxFrameWidth() {
50         return maxFrameWidth;
51     }
52
53     public void setMaxFrameWidth(int maxFrameWidth) {
54         this.maxFrameWidth = maxFrameWidth;
55     }
56
57 }
```

Individual.java

```
1 /**
4 package br.com.mobgui4so.model.guigenerating.pojo;
5
6 import br.com.mobgui4so.model.guigenerating.Genotype;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 public class Individual {
13     private Genotype genotype;
14     private BasePhenotype phenotype;
15     private double fitness;
16
17     public Genotype getGenotype() {
18         return genotype;
19     }
20
21     public void setGenotype(Genotype genotype) {
22         this.genotype = genotype;
23     }
24
25     public BasePhenotype getPhenotype() {
26         return phenotype;
27     }
28
29     public void setPhenotype(BasePhenotype phenotype) {
30         this.phenotype = phenotype;
31     }
32
33     public double getFitness() {
34         return fitness;
35     }
36
37     public void setFitness(double fitness) {
38         this.fitness = fitness;
39     }
40
41 }
42
```

```

1 /**
4 package br.com.mobgui4so.model.guigenerating.pojo;
5
6 import java.util.ArrayList;
12
13 /**
14 * @author Ercilio Nascimento
15 */
16 public class Population {
17     private List<Individual> individuals = new ArrayList<Individual>();
18     private Double fitnessAverage; // média dos fit dos individuos
19
20     public int size() {
21         return this.individuals.size();
22     }
23
24     public List<Individual> getIndividualsOrderedByWorstFitnesses() {
25         List<Individual> individuals = new ArrayList<Individual>();
26         individuals.addAll(this.individuals);
27         Collections.sort(individuals, new Comparator<Individual>() {
28
29             @Override
30             public int compare(Individual ind1, Individual ind2) {
31                 return ind1.getFitness() < ind2.getFitness() ? -1 : (ind1.getFitness() > ind2.getFitness() ? +1 : 0);
32             }
33         });
34
35         return individuals;
36     }
37
38     public List<Individual> getIndividualsOrderedByBestFitnesses() {
39         List<Individual> individuals = new ArrayList<Individual>();
40         individuals.addAll(this.individuals);
41         Collections.sort(individuals, new Comparator<Individual>() {
42
43             @Override
44             public int compare(Individual ind1, Individual ind2) {
45                 return ind1.getFitness() < ind2.getFitness() ? +1 : (ind1.getFitness() > ind2.getFitness() ? -1 : 0);
46             }
47         });
48
49         return individuals;
50     }
51
52     public void setIndividuals(List<Individual> individuals) {
53         this.individuals = individuals;
54     }
55
56     public double getFitnessesSum() {
57         double sumFitness = 0.0;
58         for (Individual individual : this.individuals) {
59             sumFitness += individual.getFitness();
60         }
61
62         return sumFitness;
63     }
64
65     public Double getFitnessAverage() {
66         return fitnessAverage;
67     }
68
69     public void setFitnessAverage(Double fitnessAverage) {
70         this.fitnessAverage = fitnessAverage;
71     }
72
73     public void removeIndividual(int index) {
74         this.individuals.remove(index);
75     }
76
77     public void removeIndividual(Individual individual) {
78         this.individuals.remove(individual);
79     }
80
81     public void addPopulation(Population population) {
82         this.individuals.addAll(population.getIndividuals());
83     }
84
85     public List<Individual> getIndividuals() {
86         return this.individuals;
87     }
88
89     public void addIndividual(Individual individual) {
90         this.individuals.add(individual);
91     }
92
93     public Individual getIndividual(int index) {
94         return this.individuals.get(index);
95     }
96
97     public static Population clone(Population toClone) {
98         Population population = new Population();
99         population.setFitnessAverage(toClone.getFitnessAverage());
100        population.addPopulation(toClone);
101
102        return population;
103    }
104
105    public Individual getBestIndividual() {
106        List<Individual> individuals = getIndividualsOrderedByWorstFitnesses();
107        return individuals.get(individuals.size() - 1);
108    }
109
110    public BasePhenotype getBestPhenotype() {
111        return getBestIndividual().getPhenotype();
112    }

```

Population.java

```
112  
113      }  
114 }  
115
```

SmartObjectCommandSender.java

```
1 /**
4 package br.com.mobgui4so.model.interaction;
5
6 import java.util.List;
11
12 /**
13 * @author Ercilio Nascimento
14 */
15 public class SmartObjectCommandSender {
16
17     public static String sendSmartObjectCommand(String url, List<String> params) {
18         String[] list = new String[params.size()];
19         for (int i = 0; i < params.size(); i++) {
20             list[i] = params.get(i);
21         }
22
23         return StringEscapeUtils.unescapeJava(HttpHelper.doPost(url, list));
24     }
25
26 }
27
```

SmartObjectListRequester.java

```
1 /**
4 package br.com.mobgui4so.model.interaction;
5
6 import java.io.IOException;
18
19 /**
20 * @author Ercilio Nascimento
21 */
22 public class SmartObjectListRequester {
23
24     public static SmartObjectList getSmartObjectList(String[] form) throws IOException {
25         String list = HttpHelper.doPost(form[2] + "/getList.do", "user", form[0], "password", form[1]);
26         /*String list = "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,05,00,Ligar,Ligado,boolean,null,null,null;" +
27         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,16,00,Configurar,Temperatura,double,18,28,null;" +
28         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,16,00,Configurar,Humidade,double,0,100,null;" +
29         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,16,00,Configurar,Velocidade,combo,null,null,Fraco|M\u00E9dio|Forte;" +
30         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,16,00,Configurar,Timer,double,0,7,null;" +
31         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,3,,Tipo de
Fun\u00E7\u00E3o,Fun\u00E7\u00E3o,combo,null,null,Circular|Refrigerar|Aquecer;" +
32         "1,A1,http://192.168.43.138:8080/SOServer/soCommand.do,Ar Condicionado,03,00,Pegar Temperatura,Temperatura,get,null,null,null;" +
33         "3,A2,http://192.168.43.138:8080/SOServer/soCommand.do,L\u00E1mpada,05,00,Acender,Aceso,boolean,null,null,null;" +
34         "5,A3,http://192.168.43.138:8080/SOServer/soCommand.do,C02,03,00,Quantidade de CO2,C02,get,null,null,null;*/"
35         if (list.equals(""))
36             return null;
37         else
38             return parseList(StringEscapeUtils.unescapeJava(list));
39     }
40
41     private static SmartObjectList parseList(String list) {
42         SmartObjectList soList = new SmartObjectList();
43         if (list.startsWith("0"))
44             soList.setErrorMsg(list.substring(1));
45
46         return soList;
47     }
48     SmartObject so = null;
49     String soName = null;
50     String serviceName = null;
51     SmartObjectService soService = null;
52     List<String> sos = Arrays.asList(list.split(";"));
53     for (int i = 0; i < sos.size(); i++) {
54         List<String> services = Arrays.asList(sos.get(i).split(","));
55         if (!services.get(3).equalsIgnoreCase(soName)) {
56             if (soName != null) {
57                 so.add(soService);
58                 soList.add(so);
59             }
60             so = new SmartObject();
61             so.setIdSOBD(Integer.parseInt(services.get(0)));
62             so.setIdSOModbus(services.get(1));
63             so.setUrl(services.get(2));
64             so.setName(services.get(3));
65             soName = services.get(3);
66             soService = null;
67         }
68         if (serviceName == null || !services.get(6).equalsIgnoreCase(serviceName)) {
69             if (soService != null)
70                 so.add(soService);
71             soService = new SmartObjectService();
72             soService.setIdServiceModbus(services.get(4));
73             soService.setIdRegisterModbus(services.get(5));
74             soService.setServiceName(services.get(6));
75             serviceName = services.get(6);
76         }
77         SOServiceParam param = new SOServiceParam();
78         param.setName(services.get(7));
79         param.setType(ParamType.valueOf(services.get(8).toUpperCase()));
80         param.setMinValue(services.get(9));
81         param.setMaxValue(services.get(10));
82         param.setOptions(services.get(11));
83         soService.addParam(param);
84     }
85     so.add(soService);
86     soList.add(so);
87
88     return soList;
89 }
90 }
```

```

1 /**
4 package br.com.mobgui4so.model.pojo;
5
6 import java.io.Serializable;
13
14 /**
15 * @author Ercilio Nascimento
16 */
17 public class SmartObject implements Serializable {
18
19     /**
20      *
21      */
22     private static final long serialVersionUID = -451798137279786633L;
23     private int idSOBD;
24     private String idSOModbus;
25     private String url;
26     private String name;
27     private List<SmartObjectService> services;
28     private boolean available = true;
29     private NavigableMap<Double, Genotype> landscapeGenotypes;
30     private NavigableMap<Double, Genotype> portraitGenotypes;
31     private Double actualLandscapeKey;
32     private Double actualPortraitKey;
33
34     public Double getActualLandscapeKey() {
35         return actualLandscapeKey;
36     }
37
38     public String getIdServiceModbus(String serviceName) {
39         String idServiceModbus = null;
40         for (SmartObjectService service : this.services) {
41             if (service.getServiceName().equalsIgnoreCase(serviceName)) {
42                 idServiceModbus = service.getIdServiceModbus();
43                 break;
44             }
45         }
46
47         return idServiceModbus;
48     }
49
50     public String getIdRegisterModbus(String serviceName) {
51         String idRegisterModbus = null;
52         for (SmartObjectService service : this.services) {
53             if (service.getServiceName().equalsIgnoreCase(serviceName)) {
54                 idRegisterModbus = service.getIdRegisterModbus();
55                 break;
56             }
57         }
58
59         return idRegisterModbus;
60     }
61
62     public String getIdSOModbus() {
63         return idSOModbus;
64     }
65
66     public void setIdSOModbus(String idSOModbus) {
67         this.idSOModbus = idSOModbus;
68     }
69
70     public void setActualLandscapeKey(Double actualLandscapeKey) {
71         this.actualLandscapeKey = actualLandscapeKey;
72     }
73
74     public Double getActualPortraitKey() {
75         return actualPortraitKey;
76     }
77
78     public void setActualPortraitKey(Double actualPortraitKey) {
79         this.actualPortraitKey = actualPortraitKey;
80     }
81
82     public SmartObject() {
83         this.services = new ArrayList<SmartObjectService>();
84         this.landscapeGenotypes = new TreeMap<Double, Genotype>();
85         this.portraitGenotypes = new TreeMap<Double, Genotype>();
86     }
87
88     public NavigableMap<Double, Genotype> getLandscapeGenotypes() {
89         return landscapeGenotypes;
90     }
91
92     public void setLandscapeGenotypes(NavigableMap<Double, Genotype> landscapeGenotypes) {
93         this.landscapeGenotypes = landscapeGenotypes;
94     }
95
96     public NavigableMap<Double, Genotype> getPortraitGenotypes() {
97         return portraitGenotypes;
98     }
99
100    public void setPortraitGenotypes(NavigableMap<Double, Genotype> portraitGenotypes) {
101        this.portraitGenotypes = portraitGenotypes;
102    }
103
104    public boolean isAvailable() {
105        return available;
106    }
107
108    public void setAvailable(boolean available) {
109        this.available = available;
110    }
111
112    public void add(SmartObjectService service) {

```

```
SmartObject.java

113     this.services.add(service);
114 }
115
116 public SmartObject(int idSOBD, String idSOModbus, String name, List<SmartObjectService> services) {
117     this.idSOBD = idSOBD;
118     this.idSOModbus = idSOModbus;
119     this.name = name;
120     this.services = services;
121 }
122
123 public String getUrl() {
124     return url;
125 }
126
127 public void setUrl(String url) {
128     this.url = url;
129 }
130
131 public String getName() {
132     return this.name;
133 }
134
135 public void setName(String name) {
136     this.name = name;
137 }
138
139 public int getIdSOBD() {
140     return idSOBD;
141 }
142
143 public void setIdSOBD(int idSOBD) {
144     this.idSOBD = idSOBD;
145 }
146
147 public List<SmartObjectService> getServices() {
148     return this.services;
149 }
150
151 public void setServices(List<SmartObjectService> services) {
152     this.services = services;
153 }
154 }
155
```

```

1 /**
4 package br.com.mobgui4so.model.pojo;
5
6 import java.io.Serializable;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public class SmartObjectList implements Serializable {
14
15     private static final long serialVersionUID = 6494791003706930546L;
16     private List<SmartObject> list;
17     private String user;
18     private String password;
19     private boolean ckSaveUser;
20     private String serverURL;
21     private String errorMsg;
22
23     public SmartObjectList merge(SmartObjectList listFromDisk) {
24         if (listFromDisk != null) {
25             listFromDisk.setAllUnavailable();
26             for (SmartObject so : list) {
27                 if (!listFromDisk.containsSO(so.getIdSOBD())) {
28                     listFromDisk.add(so);
29                 } else {
30                     listFromDisk.getSO(so.getIdSOBD()).setAvailable(true);
31                 }
32             }
33
34             return listFromDisk;
35         } else {
36             return this;
37         }
38     }
39
40     public SmartObject getSO(int idSOBD) {
41         SmartObject smart = null;
42         for (SmartObject so : list) {
43             if (so.getIdSOBD() == idSOBD) {
44                 smart = so;
45                 break;
46             }
47         }
48
49         return smart;
50     }
51
52     public String getErrorMsg() {
53         return errorMsg;
54     }
55
56     public void setAllUnavailable() {
57         for (SmartObject so : list) {
58             so.setAvailable(false);
59         }
60     }
61
62     public boolean containsSO(int idSOBD) {
63         boolean contains = false;
64         for (SmartObject so : list) {
65             if (so.getIdSOBD() == idSOBD) {
66                 contains = true;
67                 break;
68             }
69         }
70
71         return contains;
72     }
73
74     public void setErrorMsg(String errorMsg) {
75         this.errorMsg = errorMsg;
76     }
77
78     public SmartObjectList() {
79         this.list = new ArrayList<SmartObject>(0);
80     }
81
82     public String getUser() {
83         return user;
84     }
85
86     public void setUser(String user) {
87         this.user = user;
88     }
89
90     public String getPassword() {
91         return password;
92     }
93
94     public void setPassword(String password) {
95         this.password = password;
96     }
97
98     public boolean isCkSaveUser() {
99         return ckSaveUser;
100    }
101
102    public void setCkSaveUser(boolean ckSaveUser) {
103        this.ckSaveUser = ckSaveUser;
104    }
105
106    public String getServerURL() {
107        return serverURL;
108    }

```

SmartObjectList.java

```
109 public void setServerURL(String serverURL) {
110     this.serverURL = serverURL;
111 }
112 }
113 /**
114 * @return the list
115 */
116 public List<SmartObject> getList() {
117     return this.list;
118 }
119 }
120 }
121 /**
122 * @param list
123 *      the list to set
124 */
125 public void setList(List<SmartObject> list) {
126     this.list = list;
127 }
128 }
129 public String[] getSONames() {
130     String[] names = new String[list.size()];
131     for (int i = 0; i < names.length; i++) {
132         if (list.get(i).isAvailable()) {
133             names[i] = list.get(i).getName();
134         }
135     }
136     return names;
137 }
138 }
139 public int getMaxSONameLength() {
140     int length = 0;
141     String[] names = new String[list.size()];
142     for (int i = 0; i < names.length; i++) {
143         int currentLength = list.get(i).getName().length();
144         if (currentLength > length) {
145             length = currentLength;
146         }
147     }
148     return length;
149 }
150 }
151 }
152 public void add(SmartObject so) {
153     this.list.add(so);
154 }
155 }
156 }
157 }
```

SmartObjectService.java

```
1 /**
4 package br.com.mobgui4so.model.pojo;
5
6 import java.io.Serializable;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public class SmartObjectService implements Serializable {
14
15     private static final long serialVersionUID = 1334387233404595046L;
16     private String serviceName;
17     private String returnType;
18     private List<SOServiceParam> params;
19     private String idServiceModbus, idRegisterModbus;
20
21     public SmartObjectService() {
22         this.params = new ArrayList<SOServiceParam>();
23     }
24
25     public List<SOServiceParam> getParams() {
26         return params;
27     }
28
29     public void setParams(List<SOServiceParam> params) {
30         this.params = params;
31     }
32
33     public void addParam(SOServiceParam param) {
34         this.params.add(param);
35     }
36
37     public String getServiceName() {
38         return this.serviceName;
39     }
40
41     public void setServiceName(String serviceName) {
42         this.serviceName = serviceName;
43     }
44
45     public String getReturnType() {
46         return returnType;
47     }
48
49     public void setReturnType(String returnType) {
50         this.returnType = returnType;
51     }
52
53     public String getIdServiceModbus() {
54         return idServiceModbus;
55     }
56
57     public void setIdServiceModbus(String idServiceModbus) {
58         this.idServiceModbus = idServiceModbus;
59     }
60
61     public String getIdRegisterModbus() {
62         return idRegisterModbus;
63     }
64
65     public void setIdRegisterModbus(String idRegisterModbus) {
66         this.idRegisterModbus = idRegisterModbus;
67     }
68 }
69 }
70 }
```

SOServiceParam.java

```
1 /**
4 package br.com.mobgui4so.model.pojo;
5
6 import java.io.Serializable;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public class SOServiceParam implements Serializable {
14
15     private static final long serialVersionUID = 3457815774849515058L;
16     private String name;
17     private ParamType type;
18     private int minValue;
19     private int maxValue;
20     private String options;
21
22     public SOServiceParam() {
23
24     }
25
26     public SOServiceParam(String name, String type) {
27         this.name = name;
28         this.type = ParamType.valueOf(type.toUpperCase());
29     }
30
31     public String getName() {
32         return name;
33     }
34
35     public ParamType getType() {
36         return type;
37     }
38
39     public int getMinValue() {
40         return minValue;
41     }
42
43     public void setMinValue(String minValue) {
44         if (minValue != null && !minValue.equals("") && !minValue.equals("null")) {
45             this.minValue = Integer.parseInt(minValue);
46         }
47     }
48
49     public int getMaxValue() {
50         return maxValue;
51     }
52
53     public void setMaxValue(String maxValue) {
54         if (maxValue != null && !maxValue.equals("") && !maxValue.equals("null")) {
55             this.maxValue = Integer.parseInt(maxValue);
56         }
57     }
58
59     public String getOptions() {
60         return options;
61     }
62
63     public void setOptions(String options) {
64         this.options = options;
65     }
66
67     public void setName(String name) {
68         this.name = name;
69     }
70
71     public void setType(ParamType type) {
72         this.type = type;
73     }
74
75 }
76 }
```

```

1 /**
4 package br.com.mobgui4so.utils;
5
6 import android.app.AlertDialog;
18
19 /**
20 * @author Ercílio Gonçalves Nascimento
21 *
22 */
23 public class AndroidUtils {
24     protected static final String TAG = "BuscaCep";
25
26     public static boolean isNetworkAvailable(Context context) {
27         ConnectivityManager connectivity = (ConnectivityManager) context
28             .getSystemService(Context.CONNECTIVITY_SERVICE);
29         if (connectivity == null) {
30             return false;
31         } else {
32             NetworkInfo[] info = connectivity.getAllNetworkInfo();
33             if (info != null) {
34                 for (int i = 0; i < info.length; i++) {
35                     if (info[i].getState() == NetworkInfo.State.CONNECTED) {
36                         return true;
37                     }
38                 }
39             }
40         }
41         return false;
42     }
43
44     public static void alertDialog(final Context context, final int mensagem) {
45         try {
46             AlertDialog dialog = new AlertDialog.Builder(context)
47                 .setTitle(context.getString(R.string.app_name))
48                 .setMessage(mensagem).create();
49             dialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
50                 public void onClick(DialogInterface dialog, int which) {
51                     return;
52                 }
53             });
54             dialog.show();
55         } catch (Exception e) {
56             Log.e(TAG, e.getMessage(), e);
57         }
58     }
59
60     public static void alertDialogYesNo(final Context context,
61         final int mensagem) {
62         try {
63             AlertDialog.Builder dialog = new AlertDialog.Builder(context);
64             dialog.setTitle(context.getString(R.string.app_name));
65             dialog.setMessage(mensagem).create();
66             dialog.setNegativeButton("Não",
67                 new DialogInterface.OnClickListener() {
68                     public void onClick(DialogInterface dialog, int which) {
69                         return;
70                     }
71                 });
72             dialog.setPositiveButton("Sim",
73                 new DialogInterface.OnClickListener() {
74
75                 @Override
76                 public void onClick(DialogInterface dialog, int which) {
77                     WifiManager wifiManager = (WifiManager) context
78                         .getSystemService(Context.WIFI_SERVICE);
79                     wifiManager.setWifiEnabled(true);
80                 }
81             });
82             dialog.show();
83         } catch (Exception e) {
84             Log.e(TAG, e.getMessage(), e);
85         }
86     }
87
88     public static void alertDialog(final Context context, final String mensagem) {
89         try {
90             AlertDialog dialog = new AlertDialog.Builder(context).create();
91             dialog.setTitle(context.getString(R.string.app_name));
92             dialog.setMessage(mensagem);
93             dialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
94                 public void onClick(DialogInterface dialog, int which) {
95                     return;
96                 }
97             });
98             dialog.show();
99         } catch (Exception e) {
100             Log.e(TAG, e.getMessage(), e);
101         }
102     }
103
104     // Retorna se é Android 3.x "honeycomb" ou superior (API Level 11)
105     public static boolean isAndroid_3() {
106         int apiLevel = Build.VERSION.SDK_INT;
107         if (apiLevel >= 11) {
108             return true;
109         }
110         return false;
111     }
112
113     // Retorna se a tela é large ou xlarge
114     public static boolean isTablet(Context context) {
115         return (context.getResources().getConfiguration().screenLayout & Configuration.SCREENLAYOUT_SIZE_MASK) >= Configuration.SCREENLAYOUT_SIZE_LARGE;
116     }
117

```

AndroidUtils.java

```
118 // Retorna se é um tablet com Android 3.x
119 public static boolean isAndroid_3_Tablet(Context context) {
120     return isAndroid_3() && isTablet(context);
121 }
122
123 // Fecha o teclado virtual se aberto (view com foco)
124 public static boolean closeVirtualKeyboard(Context context, View view) {
125     // Fecha o teclado virtual
126     InputMethodManager imm = (InputMethodManager) context
127         .getSystemService(Context.INPUT_METHOD_SERVICE);
128     if (imm != null) {
129         boolean b = imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
130         return b;
131     }
132     return false;
133 }
134 }
135 }
```

BlockingOnUIRunnable.java

```
1 /**
4 package br.com.mobgui4so.utils;
5
6 import android.app.Activity;
7
8 /**
9 * @author Ercilio Nascimento
10 */
11 public class BlockingOnUIRunnable {
12
13     // Activity
14     private Activity activity;
15
16     // Event Listener
17     private IBlockingOnUIRunnableListener listener;
18
19     // UI runnable
20     private Runnable uiRunnable;
21
22     /**
23      * Class initialization
24      *
25      * @param activity
26      *          Activity
27      * @param listener
28      *          Event listener
29      */
30     public BlockingOnUIRunnable(Activity activity, IBlockingOnUIRunnableListener listener) {
31         this.activity = activity;
32         this.listener = listener;
33
34         uiRunnable = new Runnable() {
35             public void run() {
36                 // Execute custom code
37                 if (BlockingOnUIRunnable.this.listener != null) {
38                     BlockingOnUIRunnable.this.listener.runOnUiThread();
39                 }
40
41                 synchronized (this) {
42                     this.notify();
43                 }
44             }
45         };
46     }
47
48     /**
49      * Start runnable on UI thread and wait until finished
50      */
51     public void startOnUiAndWait() {
52         synchronized (uiRunnable) {
53             // Execute code on UI thread
54             activity.runOnUiThread(uiRunnable);
55
56             // Wait until runnable finished
57             try {
58                 uiRunnable.wait();
59             } catch (InterruptedException e) {
60                 e.printStackTrace();
61             }
62         }
63     }
64 }
65 }
```

```

1 /**
4 package br.com.mobgui4so.utils;
5
6 import java.io.BufferedReader;
15
16 /**
17 * @author Ercilio Nascimento
18 */
19 public class HttpHelper {
20
21     public static String doGet(String url, String charset) throws IOException {
22         URL u = new URL(url);
23         HttpURLConnection conn = (HttpURLConnection) u.openConnection();
24         conn.setRequestMethod("GET");
25         conn.setDoOutput(true);
26         conn.setDoInput(true);
27         conn.connect();
28         InputStream in = conn.getInputStream();
29         String s = IOUtils.toString(in, charset);
30         in.close();
31         conn.disconnect();
32         return s;
33     }
34
35     public static String doPost(String url, String... params) {
36         StringBuilder ack = new StringBuilder();
37         StringBuilder data = new StringBuilder();
38         try {
39             for (int i = 0; i < params.length; i++) {
40                 data.append(URLEncoder.encode(params[i], "UTF-8") + "=" + URLEncoder.encode(params[++i], "UTF-8") + "&");
41             }
42
43             URLConnection conn;
44             URL u = new URL(url);
45             conn = u.openConnection();
46             conn.setDoOutput(true);
47             conn.setUseCaches(false);
48             ((HttpURLConnection) conn).setRequestMethod("POST");
49             OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
50             wr.write(data.toString());
51             wr.flush();
52
53             // Get the response
54             BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
55             String line;
56             while ((line = rd.readLine()) != null) {
57                 ack.append(line);
58             }
59             wr.close();
60             rd.close();
61         } catch (IOException e) {
62             // TODO Auto-generated catch block
63             e.printStackTrace();
64         }
65
66         return ack.toString();
67     }
68 }
69 }
70

```

BaseActivity.java

```
1 /**
4 package br.com.mobgui4so.view;
5
6 import android.os.AsyncTask;
13
14 /**
15 * @author Ercilio Nascimento
16 */
17 public class BaseActivity extends SlidingFragmentActivity {
18
19     private TransactionTask task;
20
21     @Override
22     public void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.slidingmenu);
25         getSlidingMenu().setTouchModeAbove(SlidingMenu.TOUCHMODE_NONE);
26     }
27
28     protected void alert(int message) {
29         AndroidUtils.alertDialog(this, message);
30     }
31
32     public void startTransaction(ITransaction transaction) {
33         boolean networkOK = AndroidUtils.isNetworkAvailable(this);
34         if (networkOK) {
35             task = new TransactionTask(this, transaction);
36             task.execute();
37         } else {
38             AndroidUtils.alertDialog(this, R.string.error_connection_unavailable);
39         }
40     }
41
42     @Override
43     public void onDestroy() {
44         super.onDestroy();
45         this.finishTransaction();
46     }
47
48     public void finishTransaction() {
49         if (task != null) {
50             boolean executing = task.getStatus().equals(AsyncTask.Status.RUNNING);
51             if (executing) {
52                 task.cancel(true);
53             }
54         }
55     }
56
57 }
```

ITransaction.java

```
1 /**
4 package br.com.mobgui4so.view;
5
6 /**
7 * @author Ercilio Nascimento
8 */
9 public interface ITransaction {
10     public void preExecute();
11     public void execute() throws Exception;
12     public void postExecute();
13     public void swapLayouts();
14 }
15
16
17
18 }
19
```

ListSmartObjectActivity.java

```
1 package br.com.mobgui4so.view;
2
3 import java.io.FileNotFoundException;
4
5 /**
6  * @author Ercilio Nascimento
7 */
8
9
10 public class ListSmartObjectActivity extends BaseActivity implements OnItemClickListener {
11
12     private SmartObjectList listFromDisk;
13     private ListView listView;
14
15     @Override
16     public void onCreate(Bundle icicle) {
17         super.onCreate(icicle);
18         setContentView(R.layout.activity_list_smart_object);
19         try {
20             this.listFromDisk = new ApplicationFacade().loadSmartObjectList(openFileInput("SOFILE"));
21         } catch (FileNotFoundException e) {
22             // do nothing
23         } catch (IOException e) {
24             // do nothing
25         } catch (ClassNotFoundException e) {
26             e.printStackTrace();
27         }
28         // this.solist = (SmartObjectList) getIntent().getSerializableExtra("list");
29         this.listView = (ListView) findViewById(R.id.listViewSO);
30         ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, this.listFromDisk.getSONames());
31         this.listView.setAdapter(adapter);
32         this.listView.setOnItemClickListener(this);
33     }
34
35     @Override
36     public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
37         Intent i = new Intent(this, SmartObjectGUIActivity.class);
38         Bundle b = new Bundle();
39         b.putSerializable("SOLIST", this.listFromDisk);
40         b.putInt("idSOClicked", arg2);
41         i.putExtras(b);
42         startActivity(i);
43     }
44 }
45
46 }
```

MainActivity.java

```
1 /**
4 package br.com.mobgui4so.view;
5
6 import java.io.FileNotFoundException;
26
27 /**
28 * @author Ercilio Nascimento
29 */
30 public class MainActivity extends BaseActivity implements OnClickListener, OnFocusChangeListener, ITransaction {
31
32     private ImageButton btSearch;
33     private RadioGroup rgSearchType;
34     private SmartObjectList list;
35     private SmartObjectList listFromDisk;
36     private LinearLayout searchLayout;
37     private LinearLayout progressLayout;
38     private EditText etUser;
39     private EditText etPassword;
40     private EditText etServer;
41     private CheckBox ckSaveUser;
42     private ApplicationFacade facade;
43     private Handler mHandler;
44
45     @Override
46     public void onCreate(Bundle savedInstanceState) {
47         super.onCreate(savedInstanceState);
48         setContentView(R.layout.activity_main);
49         this.facade = new ApplicationFacade();
50         this.etUser = (EditText) findViewById(R.id.etUser);
51         etUser.setText("ercilio");
52         this.etPassword = (EditText) findViewById(R.id.etPassword);
53         etPassword.setText("batatinha");
54         this.etServer = (EditText) findViewById(R.id.etServer);
55         etServer.setText("http://172.23.21.31:8080/SORepository");
56         // this.ckSaveUser = (CheckBox) findViewById(R.id.ckSaveUser);
57         this.searchLayout = (LinearLayout) findViewById(R.id.searchLayout);
58         this.progressLayout = (LinearLayout) findViewById(R.id.progressLayout);
59         // this.rgSearchType = (RadioGroup) findViewById(R.id.radioGroup1);
60         this.btSearch = (ImageButton) findViewById(R.id.bSearch);
61         this.btSearch.setOnClickListener(this);
62         this.etUser.setOnFocusChangeListener(this);
63         this.etPassword.setOnFocusChangeListener(this);
64         this.etServer.setOnFocusChangeListener(this);
65         try {
66             this.listFromDisk = this.facade.loadSmartObjectListFromDisk(openFileInput("SOFILE"));
67         } catch (FileNotFoundException e) {
68             // do nothing
69         } catch (IOException e) {
70             // do nothing
71         } catch (ClassNotFoundException e) {
72             e.printStackTrace();
73         }
74
75         mHandler = new Handler() {
76             @Override
77             public void handleMessage(Message msg) {
78                 super.handleMessage(msg);
79                 switch (msg.what) {
80                     case 1:
81                         Intent i = new Intent(MainActivity.this, ListSmartObjectActivity.class);
82                         i.putExtra("list", listFromDisk);
83                         startActivity(i);
84                         break;
85                     default:
86                         break;
87                 }
88             }
89         };
90     }
91
92     @Override
93     public void preExecute() {
94         this.swapLayouts();
95     }
96
97     @Override
98     public void execute() throws IOException, ClassNotFoundException {
99         String[] form = new String[4];
100        form[0] = this.etUser.getText().toString();
101        form[1] = this.etPassword.getText().toString();
102        form[2] = this.etServer.getText().toString();
103        // form[3] = String.valueOf(this.ckSaveUser.isChecked());
104        this.list = this.facade.discovery(form, 1, getResources().getAssets().open("appConfig.properties"));
105    }
106
107    @Override
108    protected void onRestart() {
109        super.onRestart();
110        if (this.searchLayout.getVisibility() == LinearLayout.INVISIBLE)
111            this.swapLayouts();
112    }
113
114    @Override
115    public void postExecute() {
116        if (this.list == null) {
117            alert(R.string.noSofound);
118            this.swapLayouts();
119        } else if (this.list.getErrorMsg() != null) {
120            alert(R.string.invalidUser);
121            this.swapLayouts();
122        } else {
123            listFromDisk = list.merge(listFromDisk);
124            new Thread(new Runnable() {
125                public void run() {
```

```

>MainActivity.java

126
127     try {
128         facade.saveSmartObjectListToDisk(listFromDisk, openFileOutput("SOFILE", Context.MODE_PRIVATE));
129         mHandler.obtainMessage(1).sendToTarget();
130     } catch (FileNotFoundException e) {
131         e.printStackTrace();
132     }
133     }).start();
134 }
135
136
137 @Override
138 public void onClick(View v) {
139     startTransaction(this);
140 }
141
142 @Override
143 public void onFocusChange(View v, boolean hasFocus) {
144     boolean has = ((EditText) v).getText().toString().equals(getString(R.string.user)) || ((EditText)
v).getText().toString().equals(getString(R.string.password))
145     || ((EditText) v).getText().toString().equals(getString(R.string.server)) || ((EditText) v).getText().toString().equals("");
146     if (hasFocus && has) {
147         if (v.equals(this.etPassword)) {
148             this.etPassword.setInputType(InputType.TYPE_TEXT_VARIATION_PASSWORD);
149         }
150         ((EditText) v).setText("");
151     } else if (has) {
152         if (v.equals(this.etUser)) {
153             this.etUser.setText(R.string.user);
154         } else if (v.equals(this.etPassword)) {
155             this.etPassword.setInputType(InputType.TYPE_TEXT_VARIATION_NORMAL);
156             this.etPassword.setText(R.string.password);
157         } else {
158             this.etServer.setText(R.string.server);
159         }
160     }
161 }
162
163 @Override
164 public void onBackPressed() {
165     if (this.searchLayout.getVisibility() == LinearLayout.INVISIBLE) {
166         this.swapLayouts();
167         finishTransaction();
168     } else {
169         finish();
170     }
171 }
172
173 @Override
174 public void swapLayouts() {
175     this.searchLayout.setVisibility(this.searchLayout.getVisibility() == LinearLayout.VISIBLE ? LinearLayout.INVISIBLE : LinearLayout.VISIBLE);
176     this.progressLayout.setVisibility(this.progressLayout.getVisibility() == LinearLayout.VISIBLE ? LinearLayout.INVISIBLE : LinearLayout.VISIBLE);
177 }
178 }
179

```

```

NegativeSeekBar.java

1 /**
4 package br.com.mobgui4so.view;
5
6 import android.content.Context;
11
12 /**
13 * @author Ercilio Nascimento
14 */
15 public class NegativeSeekBar extends SeekBar {
16
17     private int minValue;
18     private int maxValue;
19     private TextView label;
20     private String labelText;
21     private float floatProgress;
22     private int intProgress;
23     private ParamType type;
24
25     public NegativeSeekBar(Context context, int minValue, int maxValue, TextView label, ParamType type) {
26         this(context);
27         this.minValue = minValue;
28         this.maxValue = maxValue;
29         this.label = label;
30         this.labelText = label.getText().toString();
31         this.type = type;
32         setIntervalAndListener();
33     }
34
35     public NegativeSeekBar(Context context) {
36         super(context);
37     }
38
39     public NegativeSeekBar(Context context, AttributeSet attrs) {
40         super(context, attrs);
41     }
42
43     public NegativeSeekBar(Context context, AttributeSet attrs, int defStyle) {
44         super(context, attrs, defStyle);
45     }
46
47     public String getStringProgress() {
48         String value = null;
49         if (this.type == ParamType.DOUBLE) {
50             value = String.valueOf(intProgress);
51         } else {
52             value = String.valueOf(intProgress);
53         }
54
55         return value;
56     }
57
58     private void setIntervalAndListener() {
59         int max = 0;
60         if (this.minValue < 0 && this.maxValue < 0) {
61             max = 10 * (Math.abs(this.minValue) - Math.abs(this.maxValue));
62             setOnSeekBarChangeListener(negativeLimitsListener());
63         } else if (this.minValue < 0 && this.maxValue > 0) {
64             max = this.maxValue + Math.abs(this.minValue);
65             setOnSeekBarChangeListener(negativeLimitsListener());
66         } else {
67             max = this.maxValue - this.minValue;
68             setOnSeekBarChangeListener(positiveLimitsListener());
69         }
70         setMax(max);
71     }
72
73     private OnSeekBarChangeListener negativeLimitsListener() {
74         OnSeekBarChangeListener listener = new OnSeekBarChangeListener() {
75
76             @Override
77             public void onStopTrackingTouch(SeekBar seekBar) {
78                 // TODO Auto-generated method stub
79
80             }
81
82             @Override
83             public void onStartTrackingTouch(SeekBar seekBar) {
84                 // TODO Auto-generated method stub
85
86             }
87
88             @Override
89             public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
90                 floatProgress = (float) (progress -= (10 * Math.abs(minValue))) / 10;
91                 label.setText(labelText + " : " + floatProgress);
92             };
93
94         }
95         return listener;
96     }
97
98     private OnSeekBarChangeListener positiveLimitsListener() {
99         OnSeekBarChangeListener listener = new OnSeekBarChangeListener() {
100
101             @Override
102             public void onStopTrackingTouch(SeekBar seekBar) {
103                 // TODO Auto-generated method stub
104
105             }
106
107             @Override
108             public void onStartTrackingTouch(SeekBar seekBar) {
109                 // TODO Auto-generated method stub
110

```

NegativeSeekBar.java

```
111     }
112
113     @Override
114     public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
115         progress += minValue;
116         intProgress = progress;
117         label.setText(labelText + " : " + progress);
118     }
119 }
120
121     return listener;
122 }
123
124 }
```

SmartObjectGUIActivity.java

```
1 package br.com.mobgui4so.view;
2
3 import java.io.FileNotFoundException;
43
44 public class SmartObjectGUIActivity extends BaseActivity implements OnItemClickListener, ITransaction {
45
46     private ListView listView;
47     private SmartObjectList soList;
48     private int idSOClicked;
49     private SmartObject so;
50     private ApplicationFacade facade;
51     private ScrollView soGUILayout;
52     private LinearLayout progressLayout;
53     private TextView tvProgressMsg;
54     private ScrollView layout;
55     private boolean flFirstTime;
56     private Menu menu;
57     private List<String> listParameters;
58     private boolean isSendCommand;
59     private String ack;
60     private static final String PREFS_NAME = "Preferences";
61     private AndroidDecoder decoder;
62     private Handler mHandler;
63     private Double landscapeKey;
64     private Double portraitKey;
65     private boolean isLandscape;
66     private int width;
67     private int height;
68     private Thread loop;
69     private TextView serviceName;
70
71     @Override
72     public void onCreate(Bundle savedInstanceState) {
73         super.onCreate(savedInstanceState);
74         setContentView(R.layout.activity_smart_object_gui);
75         Bundle b = getIntent().getExtras();
76         Sharedpreferences settings = getSharedPreferences(PREFS_NAME, 0);
77         this.idSOClicked = settings.getInt("idSOClicked", -1);
78         if (this.idSOClicked == -1) {
79             this.idSOClicked = b.getInt("idSOClicked", 0);
80         }
81         try {
82             this.soList = new ApplicationFacade().loadSmartObjectListFromDisk(openFileInput("SOFILE"));
83         } catch (FileNotFoundException e) {
84             // do nothing
85         } catch (IOException e) {
86             // do nothing
87         } catch (ClassNotFoundException e) {
88             e.printStackTrace();
89         }
90         // this.soList = (SmartObjectList) b.getSerializable("SOLIST");
91         this.so = soList.getList().get(idSOClicked);
92         this.landscapeKey = so.getActualLandscapeKey() == null ? (so.getLandscapeGenotypes().isEmpty() ? null : so.getLandscapeGenotypes().lastKey()) : so.getActualLandscapeKey();
93         this.portraitKey = so.getActualPortraitKey() == null ? (so.getPortaitGenotypes().isEmpty() ? null : so.getPortaitGenotypes().lastKey()) : so.getActualPortraitKey();
94         setTitle(this.soList.getList().get(idSOClicked).getName());
95         this.listView = (ListView) findViewById(R.id.smListView);
96         this.soGUILayout = (ScrollView) findViewById(R.id.soGUILayout);
97         this.progressLayout = (LinearLayout) findViewById(R.id.progressLayout);
98         this.tvProgressMsg = (TextView) findViewById(R.id.tvProgressMsg);
99         this.tvProgressMsg.setText(getText(R.string.pmgeneratinggui));
100        this.facade = new ApplicationFacade();
101        this.decoder = new AndroidDecoder();
102        this.soGUILayout.getViewTreeObserver().addOnGlobalLayoutListener(new ViewTreeObserver.OnGlobalLayoutListener() {
103            @Override
104            public void onGlobalLayout() {
105                soGUILayout.getViewTreeObserver().removeGlobalOnLayoutListener(this);
106                width = getWindow().findViewById(Window.ID_ANDROID_CONTENT).getWidth();
107                height = getWindow().findViewById(Window.ID_ANDROID_CONTENT).getHeight();
108                isLandscape = width > height;
109                getSlidingMenu().setBehindWidth((int) (width * .8));
110                if (isLandscape && landscapeKey != null) {
111                    decode(so.getLandscapeGenotypes().get(landscapeKey));
112                } else if (!isLandscape && portraitKey != null) {
113                    decode(so.getPortaitGenotypes().get(portraitKey));
114                } else {
115                    generateGUI();
116                }
117            }
118        });
119        setSlidingMenu();
120        mHandler = new Handler() {
121            @Override
122            public void handleMessage(Message msg) {
123                super.handleMessage(msg);
124                switch (msg.what) {
125                    case 1:
126                        menu.findItem(R.id.refresh_option).setEnabled(true);
127                        verifyMenuItemsStatus();
128                        break;
129                    case 2:
130                        serviceName.setText((String) msg.obj);
131                        break;
132                    case 3:
133                        makeToast((String) msg.obj);
134                    default:
135                        break;
136                }
137            }
138        };
139    }
140
141    private void setSlidingMenu() {
```

SmartObjectGUIActivity.java

```
142 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, this.soList.getSONames());
143 this.listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
144 this.listView.setAdapter(adapter);
145 this.listView.setOnItemClickListener(this);
146 SlidingMenu sm = getSlidingMenu();
147 sm.setShadowWidthRes(R.dimen.shadow_width);
148 sm.setShadowDrawable(R.drawable.shadow);
149 sm.setBehindOffsetRes(R.dimen.slidingmenu_offset);
150 sm.setFadeDegree(.6f);
151 sm.setTouchModeAbove(SlidingMenu.TOUCHMODE_MARGIN);
152 sm.setMode(SlidingMenu.LEFT);
153 sm.setBehindScrollScale(.0f);
154 sm.setBehindCanvasTransformer(new CanvasTransformer() {
155     @Override
156     public void transformCanvas(Canvas canvas, float percentOpen) {
157         float scale = (float) (percentOpen * 0.25 + 0.75);
158         canvas.scale(scale, scale, canvas.getWidth() / 2, canvas.getHeight() / 2);
159     }
160 });
161 setSlidingActionBarEnabled(false);
162 getSupportActionBar().setDisplayHomeAsUpEnabled(true);
163 }
164
165 @Override
166 public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
167     if (this.so != this.soList.getList().get(arg2)) {
168         this.idSOClicked = arg2;
169         this.so = this.soList.getList().get(this.idSOClicked);
170         this.landscapeKey = so.getActualLandscapeKey() == null ? (so.getLandscapeGenotypes().isEmpty() ? null : so.getLandscapeGenotypes().lastKey()) : so.getActualLandscapeKey();
171         this.portraitKey = so.getActualPortraitKey() == null ? (so.getPortaitGenotypes().isEmpty() ? null : so.getPortaitGenotypes().lastKey()) : so.getActualPortraitKey();
172         setTitle(this.so.getName());
173         if (isLandscape && landscapeKey != null) {
174             decode(so.getLandscapeGenotypes().get(landscapeKey));
175         } else if (!isLandscape && portraitKey != null) {
176             decode(so.getPortaitGenotypes().get(portraitKey));
177         } else {
178             generateGUI();
179         }
180     }
181     getSlidingMenu().toggle();
182 }
183
184 @Override
185 public boolean onCreateOptionsMenu(Menu menu) {
186     new MenuInflater(this).inflate(R.menu.guimenu, menu);
187     this.menu = menu;
188     verifyMenuItemsStatus();
189
190     return true;
191 }
192
193 private void verifyMenuItemsStatus() {
194     if (isLandscape) {
195         if (landscapeKey == null) {
196             setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_disabled, false);
197             setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_disabled, false);
198         } else {
199             if (so.getLandscapeGenotypes().lowerKey(landscapeKey) == null) {
200                 setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_disabled, false);
201             } else {
202                 setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_enabled, true);
203             }
204             if (so.getLandscapeGenotypes().higherKey(landscapeKey) == null) {
205                 setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_disabled, false);
206             } else {
207                 setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_enabled, true);
208             }
209         }
210     } else {
211         if (portraitKey == null) {
212             setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_disabled, false);
213             setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_disabled, false);
214         } else {
215             if (so.getPortaitGenotypes().lowerKey(portraitKey) == null) {
216                 setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_disabled, false);
217             } else {
218                 setEnabledMenuItem(R.id.next_gui, R.drawable.ic_action_next_item_enabled, true);
219             }
220             if (so.getPortaitGenotypes().higherKey(portraitKey) == null) {
221                 setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_disabled, false);
222             } else {
223                 setEnabledMenuItem(R.id.previous_gui, R.drawable.ic_action_previous_item_enabled, true);
224             }
225         }
226     }
227 }
228
229 private void setEnabledMenuItem(int itemId, int iconId, boolean enabled) {
230     this.menu.findItem(itemId).setEnabled(enabled);
231     this.menu.findItem(itemId).setIcon(iconId);
232 }
233
234 @Override
235 public boolean onOptionsItemSelected(MenuItem item) {
236     if (item.getItemId() == R.id.refresh_option) {
237         generateGUI();
238     } else if (item.getItemId() == R.id.next_gui) {
239         if (isLandscape) {
240             Double nextLandscapeKey = so.getLandscapeGenotypes().lowerKey(landscapeKey);
241             landscapeKey = nextLandscapeKey;
242             so.setActualLandscapeKey(nextLandscapeKey);
243             decode(so.getLandscapeGenotypes().get(nextLandscapeKey));
244         }
245     }
246 }
```

```

SmartObjectGUIActivity.java

244     } else {
245         Double nextPortraitKey = so.getPortraitGenotypes().lowerKey(portraitKey);
246         portraitKey = nextPortraitKey;
247         so.setActualPortraitKey(nextPortraitKey);
248         decode(so.getPortraitGenotypes().get(nextPortraitKey));
249     }
250 } else if (item.getItemId() == R.id.previous_gui) {
251     if (isLandscape) {
252         Double previousLandscapeKey = so.getLandscapeGenotypes().higherKey(landscapeKey);
253         landscapeKey = previousLandscapeKey;
254         so.setActualLandscapeKey(previousLandscapeKey);
255         decode(so.getLandscapeGenotypes().get(previousLandscapeKey));
256     } else {
257         Double previousPortraitKey = so.getPortraitGenotypes().higherKey(portraitKey);
258         portraitKey = previousPortraitKey;
259         so.setActualPortraitKey(previousPortraitKey);
260         decode(so.getPortraitGenotypes().get(previousPortraitKey));
261     }
262 } else {
263     getSlidingMenu().toggle();
264 }
265
266     return true;
267 }
268
269 @Override
270 public void onDestroy() {
271     super.onDestroy();
272     Sharedpreferences settings = getSharedpreferences(PREFS_NAME, 0);
273     Sharedpreferences.Editor editor = settings.edit();
274     editor.putInt("idSOClicked", this.idSOClicked);
275     editor.commit();
276     if (loop != null) {
277         loop.interrupt();
278     }
279     Thread save = new Thread(new Runnable() {
280         public void run() {
281             try {
282                 facade.saveSmartobjectListToDisk(soList, openfileoutput("SOFILE", Context.MODE_PRIVATE));
283             } catch (FileNotFoundException e) {
284                 e.printStackTrace();
285             }
286         }
287     });
288     save.start();
289     try {
290         save.join();
291     } catch (InterruptedException e) {
292         e.printStackTrace();
293     }
294 }
295
296 @Override
297 public void onBackpressed() {
298     super.onBackpressed();
299     this.idSOClicked = -1;
300     if (loop != null) {
301         loop.interrupt();
302     }
303 }
304
305 public void generateGUI() {
306     startTransaction(this);
307 }
308
309 private void decode(final Genotype genotype) {
310     new Thread(new Runnable() {
311         public void run() {
312             Androidphenotype phenotype = (Androidphenotype) decoder.decode(genotype, SmartObjectGUIActivity.this);
313             mHandler.obtainMessage(1, phenotype).sendToTarget();
314         }
315     }).start();
316 }
317
318 @Override
319 public void preExecute() {
320     // swapLayouts();
321 }
322
323 @Override
324 public void execute() throws Exception {
325     if (this.isSendCommand) {
326         ack = facade.sendSmartobjectCommand(so.getUrl(), this.listParameters);
327     } else {
328         Androidphenotype phenotype = (Androidphenotype) this.facade.generateGUI(this.soList, this.idSOClicked, width, height, this,
329         openfileoutput("SOFILE", Context.MODE_PRIVATE));
330         this.layout = phenotype.getLayout();
331         if (isLandscape) {
332             landscapeKey = this.so.getLandscapeGenotypes().lastKey();
333         } else {
334             portraitKey = this.so.getPortraitGenotypes().lastKey();
335         }
336         ack = "Tela gerada com sucesso!";
337     }
338 }
339
340 @Override
341 public void postExecute() {
342     if (this.isSendCommand) {
343         this.isSendCommand = false;
344     } else {
345         setEnabledMenuItem(R.id.refresh_option, R.drawable.ic_action_refresh_enabled, true);
346         verifyMenuitemsStatus();
347         setLayout(this.layout);
348     }
349 }

```

```

SmartObjectGUIActivity.java

347     // swapLayouts();
348 }
349 Toast.makeText(this, ack, Toast.LENGTH_SHORT).show();
350 }
351
352 @Override
353 public void swapLayouts() {
354     this.progressLayout.setVisibility(this.progressLayout.getVisibility() == LinearLayout.VISIBLE ? LinearLayout.INVISIBLE : LinearLayout.VISIBLE);
355 }
356
357 public void setLayout(final View layout) {
358     ViewGroupUtils.replaceView(soGUILayout, layout);
359     layout.invalidate();
360     soGUILayout = (ScrollView) layout;
361 }
362
363 public void executeService(final View view) {
364     final SOLinearLayout frame = (SOLinearLayout) view.getParent().getParent();
365     serviceName = ((TextView) ((LinearLayout) frame.getChildAt(0)).getChildAt(0));
366     this.listParameters = new ArrayList<String>();
367     this.listParameters.add("idSMModbus");
368     this.listParameters.add(so.getIdSMModbus());
369     this.listParameters.add("idServiceModbus");
370     this.listParameters.add(so.getIdServiceModbus(String.valueOf(serviceName.getText())));
371     this.listParameters.add("idRegisterModbus");
372     this.listParameters.add(so.getIdRegisterModbus(serviceName.getText().toString()));
373     // this.listParameters.add(serviceName.getText().toString());
374     this.flFirstTime = true;
375     createServiceURL(this.listParameters, frame);
376     this.isSendCommand = true;
377     serviceName = null;
378     if (listParameters.get(3).equals("03")) {
379         serviceName = getAckTextView();
380     }
381     new Thread(new Runnable() {
382         public void run() {
383             ack = facade.sendSmartObjectCommand(so.getUrl(), listParameters);
384             if (serviceName == null) {
385                 mHandler.obtainMessage(3, ack).sendToTarget();
386             } else {
387                 mHandler.obtainMessage(2, ack).sendToTarget();
388             }
389         }
390     }).start();
391     // startTransaction(this);
392 }
393
394 /**
395 * @return
396 */
397 private TextView getAckTextView() {
398     TextView tv = null;
399     LinearLayout layout = (LinearLayout) soGUILayout.getChildAt(0);
400     for (int i = 0; i < layout.getChildCount(); i++) {
401         View view = layout.getChildAt(i);
402         if (view instanceof SOLinearLayout && ((SOLinearLayout) view).isGetLayout()) {
403             LinearLayout child = (LinearLayout) ((LinearLayout) view).getChildAt(1);
404             tv = (TextView) child.getChildAt(1);
405         }
406     }
407
408     return tv;
409 }
410
411 private void makeToast(String message) {
412     Toast.makeText(SmartObjectGUIActivity.this, message, Toast.LENGTH_SHORT).show();
413 }
414
415 private void createServiceURL(List<String> list, View root) {
416     int i = 0;
417     if (this.flFirstTime) {
418         i = 1;
419         this.flFirstTime = false;
420     }
421     if (!this.flFirstTime && !(root instanceof LinearLayout)) {
422         if (root instanceof CheckBox) {
423             list.add(String.valueOf(((CheckBox) root).isChecked()));
424         } else if (root instanceof EditText) {
425             list.add(((EditText) root).getText().toString());
426         } else if (root instanceof TextView) {
427             String tv = ((TextView) root).getText().toString();
428             if (tv.contains(":")) {
429                 list.add(tv.substring(0, tv.indexOf(" ")));
430             } else {
431                 list.add(tv);
432             }
433         } else if (root instanceof Spinner) {
434             list.add((String) ((Spinner) root).getSelectedItem());
435         } else if (root instanceof NegativeSeekBar) {
436             list.add(((NegativeSeekBar) root).getStringProgress());
437         } else if (root instanceof RadioGroup) {
438             RadioGroup rg = (RadioGroup) root;
439             RadioButton rb = (RadioButton) rg.getChildAt(rg.getCheckedRadioButtonId());
440             list.add(rb.getText().toString());
441         }
442
443         return;
444     } else {
445         for (; i < ((LinearLayout) root).getChildCount(); i++) {
446             createServiceURL(list, ((LinearLayout) root).getChildAt(i));
447         }
448
449         return;
450     }
}

```

SmartObjectGUIActivity.java

```
451  
452     }  
453 }  
454
```

```
1 /**
4 package br.com.mobgui4so.view;
5
6 import android.content.Context;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public class SOLinearLayout extends LinearLayout {
14
15     private boolean isFrameLooping = false;
16     private long loopingInterval;
17     private boolean isGetLayout = false;
18
19     public SOLinearLayout(Context context) {
20         super(context);
21     }
22
23     public SOLinearLayout(Context context, AttributeSet attrs) {
24         super(context, attrs);
25     }
26
27     public SOLinearLayout(Context context, AttributeSet attrs, int defStyle) {
28         super(context, attrs, defStyle);
29     }
30
31     public boolean isFrameLooping() {
32         return isFrameLooping;
33     }
34
35     public void setFrameLooping(boolean isFrameLooping) {
36         this.isFrameLooping = isFrameLooping;
37     }
38
39     public long getLoopingInterval() {
40         return loopingInterval;
41     }
42
43     public void setLoopingInterval(long loopingInterval) {
44         this.loopingInterval = loopingInterval;
45     }
46
47     public boolean isGetLayout() {
48         return isGetLayout;
49     }
50
51     public void setGetLayout(boolean isGetLayout) {
52         this.isGetLayout = isGetLayout;
53     }
54
55 }
```

TransactionTask.java

```
1 /**
4 package br.com.mobgui4so.view;
5
6 import android.content.Context;
9
10 /**
11 * @author Ercilio Nascimento
12 */
13 public class TransactionTask extends AsyncTask<Void, Void, Boolean> {
14
15     private final Context context;
16     private final ITransaction transaction;
17     private Throwable exceptionErro;
18
19     public TransactionTask(Context context, ITransaction transaction) {
20         this.context = context;
21         this.transaction = transaction;
22     }
23
24     @Override
25     protected void onPreExecute() {
26         transaction.preExecute();
27     }
28
29     @Override
30     protected Boolean doInBackground(Void... params) {
31         try {
32             transaction.execute();
33         } catch (Throwable e) {
34             this.exceptionErro = e;
35             return false;
36         }
37         return true;
38     }
39
40     @Override
41     protected void onPostExecute(Boolean result) {
42         if (result) {
43             transaction.postExecute();
44         } else {
45             AndroidUtils.alertDialog(context, "Erro: " + exceptionErro.getMessage());
46         }
47     }
48 }
49
```

```
1 /**
2 package br.com.mobgui4so.view;
3
4 import android.view.View;
5
6 /**
7 * @author Ercilio Nascimento
8 */
9
10 public class ViewGroupUtils {
11
12     public static ViewGroup getParent(View view) {
13         return (ViewGroup) view.getParent();
14     }
15
16     public static void removeView(View view) {
17         ViewGroup parent = getParent(view);
18         if (parent != null) {
19             parent.removeView(view);
20         }
21     }
22
23     public static void replaceView(View currentView, View newView) {
24         ViewGroup parent = getParent(currentView);
25         if (parent == null) {
26             return;
27         }
28         final int index = parent.indexOfChild(currentView);
29         removeView(currentView);
30         removeView(newView);
31         parent.addView(newView, index);
32     }
33 }
34
35 }
```

AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="br.com.mobgui4so" android:versionCode="1"
4     android:versionName="1.0"
5     >
6     <uses-permission android:name="android.permission.INTERNET" />
7     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
8     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
9
10    <uses-sdk
11        android:minSdkVersion="11"
12        android:targetSdkVersion="18" />
13
14    <application
15        android:allowBackup="true"
16        android:icon="@drawable/ic_launcher"
17        android:label="@string/app_name"
18        android:theme="@style/AppTheme"
19        android:hardwareAccelerated="true" >
20        <activity
21            android:name="br.com.mobgui4so.view.MainActivity"
22            android:label="@string/app_name"
23            android:screenOrientation="portrait"
24            android:theme="@style/MainAppTheme"
25            >
26            <intent-filter>
27                <action android:name="android.intent.action.MAIN" />
28
29                <category android:name="android.intent.category.LAUNCHER" />
30            </intent-filter>
31        </activity>
32        <activity
33            android:name="br.com.mobgui4so.view.ListSmartObjectActivity"
34            android:label="@string/title_activity_list_smart_object" >
35        </activity>
36        <activity
37            android:name="br.com.mobgui4so.view.SmartObjectGUIActivity"
38            android:label="@string/title_activity_smart_object_gui" >
39        </activity>
40    </application>
41
42
43 </manifest>
```

appConfig.properties

```
1# ----- #  
2# ----- THIS PROPERTIES FILE CONTAINS APPLICATION CONFIGURATION PARAMETERS ----- #  
3# ----- #  
4  
5  
6# Class names of each discovery technology already implemented.  
7# This makes the algorithm be dynamic.  
8  
9 WiFi=br.com.mobgui4so.model.discovery.WiFiSmartObjectDiscovery  
10 Bluetooth=br.com.mobgui4so.model.discovery.BluetoothSmartObjectDiscovery  
11 Test=br.com.mobgui4so.model.discovery.TestSmartObjectDiscovery  
12
```

Anexo C – Código Fonte SORepository

```

1 /**
4 package br.com.sorepository.model.dao;
5
6 import java.sql.Connection;
16
17 /**
18 * @author Ercilio Nascimento
19 */
20 public class RequestDAO {
21
22     private final String user = "root";
23     private final String password = "anelise";
24     private final String jdbc = "com.mysql.jdbc.Driver";
25     private final String url = "jdbc:mysql://localhost:3306/sodb";
26     private Connection conn;
27
28     public RequestDAO() throws ClassNotFoundException, SQLException {
29         Class.forName(this.jdbc);
30         conn = DriverManager.getConnection(this.url, this.user, this.password);
31     }
32
33     public void closeConnection() {
34         try {
35             this.conn.close();
36         } catch (SQLException e) {
37             // TODO Auto-generated catch block
38             e.printStackTrace();
39         }
40     }
41
42     public boolean validateUser(String user, String password) throws SQLException {
43         boolean validate = false;
44         StringBuilder sql = new StringBuilder();
45         sql.append("SELECT `U`.`USERNAME` FROM `SODB`.`USER` U ");
46         sql.append("WHERE `U`.`USERNAME` = ? ");
47         sql.append("AND `U`.`PASSWORD` = ? ");
48         PreparedStatement prst = conn.prepareStatement(sql.toString());
49         prst.setString(1, user);
50         prst.setString(2, password);
51         ResultSet rs = prst.executeQuery();
52
53         if (rs.next()) {
54             validate = true;
55         }
56
57         System.out.println("Validating user...");
58         log("SQL - VALIDATE USER: ", sql, new Object[] { user, password });
59
60         return validate;
61     }
62
63     private void log(String info, StringBuilder sql, Object... params) {
64         for (int i = 0; i < params.length; i++) {
65             int index = sql.indexOf("?");
66             if (params[i] instanceof String) {
67                 sql = sql.replace(index, ++index, "" + String.valueOf(params[i]) + "");
68             } else {
69                 sql = sql.replace(index, ++index, String.valueOf(params[i]));
70             }
71         }
72         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy hh:mm:ss");
73         System.out.println(sdf.format(Calendar.getInstance().getTime()) + " " + info + " " + sql.toString());
74     }
75
76     public SmartObjectList getSOList(String user) throws SQLException {
77         SmartObjectList solist = new SmartObjectList();
78         StringBuilder sql = new StringBuilder();
79         sql.append("SELECT `SO`.`IDSMARTOBJECT`, `SO`.`IDSOMODBUS`, `SO`.`SERVERURL`, ");
80         sql.append("`CA`.`NAME`, `SER`.`IDSERVICEMODBUS`, `SER`.`IDREGISTERMODBUS`, `SER`.`NAME`, `P`.`NAME`, `P`.`TYPE`, ");
81         sql.append("`P`.`MINVALUE`, `P`.`MAXVALUE`, `P`.`OPTIONS` ");
82         sql.append("FROM `SODB`.`USER` U ");
83         sql.append("JOIN `SODB`.`SOUSERJOIN` SOU ON `SOU`.`IDUSER` = `U`.`IDUSER` ");
84         sql.append("JOIN `SODB`.`SMARTOBJECT` SO ON `SO`.`IDSMARTOBJECT` = `SOU`.`IDSMARTOBJECT` ");
85         sql.append("JOIN `SODB`.`CATEGORY` CA ON `CA`.`IDCATEGORY` = `SO`.`IDCATEGORY` ");
86         sql.append("JOIN `SODB`.`SERVICE` SER ON `SER`.`IDCATEGORY` = `CA`.`IDCATEGORY` ");
87         sql.append("(LEFT OUTER JOIN `SODB`.`SERVICEPARAMETER` SERPAR ON `SERPAR`.`IDSERVICE` = `SER`.`IDSERVICE` ");
88         sql.append("(LEFT OUTER JOIN `SODB`.`PARAMETER` P ON `P`.`IDPARAMETER` = `SERPAR`.`IDPARAMETER` ");
89         sql.append("WHERE `U`.`USERNAME` = ? ");
90         sql.append("ORDER BY `SO`.`IDSMARTOBJECT` , ");
91         sql.append("`SER`.`IDSERVICE` , ");
92         sql.append("`P`.`IDPARAMETER` ");
93         PreparedStatement prst = conn.prepareStatement(sql.toString());
94         prst.setString(1, user);
95         ResultSet rs = prst.executeQuery();
96
97         while (rs.next()) {
98             solist.getList().add(getConcatValues(rs));
99         }
100
101        System.out.println("Getting smart object list...");
102        log("SQL - GET SO LIST: ", sql, new Object[] { user });
103
104        return solist;
105    }
106
107
108    public String refreshSO(String command) throws SQLException {
109        StringBuilder sql = new StringBuilder();
110        SmartObject so = SmartObject.splitCommandToSmartObject(command);
111        if (this.existsSO(so.getSoName())) {
112            // TODO atualizar SO
113        }
114        return "objeto inteligente refrescado com sucesso!";
115    }

```

RequestDAO.java

```
116 private boolean existsSO(String name) throws SQLException {
117     boolean exists = false;
118     StringBuilder sql = new StringBuilder();
119     sql.append("SELECT 1 FROM `SO` WHERE `SO`.`NAME` = ?");
120     PreparedStatement prst = conn.prepareStatement(sql.toString());
121     prst.setString(1, name);
122     ResultSet rs = prst.executeQuery();
123
124     if (rs.next()) {
125         exists = true;
126     }
127
128     log("SQL - EXISTS SO: ", sql, new Object[] { name });
129
130     return exists;
131 }
132
133
134 private String getConcatValues(ResultSet rs) throws SQLException {
135     String s = "";
136     for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
137         s += rs.getString(i) + ",";
138     }
139     s = s.substring(0, s.length() - 1);
140
141     return s;
142 }
143 }
144 }
```

```

1 /**
4 package br.com.sorepository.model.pojo;
5
6 /**
7 * @author Ercilio Nascimento
8 */
9 public class SmartObject {
10
11    private String soName;
12    private String serviceName;
13    private String serviceReturnType;
14    private String param1;
15    private String param2;
16    private String param3;
17    private String param4;
18    private String param5;
19
20    public SmartObject() {
21    }
22
23    public static SmartObject splitCommandToSmartObject(String command) {
24        SmartObject so = new SmartObject();
25        String[] array = command.split(",");
26        so.setSoName(array[0]);
27        so.setServiceName(array[1]);
28        so.setServiceReturnType(array[2]);
29        so.setParam1(array[3]);
30        so.setParam2(array[4]);
31        so.setParam3(array[5]);
32        so.setParam4(array[6]);
33        so.setParam5(array[7]);
34
35        return so;
36    }
37
38    public String getSoName() {
39        return soName;
40    }
41
42    public void setSoName(String soName) {
43        this.soName = soName;
44    }
45
46    public String getServiceName() {
47        return serviceName;
48    }
49
50    public void setServiceName(String serviceName) {
51        this.serviceName = serviceName;
52    }
53
54    public String getServiceReturnType() {
55        return serviceReturnType;
56    }
57
58    public void setServiceReturnType(String serviceReturnType) {
59        this.serviceReturnType = serviceReturnType;
60    }
61
62    public String getParam1() {
63        return param1;
64    }
65
66    public void setParam1(String param1) {
67        this.param1 = param1;
68    }
69
70    public String getParam2() {
71        return param2;
72    }
73
74    public void setParam2(String param2) {
75        this.param2 = param2;
76    }
77
78    public String getParam3() {
79        return param3;
80    }
81
82    public void setParam3(String param3) {
83        this.param3 = param3;
84    }
85
86    public String getParam4() {
87        return param4;
88    }
89
90    public void setParam4(String param4) {
91        this.param4 = param4;
92    }
93
94    public String getParam5() {
95        return param5;
96    }
97
98    public void setParam5(String param5) {
99        this.param5 = param5;
100   }
101
102 }
103

```

```
1 /**
4 package br.com.sorepository.model.pojo;
5
6 import java.util.ArrayList;
8
9 /**
10 * @author Ercilio Nascimento
11 */
12 public class SmartObjectList {
13
14     private List<String> list;
15
16     public SmartObjectList() {
17         this.list = new ArrayList<String>(0);
18     }
19
20     public List<String> getList() {
21         return list;
22     }
23
24     public void setList(List<String> list) {
25         this.list = list;
26     }
27
28     public String printList() {
29         String s = "";
30         for (int i = 0; i < this.list.size(); i++) {
31             s += this.list.get(i) + ";";
32         }
33
34     return s;
35 }
36
37 }
38 }
```

```

1 package br.com.sorepository.controller.servlet;
2
3 import java.io.BufferedReader;
4
5 /**
6  * @author Ercilio Nascimento
7 */
8 public class S0RequestServlet extends HttpServlet {
9     private static final long serialVersionUID = 1L;
10
11     public S0RequestServlet() {
12         super();
13     }
14
15     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16         doPost(request, response);
17     }
18
19     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
20         PrintWriter writer = response.getWriter();
21         String out = null;
22         String user = request.getParameter("user");
23         String password = request.getParameter("password");
24         RequestDAO dao = null;
25         SmartObjectList list = new SmartObjectList();
26
27         StringBuffer sb = new StringBuffer();
28         BufferedReader bufferedReader = null;
29         String content = "";
30
31         System.out.println(user + password);
32
33         try {
34             bufferedReader = request.getReader();
35             char[] charBuffer = new char[128];
36             int bytesRead;
37             while ((bytesRead = bufferedReader.read(charBuffer)) != -1) {
38                 sb.append(charBuffer, 0, bytesRead);
39             }
40
41             dao = new RequestDAO();
42             if (dao.validateUser(user, password)) {
43                 list = dao.getSOList(user);
44                 out = list.printList();
45             } else {
46                 out = "O Usuario ou senha invalido!";
47             }
48
49             } catch (IOException ex) {
50                 throw ex;
51             } catch (ClassNotFoundException e) {
52                 e.printStackTrace();
53             } catch (SQLException e) {
54                 e.printStackTrace();
55             } finally {
56                 if (bufferedReader != null) {
57                     try {
58                         bufferedReader.close();
59                     } catch (IOException ex) {
60                         throw ex;
61                     }
62                 }
63             }
64
65             writer.write(StringEscapeUtils.escapeJava(out));
66
67             System.out.println(sb);
68         }
69     }
70 }
71
72 }
```

Anexo D – Código Fonte SOServer

BasePortManager.java

```
1 /**
4 package br.com.soserver.comm;
5 /**
7 * @author Ercilio Nascimento
8 */
9 public abstract class BasePortManager {
10
11     public abstract String write(byte[] command);
12
13 }
14
```

ParserCommand.java

```
1 /**
4 package br.com.soserver.comm;
5
6 import java.util.Arrays;
10
11 /**
12 * @author Ercilio Nascimento
13 */
14 public class ParserCommand {
15
16     private BasePortManager manager;
17
18     public ParserCommand() {
19         this.manager = PortManager.getInstance();
20         // this.manager = new SimulatePortManager();
21     }
22
23     public String send(String command) {
24         String modbusReturn = manager.write(parseCommandToASCIIModBus(command).getBytes());
25
26         return parseModbusReturn(modbusReturn);
27     }
28
29     private String parseModbusReturn(String modbus) {
30         String modbusReturn = null;
31         if (modbus.equals("OK")) {
32             modbusReturn = "Operação executada com sucesso!";
33         } else {
34             modbusReturn = String.valueOf(Integer.valueOf(modbus.substring(7, 9), 16));
35         }
36
37         return modbusReturn;
38     }
39
40     private String parseCommandToASCIIModBus(String command) {
41         // command = "idSOModbus=A3&idServiceModbus=03&idRegisterModbus=00&ligar=false";
42         System.out.println("Command: " + command);
43         System.out.println("Parsing command to ASCII Modbus...");
44         StringBuilder modBus = new StringBuilder(":");
45         List<String> params = Arrays.asList(command.split("&"));
46         String AA = StringUtils.substringAfter(params.get(0), "=");
47         String BB = StringUtils.substringAfter(params.get(1), "=");
48         String CC = StringUtils.substringAfter(params.get(2), "=");
49         String DD = "";
50         int AAhex = Integer.valueOf(AA, 16);
51         int BBhex = Integer.valueOf(BB, 16);
52         int CChex = Integer.valueOf(CC, 16);
53         int DDhex = 0x00;
54         if (BBhex != 0x03) {
55             DD = StringUtils.substringAfter(params.get(3), "=");
56             if (DD.equalsIgnoreCase("false")) {
57                 DD = "00";
58             } else if (DD.equalsIgnoreCase("true") && AA.equals("A1")) {
59                 DD = "FF";
60                 DDhex = 0xFF;
61             } else if (DD.equalsIgnoreCase("true") && AA.equals("A2")) {
62                 DD = "01";
63                 DDhex = 0x01;
64             } else {
65                 DDhex = Integer.valueOf(DD);
66                 DD = Integer.toHexString(DDhex).toUpperCase();
67             }
68         }
69         String irc = Integer.toHexString(((AAhex + BBhex + DDhex + CChex) ^ 0xFF) + 1) & 0xFF;
70         modBus.append(AA);
71         modBus.append(BB);
72         modBus.append(CC);
73         modBus.append(DD);
74         modBus.append(irc.toUpperCase());
75         System.out.println("Command parsed: " + modBus.toString() + "\r\n");
76
77         return modBus.toString() + "\r\n";
78         // return ":A303005A\r\n";
79     }
80 }
```

PortManager.java

```

1 /**
4 package br.com.soserver.comm;
5
6 import gnu.io.CommPortIdentifier;
16
17 /**
18 * @author Ercilio Nascimento
19 */
20 public class PortManager extends BasePortManager implements SerialPortEventListener {
21     SerialPort serialPort;
22     /** The port we're normally going to use. */
23     private static final String PORT_NAMES[] = {
24         // "/dev/tty.usbserial-A9007UX1", // Mac OS X
25         // "/dev/ttyUSB0", // Linux
26         "COM9", // Windows
27     };
28
29     private static PortManager instance;
30     private String inputLine = "OK";
31
32     /**
33      * A BufferedReader which will be fed by a InputStreamReader
34      * converting the bytes into characters
35      * making the displayed results codepage independent
36      */
37     private BufferedReader input;
38     /** The output stream to the port */
39     private OutputStream output;
40     /** Milliseconds to block while waiting for port open */
41     private static final int TIME_OUT = 2000;
42     /** Default bits per second for COM port. */
43     private static final int DATA_RATE = 9600;
44
45     public static PortManager getInstance() {
46         if (instance == null) {
47             instance = new PortManager();
48             instance.initialize();
49         }
50
51         return instance;
52     }
53
54     private void initialize() {
55         CommPortIdentifier portId = null;
56         @SuppressWarnings("unchecked") Enumeration<CommPortIdentifier> portEnum = CommPortIdentifier.getPortIdentifiers();
57
58         // First, Find an instance of serial port as set in PORT_NAMES.
59         while (portEnum.hasMoreElements()) {
60             CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();
61             for (String portName : PORT_NAMES) {
62                 if (currPortId.getName().equals(portName)) {
63                     portId = currPortId;
64                     break;
65                 }
66             }
67         }
68         if (portId == null) {
69             System.out.println("Could not find COM port.");
70             return;
71         }
72
73         try {
74             // open serial port, and use class name for the appName.
75             serialPort = (SerialPort) portId.open(this.getClass().getName(), TIME_OUT);
76
77             // set port parameters
78             serialPort.setSerialPortParams(DATA_RATE, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
79             serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_RTSCTS_IN | SerialPort.FLOWCONTROL_RTSCTS_OUT);
80
81             // open the streams
82             input = new BufferedReader(new InputStreamReader(serialPort.getInputStream()));
83             output = serialPort.getOutputStream();
84
85             // add event listeners
86             serialPort.addEventListener(this);
87             serialPort.notifyOnDataAvailable(true);
88             // serialPort.enableReceiveTimeout(1000);
89             serialPort.enableReceiveThreshold(0);
90             System.out.println("Server is listening port " + serialPort.getName());
91         } catch (Exception e) {
92             System.err.println(e.toString());
93         }
94     }
95
96     /**
97      * This should be called when you stop using the port.
98      * This will prevent port locking on platforms like Linux.
99      */
100    public synchronized void close() {
101        if (serialPort != null) {
102            serialPort.removeEventListener();
103            serialPort.close();
104        }
105    }
106
107    /**
108     * Handle an event on the serial port. Read the data and print it.
109     */
110    public synchronized void serialEvent(SerialPortEvent oEvent) {
111        if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
112            try {
113                inputLine = input.readLine();
114            } catch (Exception e) {
115                System.err.println(e.toString());
116            }
117        }
118    }

```

PortManager.java

```
116     }
117 }
// Ignore all the other eventTypes, but you should consider the other ones.
118 }
119
120 @Override
121 public String write(byte[] command) {
122     try {
123         System.out.println("Sending smart object instructions...");
124         output.write(command);
125         Thread.sleep(2000);
126         output.flush();
127         output.close();
128     } catch (IOException e) {
129         e.printStackTrace();
130     } catch (InterruptedException e) {
131         e.printStackTrace();
132     }
133
134     return inputLine;
135 }
136 }
137 }
138 }
```

SimulatePortManager.java

```
1 /**
4 package br.com.soserver.comm;
5
6 /**
7 * @author Ercilio Nascimento
8 */
9 public class SimulatePortManager extends BasePortManager {
10
11     @Override
12     public String write(byte[] command) {
13         String ack = null;
14         String commandString = new String(command);
15         int id = Integer.valueOf(commandString.substring(1, 3), 16);
16         ack = "Comando executado com sucesso!";
17         switch (id) {
18             case 0xA1:
19                 break;
20             case 0xA2:
21                 break;
22             case 0xA3:
23                 break;
24             default:
25                 break;
26         }
27         return ack;
28     }
29 }
30 }
```

```

1 package br.com.soserver.controller.servlet;
2
3 import java.io.BufferedReader;
4
5 /**
6  * @author Ercilio Nascimento
7 */
8
9 public class SOCommandServlet extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     public SOCommandServlet() {
13         super();
14     }
15
16     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
17         /*PrintWriter writer = response.getWriter();
18         StringBuffer sb = new StringBuffer();
19         BufferedReader bufferedReader = null;
20         String ack = "";
21         ParserCommand parser = new ParserCommand();
22
23         try {
24             bufferedReader = request.getReader();
25             char[] charBuffer = new char[128];
26             int bytesRead;
27             while ((bytesRead = bufferedReader.read(charBuffer)) != -1) {
28                 sb.append(charBuffer, 0, bytesRead);
29             }
30
31             ack = parser.send(sb.toString());
32
33         } catch (IOException ex) {
34             throw ex;
35         } finally {
36             if (bufferedReader != null) {
37                 try {
38                     bufferedReader.close();
39                 } catch (IOException ex) {
40                     throw ex;
41                 }
42             }
43         }
44
45         writer.write(StringEscapeUtils.escapeJava(ack));
46
47         System.out.println(ack);*/
48         doPost(request, response);
49     }
50
51     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
52         PrintWriter writer = response.getWriter();
53         StringBuffer sb = new StringBuffer();
54         BufferedReader bufferedReader = null;
55         String ack = "";
56         ParserCommand parser = new ParserCommand();
57
58         try {
59             bufferedReader = request.getReader();
60             char[] charBuffer = new char[128];
61             int bytesRead;
62             while ((bytesRead = bufferedReader.read(charBuffer)) != -1) {
63                 sb.append(charBuffer, 0, bytesRead);
64             }
65
66             ack = parser.send(sb.toString());
67
68         } catch (IOException ex) {
69             throw ex;
70         } finally {
71             if (bufferedReader != null) {
72                 try {
73                     bufferedReader.close();
74                 } catch (IOException ex) {
75                     throw ex;
76                 }
77             }
78         }
79         writer.write(StringEscapeUtils.escapeJava(ack));
80
81         // System.out.println(sb);
82     }
83 }
84
85
86
87
88
89
90
91
92
93
94 }
```