Gustavo Paulo Medeiros da Silva

Desenvolvimento do Servidor de Automação Residencial do Sistema DroidLar para dispositivos embarcados

São José – SC

Outubro / 2012

Gustavo Paulo Medeiros da Silva

Desenvolvimento do Servidor de Automação Residencial do Sistema DroidLar para dispositivos embarcados

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Marcelo Maia Sobral, Dr.

Co-orientador:
Prof. Emerson Ribeiro de Mello, Dr.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Outubro / 2012

Monografia sob o título *"Desenvolvimento do Servidor de Automação Residencial do Sistema DroidLar para dispositivos embarcados"*, defendida por Gustavo Paulo Medeiros da Silva e aprovada em 22 de outubro de 2012, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Marcelo Maia Sobral, Dr.
Orientador

Prof. Evandro Cantú, Dr.
IFSC

Prof. Tiago Semprebom, Dr.
IFSC

“Acredite que você pode, assim você já está no meio do caminho.”

Theodore Roosevelt

Agradecimentos

Dedico meus sinceros agradecimentos à Deus, à minha família e os amigos que me deram forças para vencer muitos obstáculos e chegar até aqui.

Resumo

Desde a década de 70 a automação residencial vem ganhando espaço na área de pesquisa e desenvolvimento sendo o padrão X.10 o pioneiro. Esse padrão permitia que dispositivos eletro-eletrônicos fossem controlados remotamente através de uma central de controle ou ainda através de um controle remoto sem fio. Com a popularização dos telefones inteligentes e tablets, bem como os avanços dos padrões de redes sem fio, a área de automação residencial está novamente em foco, atraindo atenção de empresas e grupos de pesquisas. Entre 2010 e 2011 no Instituto Federal de Santa Catarina campus São José foi desenvolvido um protótipo de um sistema de automação residencial chamado de Droidlar. Esse sistema utiliza tecnologias como Android, kits Arduinos e redes sem fio ZigBee. Neste trabalho implementou-se em um sistema embarcado o componente SAR (Sistema de Automação Residencial) do Droidlar. Esse componente registra, recebe notificações e envia comandos para os dispositivos residenciais automatizados. A implementação do SAR realizada é compatível com dispositivos baseados em Linux, tais como pontos de acesso e roteadores com firmware Openwrt ou DD-Wrt (distribuições Linux feitas especialmente para equipamentos de rede de baixo custo).

Abstract

Since the 70's home automation has become more popular in research and development being the pioneer the X.10 standard. This standard allows electronic devices to be controlled remotely via a central control or via a wireless remote control. With the popularity of smart phones and tablets, and the advances in wireless networking standards, the area of home automation is again in focus, attracting attention from companies and research groups. Between 2010 and 2011 in Federal Institute of Santa Catarina campus São José was developed a prototype of a home automation system, the system DroidLar. This system uses technologies such as Android, Arduinos kits and ZigBee wireless networks. This work provided an implementation of the Droidlar's component SAR (Home Automation System) targeted to an embedded system. This component registers, receives notifications and send commands to automated home devices. Its implementation is compatible with Linux based devices, such as access points and routers running Openwrt or DD-Wrt (Linux distributions specially built for low-cost network devices).

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas p. 11

1 Introdução p. 12

1.1 Motivação . . . . . p. 12

1.2 Organização do texto . . . . . p. 13

2 Fundamentação p. 14

2.1 Projeto DroidLar . . . . . p. 14

2.1.1 Cliente Android . . . . . p. 15

2.1.2 Controladores de dispositivos . . . . . p. 15

2.1.3 Servidor de Automação Residencial (SAR) . . . . . p. 16

2.2 Sistemas embarcados . . . . . p. 17

2.2.1 Roteador sem fio TP-Link TL-WR941ND . . . . . p. 18

2.2.2 *BeagleBone* . . . . . p. 19

2.3 Sistemas operacionais para roteadores sem fio . . . . . p. 19

2.3.1 OpenWrt . . . . . p. 19

2.3.2 DD-WRT . . . . . p. 20

2.4 *Common Gateway Interface* (CGI) . . . . . p. 20

2.5 Bibliotecas C . . . . . p. 21

2.5.1 Servidor *web* . . . . . p. 21

2.5.2 Gerenciador do módulo XBee . . . . . p. 22

2.5.3 Banco de dados . . . . . p. 22

3 Servidor de Automação Residencial (SAR) p. 23

3.1 Entendendo o SAR . . . . . p. 23

3.2 Desenvolvimento do novo SAR . . . . . p. 24

3.2.1 Primeira proposta de desenvolvimento do novo SAR . . . . . p. 25

3.2.2 Segunda proposta de desenvolvimento do novo SAR . . . . . p. 27

3.3 Arquitetura . . . . . p. 32

3.4 Implementação do SAR em um sistema embarcado . . . . . p. 33

4 Conclusões p. 35

4.1 Trabalhos futuros . . . . . p. 36

Referências Bibliográficas p. 37

Lista de Figuras

2.1 Elementos do DroidLar. (EUZÉBIO, 2011) . . . . . p. 14

2.2 Interface principal e de configuração do SAR (EUZÉBIO, 2011) . . . . . p. 15

2.3 Protótipos do DroidLar (EUZÉBIO, 2011) . . . . . p. 17

2.4 Roteador sem fio TP-Link TL-WR941ND (TP-LINK, 2012). . . . . p. 18

2.5 BeagleBone (BEAGLEBONE, 2011). . . . . p. 19

3.1 Emulador de telefone celular com Android. . . . . p. 26

3.2 Proposta para o novo sistema DroidLar. . . . . p. 28

3.3 Arquitetura do SAR. . . . . p. 32

3.4 Novo cenário do sistema DroidLar. . . . . p. 33

Lista de Tabelas

3.1 Comparação das características das bibliotecas - Servidores *web*. . . . . p. 29

3.2 Características do libxbee. . . . . p. 30

3.3 Comparação das características das bibliotecas - Banco de dados. . . . . p. 30

Lista de Abreviaturas

API *Aplication Program Interface*

CGI *Common Gateway Interface*

CSMA/CA *Carrier Sense Multiple Access With Collision Avoidance*

GPL *GNU General Public License*

HTTP *Hypertext Transfer Protocol*

HTTPS *HyperText Transfer Protocol Secure*

SAR Servidor de Automação Residencial

SSL *Secure Sockets Layer*

TLS *Transport Layer Security*

USB *Universal Serial Bus*

WWW *World Wide Web*

1 Introdução

1.1 Motivação

Os sistemas de automação residencial presentes atualmente no mercado oferecem como interfaces de controle páginas *web* que podem ser acessadas de qualquer computador pessoal. Com a disseminação dos telefones celulares em todo mundo, tornou-se possível que eles também tivessem acesso a essas páginas *web*, porém, com uma interface mais simples por conta das limitações de *hardware* e *software* desses telefones (EUZÉBIO, 2011).

O surgimento dos telefones inteligentes abriu novas possibilidades para utilizar os dispositivos móveis como interfaces de controle para sistemas de automação residencial. Esses telefones possuem recursos como maior quantidade de memória e poder de processamento, aliados a sistemas operacionais mais sofisticados, se comparados aos celulares mais comuns (EUZÉBIO, 2011).

O surgimento do sistema operacional Android[1] para telefones inteligentes e a arquitetura de rede sem fio ZigBee[2] motivaram a criação do sistema de automação residencial DroidLar. Esse sistema de automação utiliza celulares Android como interface de controle, *kits* Arduino para o controle dos eletroeletrônicos, protocolos da especificação ZigBee para a comunicação dos *kits* com o servidor e um computador pessoal onde é hospedado o Servidor de Automação Residencial (SAR).

Para o sistema de automação funcionar corretamente, o SAR precisa estar ligado constantemente mas não é desejável manter um computador pessoal ligado o permanentemente somente para o SAR. Além disso, um computador é um recurso dispendioso e sobre-dimensionado para as necessidades computacionais do SAR. Desta forma, a motivação deste trabalho é implementar o SAR em um dispositivo embarcado, e assim viabilizar a implantação do sistema DroidLar com custo reduzido.

[1] http://www.android.com
[2] http://www.zigbee.org

1.2 Organização do texto

Esse texto está organizado em quatro capítulos. O capítulo 2 descreve sucintamente o sistema DroidLar, que é a base deste trabalho, e também as bibliotecas que foram utilizadas para o desenvolvimento do novo SAR, assim como as plataformas de hardware e software onde o SAR foi implementado. No capítulo 3 é apresentado o modelo para o novo SAR junto com sua implementação voltada a dispositivos embarcados baseados em Linux. No capítulo 4 são apresentadas as conclusões sobre este trabalho e apontados possíveis trabalhos futuros.

2 Fundamentação

2.1 Projeto DroidLar

Desenvolvido no *Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, campus São José* (IFSC campus São José) o DroidLar é um projeto de um sistema de automação residencial onde todos os seus componentes, *softwares* e *hardwares*, possuem licença livre. O objetivo desse projeto foi desenvolver um sistema de automação alternativo aos sistemas proprietários presentes no mercado.

O sistema DroidLar é formado por três partes, mostradas na figura 2.1: (1) cliente Android, com que um usuário pode monitorar e controlar os dispositivos residenciais; (2) o Servidor de Automação Residencial (SAR), que se comunica diretamente com os dispositivos residenciais e faz a interface entre eles e os clientes Android; (3) os controladores e dispositivos baseados em Arduino.

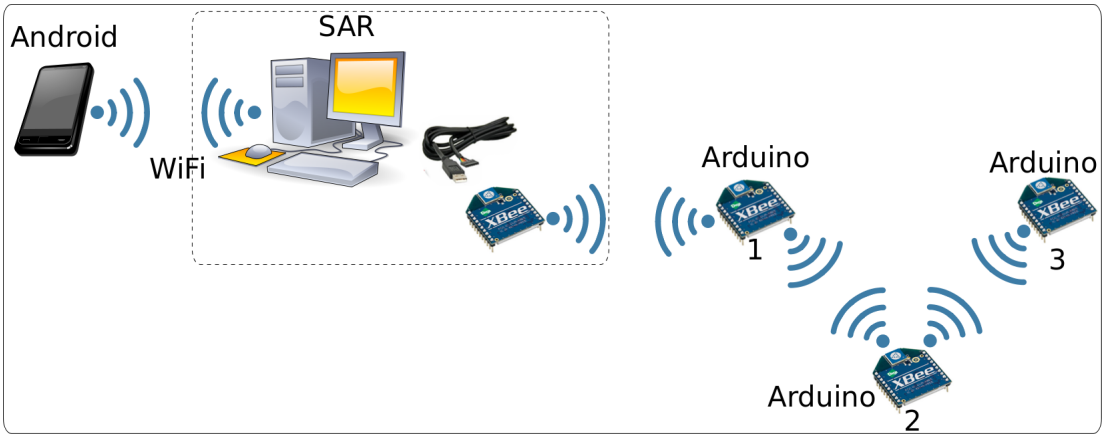

Figura 2.1: Elementos do DroidLar. (EUZÉBIO, 2011)

2.1.1 Cliente Android

Android[1] é um sistema operacional desenvolvido para dispositivos móveis inteligentes que roda sobre o núcleo Linux. Foi inicialmente desenvolvido pela Google e depois pela *Open Handset Alliance*, mas a Google é a responsável pela gerência do produto e engenharia de processos. O motivo da sua criação é devido as dificuldades de desenvolver uma aplicação para diferentes dispositivos pois os fabricantes possuíam seus próprios sistemas operacionais.

A Aplicação DroidLar foi desenvolvida para dispositivos com o sistema Android. Essa aplicação é responsável pela comunicação do usuário com o SAR. A comunicação é feita utilizando requisições web através de uma rede IP. Essas requisições usam o protocolo HTTP (ou sua versão segura HTTPS), que implementam um modelo de pedido e resposta. Assim, a comunicação é estabelecida quando um cliente quer enviar uma requisição e é finalizada quando houver uma resposta, por isso a aplicação DroidLar não precisa estar conectado ao SAR permanentemente.

(a) Tela inicial (b) Informações sobre o servidor

Figura 2.2: Interface principal e de configuração do SAR (EUZÉBIO, 2011)

.

2.1.2 Controladores de dispositivos

Os controladores de dispositivos têm a função de executar nos eletroeletrônicos da residência os comandos enviados pelas interfaces de controle nos clientes Android, tais como acender e apagar lâmpadas, fechar cortinas, e possivelmente outras. Normalmente, esses dispositivos são constituídos por microcontroladores e realizam ações de acordo com o comando

[1] http://www.android.com

recebido.

Para o papel de controladores de dispositivos no sistema DroidLar foram utilizados *kits* Arduino[2]. Arduino é uma plataforma aberta desenvolvida em 2005 pela *Interaction Design Institute Ivrea*, e vem sendo muito utilizada para desenvolvimento de protótipos por ser uma ferramenta simples e fácil de usar.

De acordo com Euzébio (2011, p. 37):

> Os *kits* Arduino possuem apenas duas funções: (1) executar efetivamente os comandos recebidos no eletroeletrônico conectado a uma de suas portas; (2) enviar ao SAR uma mensagem contendo o estado dos dispositivos que controla, por exemplo, a quantidade de dispositivos, o tipo de dispositivo e se estão ligados ou não. A primeira é executada toda vez que é recebida uma mensagem do SAR com a requisição do comando, já a segunda, é executada sempre que o dispositivo controlador é ligado e, também, quando é recebida uma mensagem de varredura do SAR.

Os *kits*, mostrados na figura 2.3, se comunicam com o SAR por meio de uma rede ZigBee (ZIGBEE ALLIANCE, 2005), usando um módulo XBee[3]. A arquitetura ZigBee foi criada por um grupo de empresas chamada de *ZigBee Alliance*, e foi construída usando o padrão IEEE 802.15.4 como camada física e de acesso ao meio sem-fio. Esse padrão provê taxas de transmissão de até 250 kbps a distância de algumas dezenas de metros, enfatizando baixo consumo de energia. Além disso, IEEE 802.15.4 usa uma versão simplificada do protocolo *Carrier Sense Multiple Access With Collision Avoidance* (CSMA/CA) para o controle de acesso ao meio, incluindo um modo de operação com provisionamento de banda. Por fim, oferece um mecanismo para encriptação dos dados transmitidos, conferindo assim um grau de segurança às transmissões.

2.1.3 Servidor de Automação Residencial (SAR)

O Servidor de Automação Residencial (SAR) é o elemento central do sistema DroidLar. Ele é responsável por realizar a ponte entre o cliente Android, que está na rede IP, e os controladores de dispositivos, que estão na rede ZigBee. O SAR deve estar sempre ativo, pois, além de gerenciar as mensagens trocadas entre os membros da rede, também é responsável por executar tarefas agendadas pelo usuário, tais como acender ou apagar lâmpadas em instantes determinados.

O SAR é um *software* que foi originalmente desenvolvido em Java, sendo executado em um

[2]http://www.arduino.cc/
[3]http://www.digi.com

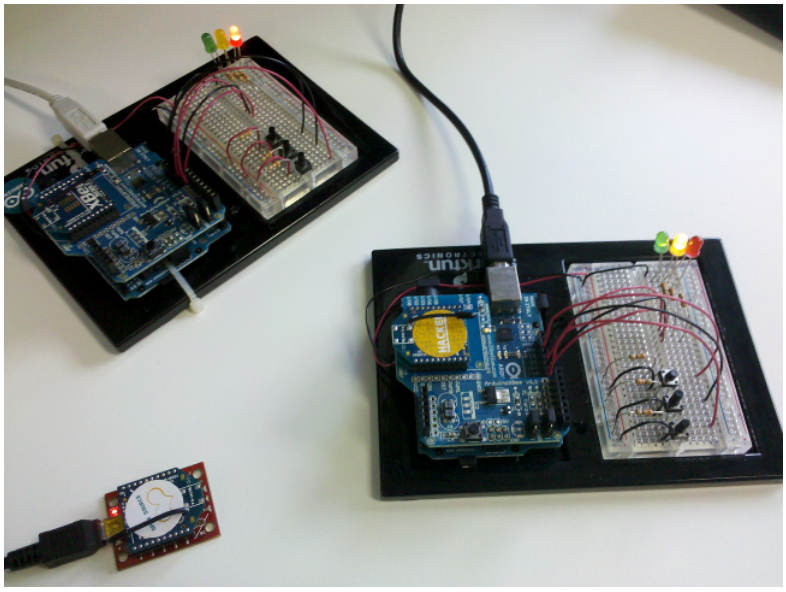

Figura 2.3: Protótipos do DroidLar (EUZÉBIO, 2011)

computador pessoal. Para a comunicação com os controladores de dispositivos através da rede ZigBee foi utilizada o xbee-api[4]. O programador utiliza essa API para a transmissão e recepção dos dados através de um módulo XBee inserido na porta USB do computador.

A comunicação do cliente Android com o SAR através das mensagens *web* é realizada utilizando um Java *Servlet Container* no servidor do sistema. Java *Servlet Container* são programas utilizados em aplicações Java para terem acesso às requisições HTTP recebidas pelo servidor *web*. Diferentemente dos serviços *web*, eles consistem em classes Java instanciadas uma única vez, o que os permitem memorizar as variáveis dos seus códigos e mantê-las durante todo o tempo de execução.

2.2 Sistemas embarcados

Os sistemas embarcados segundo Heath (2002, p. 2) é um sistema baseado em microprocessador, que é construído para controlar uma função ou conjunto de funções e não foi concebido para ser programada pelo usuário final, da mesma forma que um computador é. Um sistema embarcado é projetado para executar uma determinada tarefa, embora com escolhas e opções diferentes. O último ponto é importante porque diferencia-se do mundo do computador pessoal onde o usuário final faz reprogramá-lo sempre que um pacote de software diferente é comprado e executado.

Entre os sistemas embarcados exitem aqueles que são capazes de rodar *firmwares* baseados

[4]http://code.google.com/p/xbee-api

no *kernel* Linux como por exemplo os roteadores e pontos de acesso IEEE 802.11. Há também sistemas que são pequenos computadores de baixo custo, como o Beaglebone (BEAGLEBONE, 2011) e o RaspberryPi[5]. Esses dispositivos possuem características que são limitados para o SAR na versão Java como pouca capacidade de armazenamento, quantidade de memória RAM e pouco poder de processamento. No ponto de vista do sistema DroidLar as características desses dispositivos são necessárias para serem implementados pois são pequenos, não necessitam de grande quantidade de energia para funcionarem e possuem um custo financeiro baixo comparados aos computadores pessoais.

2.2.1 Roteador sem fio TP-Link TL-WR941ND

O TL-WR941ND (TP-LINK, 2012) é um dispositivo de conexão de rede sem fio projetado especificamente para pequenas empresas, escritórios e cenários domésticos. Ele fornece um maneira simples, agradável e muito rápido para acessar a internet para trabalho ou diversão. É compatível com o IEEE 802.11 (Draft 2.0) padrão com velocidades de transmissão sem fio de até 300 Mbps. O roteador possui três antenas externas destacáveis omnidirecional que proporciona um melhor desempenho nas taxas de transmissão, estabilidade e na cobertura do sinal.

Atualmente o TL-WR941ND possui quatro versões sendo que temos a terceira. Com uma arquitetura MIPS[6], esse roteador possui um processador Atheros[7] AR9132 de 400 MHz, 32 MB de memória RAM e 4 MB de memória *Flash*.

Figura 2.4: Roteador sem fio TP-Link TL-WR941ND (TP-LINK, 2012).

[5]http://www.raspberrypi.org/
[6]http://www.mips.com/
[7]http://www.atheros.com

2.2.2 *BeagleBone*

BeagleBone (BEAGLEBONE, 2011) é um pequeno dispositivo de baixo custo projetado para ser uma plataforma de prototipação. Todo o seu projeto é livre e utiliza o sistema operacional baseado no kernel Linux o Ångström[8]. O dispositivo possui um processador AM3359, um ARM Cortex-A8 da *Texas Instruments*[9] que roda em até 720 MHz, possui 256 MB de RAM, dois conectores quarenta e seis 46 pinos de expansão, entrada *ethernet*, um *slot microSD* e uma porta *Universal Serial Bus* (USB).

Figura 2.5: BeagleBone (BEAGLEBONE, 2011).

2.3 Sistemas operacionais para roteadores sem fio

2.3.1 OpenWrt

O OpenWrt (OPENWRT, 2012) é desenvolvido para sistemas embarcados, principalmente roteadores residenciais. O seu nome deriva dos roteadores sem fio da Cisco da linha Linksys WRT54G. A sigla WRT é o acrônimo de Wireless Router Technology (Tecnologia de Roteador Sem-Fio), e Open significa aberto.

Após a liberação dos códigos fontes Linux da série de roteadores Linksys WRT54G surgiram um grande número de firmwares modificados para estender as funcionalidades dos roteadores. Esses firmwares possuem 99% de códigos conservados e 1% de funcionalidades adicionadas, e cada projeto tentou seguir um determinado segmento de mercado. Por isso, era difícil encontrar um firmware com a combinação da funcionalidade desejada.

O desenvolvimento do OpenWrt foi diferente da maioria dos firmwares que se baseavam

[8]http://www.angstrom-distribution.org/
[9]http://www.ti.com/

nos sistemas Linksys WRT54G, o OpenWrt foi desenvolvido a partir do código fonte do Linux com o intuito de não só ter um simples firmware mas um sistema operacional para podemos gerenciar os dispositivos como se fossem computadores, assim possuindo a maioria dos comandos Linux tradicionais e com um sistema de gerenciamento de pacotes para facilmente carregar vários softwares. Exemplos de pacotes que podem ser instalados no OpenWrt: interpretadores de PHP, python, bash, servidores web (uhttpd, Lighttpd, Apache), banco de dados (sqlite, mysql), servidor CUPS para impressão e até mesmo um servidor Asterisk.

2.3.2 DD-WRT

DD-WRT (DD-WRT, 2012) é um projeto de um sistema operacional de código aberto baseado em Linux para roteadores sem fio, não oficial sob os termos da licença *GNU General Public License* (GPL) e baseados nos designs de chips de referência Broadcom ou Atheros. O nome DD-WRT também tem origem na sigla WRT do roteador WRT54G da Cisco. O significado oficial da sigla DD não foi encontrado.

O objetivo principal do DD-WRT é disponibilizar um sistema operacional que seja fácil de usar e tenha suporte a uma grande quantidade de funcionalidades e plataformas de hardware. Ele é desenvolvido focalizando a velocidade e a estabilidade, também possibilita a adição de novos recursos que não são suportados pelo firmware do fabricante do roteador.

O DD-WRT também possui plataformas para o uso comercial que dependem de licença paga. As versões comerciais possibilitam a configuração de rede mais confiáveis e poderosas.

2.4 *Common Gateway Interface* (CGI)

Common Gateway Interface (CGI, 2011) é uma interface que permite passar parâmetros a um programa alojado no servidor, podendo resultar em páginas dinâmicas. O CGI está no âmbito da internet desde 1993 e foi concebido por discussões de desenvolvedores de software e desenvolvedores *World Wide Web*, são eles: Rob McCool (autor de o servidor *web* NCSA httpd), John Franks (autor do servidor *web* GN), Ari Luotonen (o desenvolvedor do servidor *web* httpd CERN), George Phillips (mantenedor da Universidade de British Columbia).

2.5 Bibliotecas C

As APIs que foram usadas em Java não possuem versões para softwares em C como por exemplo o xbee-api. Para desenvolver o novo Servidor de Automação Residencial (SAR) pesquisou-se algumas bibliotecas que irão satisfazer as nossas necessidades como a comunicação pelas redes ZigBee, comunicação pelas redes IP através da *web* e armazenamento de dados.

2.5.1 Servidor *web*

Appweb

Appweb (APPWEB, 2011) é um servidor *web* compacto e veloz incorporado para a hospedagem de aplicações web e frameworks. Criado e mantido pela empresa *Embedthis web technologies*[10] o Appweb fornece as licenças *open-source* e comercial. O Appweb também é disponibilizada em forma de API que possibilita a criação de *softwares* que implementam as funções de suas bibliotecas. Appweb possui um conjunto de recursos que inclui *Secure Sockets Layer* (SSL), *Transport Layer Security* (TLS), autenticação básica, hospedagem virtual, módulos carregáveis, arquivo de configuração de estilo do servidor Apache [11], PHP, Ejscript, CGI e muito outros.

Libwww

Libwww (LIBWWW, 2012) é uma API altamente modular escrita em C para sistemas Unix e Windows, foi criada para ser utilizada em grandes e pequenas aplicações como navegadores, editores, servidor web, etc. Foi concebida em novembro de 1992 por Tim Berners-Lee com o intuito de demonstrar o potencial da *World Wide Web*. Libwww possui uma boa documentação que acompanha com vários exemplos de seu uso.

GNU libmicrohttpd

GNU libmicrohttpd (LIBMICROHTTPD, 2011) é uma biblioteca C de pequeno porte criado por Christian Grothoff, foi construído para tornar mais fácil a criação de um servidor *web* como parte de outro aplicativo. O GNU libmicrohttpd é um software livre e parte do projeto GNU.

[10] http://embedthis.com/
[11] http://www.apache.org/

Esse projeto foi iniciado porque o autor precisava de uma maneira fácil de adicionar um servidor *web* simultâneo a outros projetos. As alternativas existentes eram as soluções que, segundo Grothoff, possuiam terríveis qualidades.

2.5.2 Gerenciador do módulo XBee

libxbee

Criada por Attie Grande, o libxbee (LIBXBEE, 2011) foi desenvolvida para auxiliar os programadores a utilizar os módulos XBee conectados em portas USB com maior facilidade. Segundo o autor é a única biblioteca C para Linux, possui poucas informações e pouca documentação.

O libxbee foi desenvolvido com uma abordagem orientado a conexão. Inicialmente o programador configura uma interface com um módulo XBee e configura conexões de diferentes tipos de nós remotos. É muito simples para enviar e receber dados de uma determinada ligação.

2.5.3 Banco de dados

SQLite

SQLite (SQLITE, 2012) é uma biblioteca que implementa um banco de dados SQL em códigos de linguagem C, possui seu código livre e pode ser usado para qualquer finalidade como projetos de código aberto, comercial e privado. O SQLite foi criado por Dwayne Richard Hipp[12] com o objetivo de permitir que um programa utilize um banco de dados sem a necessidade de consultar um gerenciador ou requerendo um administrador de banco de dados.

GNU dbm

GNU dbm (GDBM, 2011), mais conhecido como GDBM, foi criado por Philip A. Nelson, Jason Downs e Poznyakoff Sergey. É uma biblioteca com funções que ajudam o programador a criar e a manipular bancos de dados em um programa C. O GDBM é um software livre e parte do projeto GNU.

[12] http://www.hwaci.com/drh/

3 *Servidor de Automação Residencial (SAR)*

O sistema DroidLar descrito na seção 2.1 apresenta um protótipo de um sistema de automação residencial, capaz de controlar eletroeletrônicos de uma residência utilizando celulares com Android como interface de controle. Porém o protótipo apenas foi um ponto de partida para a criação de um sistema de automação tão sofisticado quanto os presentes no mercado.

O objetivo geral desse trabalho é reduzir custos do sistema DroidLar para tornar a sua implentação mais viável em cenários domésticos, implementando o Servidor de Automação Residencial (SAR) em um dispositivo embarcado.

A primeira versão do SAR foi um *software* escrito como uma aplicação para um Java *servlet container*, a ser executado em um computador pessoal. Apesar de funcional, essa versão não se mostrou adequada para ser implantada em um dispositivo de baixo custo e com recursos limitados, como por exemplo um ponto de acesso sem-fio IEEE 802.11. Essa limitação existe porque um Java *servlet container* demanda um poder de processamento e uma quantidade de memória RAM não disponíveis em tais dispositivos.

3.1 Entendendo o SAR

Como mencionado na seção 2.1.3 o SAR é o servidor responsável por fazer a ponte entre o cliente Android que está na rede IP com os controladores de dispositivos que estão na rede ZigBee, além de executar tarefas agendadas pelos usuários.

A comunicação pelas redes IP é realizada utilizando mensagens encapsuladas em requisições do protocolo HTTP ou HTTPS. O cliente Android, além de realizar a interface dos usuários com o sistema, atua como um cliente semelhante aos navegadores *web*, enviando requisições ao servidor *web* com os seus comandos.

Em sua versão atual, o SAR tem acesso às requisições do cliente Android por meio de

um *Servlet Container*[1]. *Servlets* são programas escritos em Java que atendem e processam requisições HTTP segundo uma interface bem definida. Por suportar a tecnologia *Servlet* o servidor GlassFish [2] foi utilizado no projeto DroidLar.

Para a comunicação com os controladores de dispositivos foi implementado no SAR o xbee-api [3]. Essa API permite que um programa acesse os dados ZigBee através de um módulo XBee como mostrado na figura 2.3 (ao lado dos controladores de dispositivos).

Para o SAR saber da existência dos controladores de dispositivos ele envia o comando de varredura pela rede ZigBee, essa ação é tomada quando ele recebe a requisição do cliente Android. O comando de varredura é uma mensagem em *broadcast* e os controladores respondem com os estados dos dispositivos que controlam. O SAR diferencia os controladores através dos seus endereços MAC e diferencia os dispositivos por um número de série.

O SAR grava os dados no disco em arquivos objetos (.obj). Esses dados são informações dos dispositivos como o seu nome dado pelo usuário (ex.: Lâmpada da cozinha), o seu tipo de dispositivo (ex.: lâmpada ou persiana), o endereço ZigBee do seu controlador e as tarefa agendadas.

O agendamento das tarefas é realizado utilizando as classes Timer e TimerTask. Essas classes facilitam para o programador o desenvolvimento de *softwares* que executam tarefas em instantes determinados. Quando o SAR é iniciado ele recupera os dados que foram salvos no disco, entre os dados estão as tarefas que são agendadas para serem executadas no tempo que foi determinado pelo usuário.

Para uma pessoa utilizar o sistema DroidLar deve possuir uma conta registrada no servidor. O SAR possui duas contas padrões, o ”admin”e o ”android”e suas respectivas senhas ”admin”e ”android”, esses usuários são declarados em variáveis do tipo *String* assim como as senhas, sem qualquer mecanismo de segurança. Não é possível adicionar ou remover usuários sem modificar o código fonte do SAR.

3.2 Desenvolvimento do novo SAR

Com a desvantagem do Java para dispositivos com poucos recursos de memória e processamento, foi necessário desenvolver um SAR em outra linguagem de programação que dispensa máquinas virtuais ou interpretadores, portanto, uma linguagem que é compilada diretamente

[1] http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html
[2] http://glassfish.java.net
[3] http://code.google.com/p/xbee-api

para um executável. Por ser uma linguagem simples e uma das mais populares para o desenvolvimento de *softwares*, a linguagem C foi adotada para desenvolver o novo Servidor de Automação Residencial.

O desenvolvimento do novo servidor não foi apenas uma tradução de Java para C, apesar de algumas partes serem baseadas da primeira versão do SAR. A sua estrutura foi modificada para se adaptar ao novo cenário proposto por esse trabalho e algumas funcionalidades foram acrescentadas.

Antes de começar o trabalho definimos dois pontos para nos auxiliarmos durante o desenvolvimento do novo SAR.

- O SAR será desenvolvido para dispositivos que possuem um sistema operacional baseado no núcleo Linux. Apesar de utilizamos o OpenWrt no roteador, queremos desenvolver um SAR flexível onde possar ser embarcado em vários sistemas operacionais.
- Definimos um limite para o tamanho do SAR de 1,4 Megabytes. Esse limite foi estipulado pois o roteador sem fio possui 4 MB de memória *Flash*, com o sistema OpenWrt, o tamanho de memória disponível é apenas 1,5 MB.

Ao longo do desenvolvimento do SAR foi necessário testar as suas funções separadamente e depois com as funções integradas, para isso montamos um cenário para facilitar esse processo. O cenário constitui um emulador de telefone celular, monstrado na figura 3.1, onde será executado o cliente Android. Esse emulador vem integrado ao Software Development Kit[4] (SDK) disponibilizada pela Google.

Para o papel de dispositivos eletroeletrônicos utilizamos LEDs montados em *protoboards* como mostrado na figura 2.3, eles simulam algumas lâmpadas de uma residência.

3.2.1 Primeira proposta de desenvolvimento do novo SAR

A primeira proposta deste trabalho sugeria desenvolver um novo Servidor de Automação Residencial para um cenário específicos com um roteador sem fio como mostrado na figura 3.2.

A princípio o roteador TP-Link TL-WR941ND foi utilizado para embarcar o SAR. O *firmware* original que acompanhava esse roteador não permitia executar *softwares* além daqueles desenvolvidos pelo fabricante TP-Link. Para desenvolvemos o SAR primeiramente

[4]http://developer.android.com/sdk/index.html

Figura 3.1: Emulador de telefone celular com Android.

deveríamos trocar o firmware original por um que fosse muito mais flexível em termos de configurações e execução de *softwares*.

Na seção 2.3 é apresentados dois sistema operacionais mais usados para roteadores sem fio, o OpenWrt e o DD-WRT. Para a escolha do sistema consideramos os seguintes requisitos:

- Um grande número de roteadores devem suportar o sistema.
- O sistema deverá conter uma grande comunidade e, portanto, uma vasta documentação.

A tabela de roteadores apresentada no *site* do OpenWrt é superior à tabela do DD-WRT. A comunidade do OpenWrt também é superior, portanto o sistema operacional escolhido para o roteador foi o OpenWrt.

A linguagem de programação que utilizamos para desenvolver o novo SAR foi a linguagem de programação C. Ela foi escolhida pelos seguintes motivos:

- A linguagem não necessita de um interpretador, assim economizando em memória.
- O código passará por uma compilação cruzada para a arquitetura do roteador, tornado o programa mais eficiente em termos de memória e processamento.

O sistema operacional OpenWrt que instalamos no roteador vem com o servidor *web* uhttpd para configurar o roteador através de navegadores *web*. O uhttpd[5] (micro httpd) é um servidor HTTP muito pequeno com cerca de 200 linhas de código. Ele realiza todas as funções básicas de um servidor web mas não suporta um tráfego grande. Por ser simples, ele apresenta alguns problemas como baixo desempenho, poucos recursos e apresenta algumas vulnerabilidades. É bastante usado em dispositivos que possui sua configuração por uma interface web.

Para haver comunicação com o cliente Android pelas das redes IP é necessário que o SAR tenha acesso as requisições do cliente através de um servidor *web*. O servidor *uhttpd* suporta a tecnologia CGI descrito na seção 2.4. Com o CGI o uhttpd iria receber a requisições do cliente Android e passá-las para o SAR processar.

A comunicação do SAR com os Controladores de dispositivos seria realizado por um *gateway* que se comunica com as redes IP e ZigBee. O *gateway* seria um kit Arduino que possuísse um *shield* XBee e um *shield* Ethernet. Esse *gateway* seria necessário no novo cenário do sistema DroidLar pois o roteador sem fio não possui uma porta USB para inserir um módulo XBee. Portanto, o SAR ficaria hospedado em um roteador sem fio onde se comunica somente pelas redes IP, para enviar os comandos para os controladores de dispositivos, o SAR enviaria para o *gateway* que por sua vez enviaria paras os controladores.

O cenário com o roteador sem fio possui dois grandes problemas. Primeiro, a tecnologia CGI permite que um servidor *web* passe requisições para um programa, no nosso caso o SAR, mas o SAR somente seria executado quando um requisição chegasse ao servidor, o cenário do sistema DroidLar requer que o SAR esteja executando o tempo todo para assim executar tarefas agendadas. Segundo problema, o Arduino que utilizamos em nosso projeto não suporta o uso de um *shield* XBee e um *shield* Ethernet, ou usamos um ou o outro. Por esses dois motivos não seguimos o desenvolvimento de um novo SAR com o cenário da figura 3.2.

3.2.2 Segunda proposta de desenvolvimento do novo SAR

Com o deficiência do cenário com o roteador sem fio tivemos que redefinimos-lo. No lugar do roteador vamos utilizar um dispositivos que possui pelo menos um porta USB para o módulo XBee. Com essa mudança o SAR não precisará de um *gateway* para se comunicar pelas rede ZigBee. A estrutura interna do SAR passará por modificações para torná-lo mais compacto e mais completo em relação ao SAR na versão em Java.

Primeira mudança foi em relação ao servidor *web*. Em uma automação residencial onde

[5]http://wiki.openwrt.org/doc/uci/uhttpd

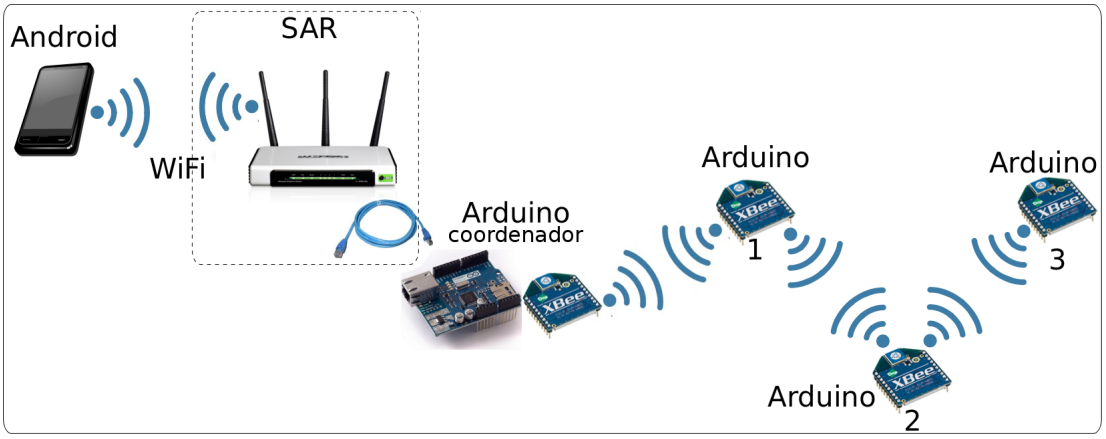

Figura 3.2: Proposta para o novo sistema DroidLar.

possui pouco tráfego de dados não é necessário ter um servidor *web* robusto que suporta um tráfego pesado, mas um que seja leve e atenda um certo número de conexões. Para economizar o espaço de armazenamento do dispositivo embarcado, um servidor *web* foi desenvolvido para atender apenas as requisições dos clientes Android. Esse servidor será integrado aos códigos do SAR.

Outra modificação é em relação ao modo que o SAR grava os dados no dispositivo de armazenamento. A versão do SAR em Java apenas grava os dados em arquivos objetos. Para facilitar esse processo de armazenamento utilizamos um banco de dados que proporciona agilidade para os programadores.

O gerenciador do módulo XBee da nova versão do SAR não teve grandes mudanças em relação à versão anterior, a lógica é semelhante e a diferença esta na sintaxe.

Para desenvolver o servidor *web*, o banco de dados e o gerenciador do módulo XBee utilizamos bibliotecas que ajudam a implementar essas funcionalidades. Foi realizado um estudo para decidir quais bibliotecas iriam ser utilizadas no SAR. Os quesitos considerados na seleção das bibliotecas eram:

- Documentação - a biblioteca deveria possuir uma documentação robusta.
- Exemplos - como o tempo para o desenvolvimento do SAR era curto, desejamos que as bibliotecas acompanhassem com exemplos para o entendimento rápido de suas funcionalidades.
- Atualização - as bibliotecas deveriam ser atualizadas com uma certa frequência pelos seus desenvolvedores.

- Tamanho - Como desenvolvemos o SAR para rodar em dispositivos embarcado é necessário que as bibliotecas sejam pequenas.

O estudo também se dividiu em três categorias sendo que elas estão listadas em ordem de implementação.

- Servidor WEB - Existem várias soluções de bibliotecas que implementam um servidor WEB, mas por existirem várias tivemos que selecionar três para um estudo um pouco mais aprofundado.
- Gerenciador do módulo Xbee - A pesquisa realizada para essa categoria obteve apenas uma solução, que será apresentada a seguir.
- Banco de dados - Para essa categoria obtivemos duas soluções.

Na categoria servidor *web* tivemos as bibliotecas apresentadas na seção 2.5.1, são elas a biblioteca Appweb, Libwww e libmicrohttpd. Libwww é uma ótima solução mas como não recebe atualizações constantes e ainda é a mais pesada essa biblioteca foi descartada. As bibliotecas Appweb e libmicrohttpd são parecidas em termos de documentação, quantidade de exemplos e atualizações, mas no requisito de tamanho a biblioteca Appweb deixa a desejar. Por fazer parte do projeto GNU, ser atualizado constantemente e ser uma biblioteca leve o libmicrohttpd foi utilizado no SAR para a implementação do servidor *web*.

	Appweb	**Libwww**	**GNU libmicrohttpd**
Documentação	Ótima	Ótima	Ótima
Exemplos	Grande quantidade	Grande quantidade	Grande quantidade
Atualizações	Constante	Sem atualizações desde 2002	Constante
Tamanho	2.4 MB	$>$ 4 MB	320,5 KB

Tabela 3.1: Comparação das características das bibliotecas - Servidores *web*.

A variedade de exemplos acompanhados com a biblioteca libmicrohttpd facilitou o desenvolvimento do servidor *web*. Basicamente, o desenvolvimento foi uma adaptação dos exemplos. O pouco de dificuldade que tivemos aqui foi o entendimento da dinâmica entre as funções da biblioteca, mas que foi resolvido com a leitura de sua documentação que é clara e objetiva.

As informações do cliente Android destinadas para o servidor *web* tinham que seguir um caminho pela estrutura interna do SAR. Assim como na versão anterior do SAR, essa versão também deverá encaminhar as mensagens e adaptá-las para outras funções. As funções responsável por fazer o encaminhamento e adaptação das mensagens definem um conjunto que nomeamos de **Processador das mensagens *web***.

O Gerenciador do módulo Xbee é um elemento essencial para a automação, ele é responsável pela comunicação do SAR com os controladores de dispositivos através da rede ZigBee. Mas a única solução encontrada para controlar um módulo Xbee em um programa em C foi o libxbee. Não só a nossa pesquisa retornou um resultado como o próprio autor do libxbee afirma que essa é a única biblioteca C existente para essa finalidade.

	libxbee
Documentação	Pouca documentação
Exemplos	Poucos exemplos
Atualizações	Constante
Tamanho	403,2 KB

Tabela 3.2: Características do libxbee.

O desenvolvimento do gerenciador do módulo XBee foi o mais trabalhoso. Isso se deve ao fato da documentação do libxbee não ser rica em informações. Para desenvolver o gerenciador, além de testar os poucos exemplos que vem acompanhado com libxbee, tivemos que estudar o código fonte da biblioteca para entender a interação das suas funções e estruturas.

Assim como as mensagens *web*, as mensagens ZigBee também precisam ser formatadas para serem transmitidas pela rede ou pela estrutura interna do SAR. O conjunto de funções envolvidas por essas mensagens é nomeada **Processador das mensagens ZigBee**.

Para guarda informações como nomes de usuários e suas respectivas senhas ou os nomes dos eletroeletrônicos da residência foi necessário utilizar um banco de dados. Na nossa pesquisa encontramos duas bibliotecas mais usadas para implementar um gerenciador de banco de dados em C, o SQLite e o GNU dbm. As duas soluções são ótimas mas o GNU dbm é mais leve e, portanto, atende mais as nossas exigências.

	SQLite	**GNU dbm**
Documentação	Ótima	Ótima
Exemplos	Grande quantidade	Grande quantidade
Atualizações	Constante	Última atualização 12/11/11
Tamanho	5,2 MB	232,2 KB

Tabela 3.3: Comparação das características das bibliotecas - Banco de dados.

A implementação do banco de dados com o GDBM foi simples. Tivemos que desenvolver apenas três funções, um para a gravação, um para a leitura o outra função para a exclusão de dados.

Para cada chave, ou identificador de dados, o GDBM pode guardar apenas uma *string* como informação, independentemente do tamanho. No SAR é necessário guardar informações conti-

das em estruturas, ou *structs*, que possuem mais de uma informação. A solução foi concatenar todas as informações em uma *string*, entre as informações eram concatenados caracteres delimitadores como a porcentagem (%) ou o sustenido (#). A recuperação de dados é realizada separando os dados de acordo com os caracteres delimitadores. O grupo de funções responsáveis por tratar os dados para serem gravados ou recuperados do banco de dados é denominado **Processador dos dados**.

O SAR possui a funcionalidade de executar tarefas em momentos definidos pelos usuários. Na versão do SAR em Java foi utilizadas as classes Timer e TimerTask. Na nossa pesquisa não encontramos bibliotecas C que possuíssem a função de agendar tarefas. A solução foi utilizar as bibliotecas time e POSIX *thread* em conjunto.

A biblioteca *time* (TIME, 2012) é uma biblioteca padrão da linguagem C. É utilizada em programas para a manipulação de horários e datas. Ela também fornecem suporte para conversão de tempo de aquisição, entre os formatos de data e saída formatada para *strings*.

A biblioteca POSIX *threads* (PTHREAD, 2012), mais conhecida como *pthread*, é uma API de *threads* baseada em C/C++. Ela permite gerar um novo fluxo de execução de um processo sendo que cada fluxo é concorrente. A utilização de *threads* é mais eficaz em sistemas que possuem vários processadores, onde o fluxo do processo pode ser programado para executar em outro processador ganhando assim velocidade através de processamento paralelo ou distribuído.

Ao chegar uma solicitação para agendar uma tarefa no SAR, os horários e dias eram ajustados pela biblioteca *time*. Em seguida era realizado um cálculo para determinar a diferença entre o tempo estipulado para executar a tarefa e o momento presente em que o pedido de agendamento era entregue ao SAR. Caso o tempo para executar a tarefa é um momento no futuro, uma *thread* era disparada com um temporizador, que realizava uma contagem regressiva até chegar o momento da execução da tarefa. As funções envolvidas com o agendamento e execução de tarefas são denominadas **Serviços de perfis**.

Uma melhoria realizada no SAR foi a parte relacionada aos usuários. Os *logins* dos usuários e suas senhas não são mais declaradas dentro do código fonte, como era na versão anterior do SAR, agora os *logins* são gravados em um banco de dados. No caso das senhas, são gravados seus hash criptográficos MD5, por meio da biblioteca crypt (CRYPT, 2012). As funções que são responsáveis por autenticação de usuário, criação e remoção de contas são denominadas **Serviços de usuários**.

As funções que compõem os **Serviços de controladores e dispositivos** são responsáveis por organizar os dispositivos eletroeletrônicos de acordo com o seu controlador e também ligar

o endereço ZigBee do controlador e número de sequência ao nome dado pelo usuário (ex.: Endereço 00:13:A2:00:40:5D:E9:79, nº sequência 0 ≡ Lâmpada da cozinha).

3.3 Arquitetura

Com o desenvolvimento do novo SAR e a mudança de sua estrutura em relação à versão anterior a arquitetura resultante foi a que está apresentada na figura 3.3. Ela possui duas camadas onde cada uma é responsável por um grupo de funções.

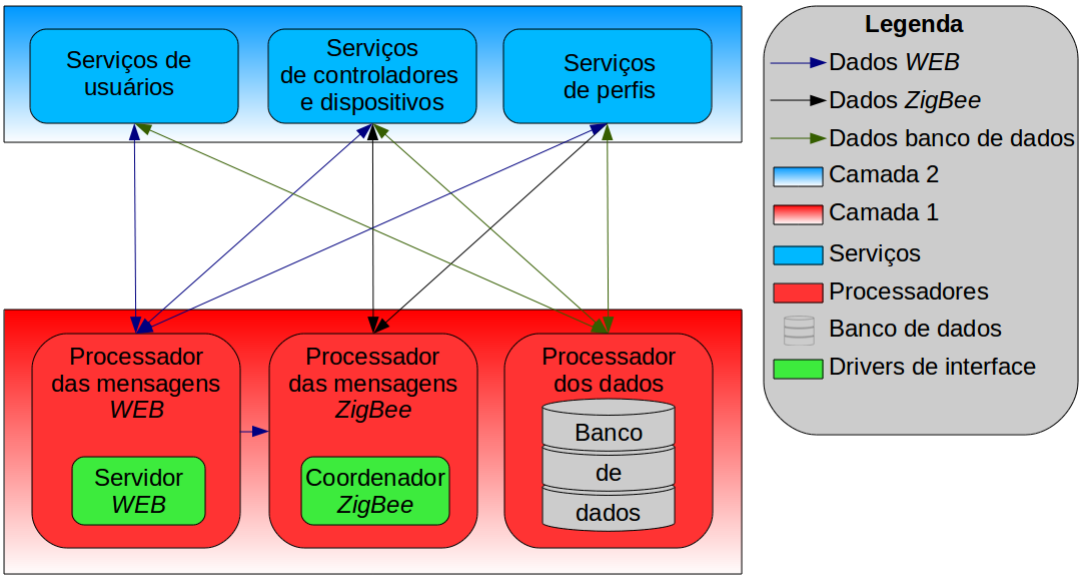

Figura 3.3: Arquitetura do SAR.

A camada 1 é responsável pelas operações básicas com os dados como transmitir, receber, grava, ler e processar. A transmissão e a recepção são realizadas pelos *Drivers* de interface representados por um servidor *web* e pelo coordenador ZigBee. A gravação e a leitura de dados são realizadas pelo banco de dados GDBM. Os processadores são responsáveis por processar ou interpretar os dados e passá-los para um dos servidores da camada superior, transmiti-las por um dos *Drivers* de interface ou gravar em um banco de dados.

1. **Processador das mensagens *web*** - Possuindo um servidor *web* integrado à sua estrutura, sua função é fazer uma leitura dos dados provindos do cliente Android e passá-los para um dos serviços da camada superior ou para o **Processador das mensagens ZigBee**. Além da leitura esse processador verifica se os dados recebidos são de um usuário que já esteja logado no sistema.

2. **Processador das mensagens ZigBee** - Este processa os dados para a transmissão na rede ZigBee e para o envio dos dados para o **Serviços de controladores e dispositivos**.

3. **Processador dos dados** - Esse processador realiza ajustes nos dados para serem concatenadas em uma *string*. Esse passo é necessário pois o GDBM guarda uma chave, que é um identificador de cada dado, e uma *string* que é o dado. Para a recuperar os dados, o Processador dos dados separa a *string* por caracteres separadores como a porcentagem (%) ou o sustenido (#).

Na camada 2 estão os serviços do SAR responsáveis pelas tarefas mais complexas do sistema. Entre esses serviços oferecidos estão os **Serviços de usuários**, **Serviços de controladores e dispositivos** e **Serviços de perfis**.

1. **Serviços de usuários** - São realizadas as operações de autenticação dos usuários, criação e exclusão de contas.

2. **Serviços de controladores e dispositivos** - É responsável por organizar os dispositivos eletroeletrônicos de acordo com o seu controlador. Também liga o endereço ZigBee do controlador e número de sequência ao nome dado pelo usuário.

3. **Serviços de perfis** - São os serviços de agendamento e execução de tarefas determinadas pelos usuários, por exemplo acender uma determinada lâmpada às 19 horas todos domingos.

3.4 Implementação do SAR em um sistema embarcado

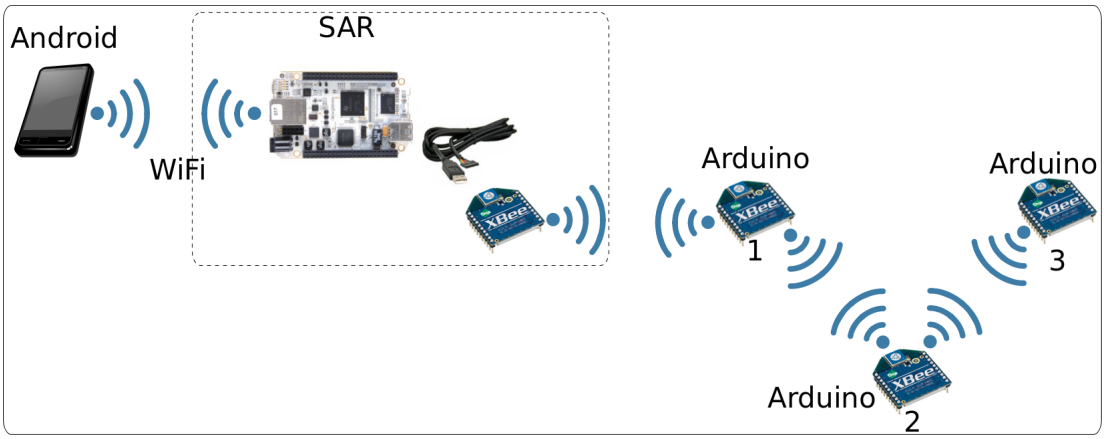

Figura 3.4: Novo cenário do sistema DroidLar.

Para testar o novo SAR em um sistema embarcado utilizamos o *BeagleBone* apresentado na seção 2.2.2. O *BeagleBone* possui seu projeto livre e um sistema robusto projetado para rodar diversas aplicações. Apesar de utilizamos um dispositivo embarcado específico o SAR poderá ser executado em dispositivos que possuem um sistema operacional baseado no *kernel* Linux, possuem uma porta ethernet para a comunicação com o cliente Android, e um porta USB para a comunicação com os controladores de dispositivos.

Com o SAR embarcado no *BeagleBone* obtivemos o cenário da figura 3.4. Um cenário com custos financeiro e energético reduzidos, comparado ao cenário com o SAR rodando em um computador pessoal.

4 Conclusões

O sistema de automação residencial DroidLar permite que um aplicativo rodando em celulares Android controle os eletroeletrônicos de uma residência e também possibilita o uso de tarefas agendadas com a utilização de perfis. A primeira versão do Servidor de Automação Residencial (SAR) é um *software* escrito em Java, apesar de funcional não é indicado para a implantação em sistemas embarcados.

Neste trabalho foi apresentado um novo Servidor de Automação Residencial (SAR) para o sistema DroidLar. Desenvolvido na linguagem de programação C, o novo SAR veio para suprir a necessidade de reduzir custos do sistema de automação implementando-o em um dispositivo de baixo custo econômico e energético.

Com o trabalho desenvolvido o SAR poderá ser embarcado em vários dispositivos que possuem um sistema baseado no *kernel* Linux, uma porta ethernet, uma porta USB e aproximadamente 1,22 Megabytes disponível na memória de armazenamento. O SAR possui o tamanho de 292,8 Kilobytes, as bibliotecas libmicrohttpd, libxbee e o GDBM juntas possuem 955,9 Kilobytes. O tamanho total do SAR com as bibliotecas é 1248,7 Kilobytes, 1,22 Megabytes, assim ficando abaixo do limite estipulado de 1,4 Megabytes.

O desenvolvimento do SAR foi dificultada pela manipulação da biblioteca libxbee. Ela é responsável pela conexão ZigBee, realizada entre o SAR e os controladores de dispositivos. A falta de documentação dificultou e atrasou o desenvolvimento do SAR porque foi necessário estudar e entender a biblioteca.

Outra dificuldade que encontramos neste trabalho foi desenvolver o SAR tão eficaz quanto o SAR na versão em Java e pequeno o suficiente para implementá-lo nos sistemas embarcados. Em alguns testes durante o desenvolvimento o SAR ser tornava o gargalo do sistema, ou seja consumia a processador do dispositivo. Tivemos que tomar cuidado para o SAR não consumir o processador.

Apesar das dificuldades encontradas no desenvolvimento o novo SAR é mais eficaz e trás mais economia para o sistema DroiLar do que a versão em Java. Portanto os objetivos propostos

neste trabalho foram realizados.

4.1 Trabalhos futuros

O sistema DroidLar é um sistema de automação novo onde necessita de adaptações e inserção de novos recursos, existindo então, uma infinidade de trabalhos a serem feitos. Atualmente o DroidLar apenas suporta lâmpadas. Um dos trabalhos futuros seria a modificação do sistema para suportar mais dispositivos eletroeletrônicos, tornando o sistema mais completo.

Outra possibilidade seria desenvolver um sistema de gerenciamento do DroidLar onde um usuário poderia configurar e gerenciar o sistema DroidLar através de uma interface *web*. Também seria interessante a proposta de um sistema de monitoração por câmeras, tornando o DroidLar em um sistema de automação residencial e um sistema de segurança.

Referências Bibliográficas

APPWEB: Appweb embedded web server. 2011. Disponível em: <http://appwebserver.org/>. Acesso em: 15 mar. 2012.

BEAGLEBONE: Beagleboard.org – bone. 2011. Disponível em: <http://beagleboard.org/bone>. Acesso em: 09 mai. 2012.

CGI: Common gateway interface. 2011. Disponível em: <http://www.w3.org/CGI/>. Acesso em: 20 jan. 2012.

CRYPT: crypt - the gnu c library. 2012. Disponível em: <http://www.gnu.org/software/libc/manual/html_node/crypt.html>. Acesso em: 19 jun. 2012.

DD-WRT. 2012. Disponível em: <http://www.dd-wrt.com>. Acesso em: 17 jan. 2012.

EUZÉBIO, M. V. M. *DroidLar - Automação residencial através de um celular Android*. Monografia (Graduação em Sistemas de Telecomunicações) — Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, São José, 2011. Disponível em: <http://wiki.sj.ifsc.edu.br/wiki/images/b/b7/TCC_MichelEusebioMello.pdf>.

GDBM: Gnu dbm. 2011. Disponível em: <http://www.gnu.org.ua/software/gdbm/>. Acesso em: 12 jun. 2012.

HEATH, S. *Embedded Systems Design*. Elsevier Science, 2002. ISBN 9780080477565. Disponível em: <http://books.google.com.br/books?id=BjNZXwH7HlkC>.

LIBMICROHTTPD: Gnu libmicrohttpd: a library for creating an embedded http server. 2011. Disponível em: <http://www.gnu.org/software/libmicrohttpd/>. Acesso em: 21 mar. 2012.

LIBWWW: Libwww - the w3c sample code library. 2012. Disponível em: <http://appwebserver.org/>. Acesso em: 26 mar. 2012.

LIBXBEE: attie.co.uk – libxbee. 2011. Disponível em: <http://attie.co.uk/libxbee>. Acesso em: 23 abr. 2012.

OPENWRT. 2012. Disponível em: <http://wiki.openwrt.org/>. Acesso em: 13 jan. 2012.

PTHREAD: Posix threads programming. 2012. Disponível em: <https://computing.llnl.gov/tutorials/pthreads/>. Acesso em: 5 jun. 2012.

SQLITE: Sqlite. 2012. Disponível em: <http://www.sqlite.org/>. Acesso em: 25 mai. 2012.

TIME: time - gnu project - free software foundation (fsf). 2012. Disponível em: <http://www.gnu.org/software/time/>. Acesso em: 11 jun. 2012.

TP-LINK: Tl-wr941nd. 2012. Disponível em: <http://www.tp-link.com.br/products/details/?categoryid=&model=TL-WR941ND>. Acesso em: 16 jan. 2012.