UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

PROTÓTIPO DE SISTEMA PARA AUTOMATIZAÇÃO RESIDENCIAL CONTROLADO POR COMANDO DE VOZ

RONALDO ROTHER

BLUMENAU
2012

2012/2-22

RONALDO ROTHER

PROTÓTIPO DE SISTEMA PARA AUTOMATIZAÇÃO RESIDENCIAL CONTROLADO POR COMANDO DE VOZ

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Francisco Adell Péricas, Mestrado - Orientador

BLUMENAU
2012

2012/2-22

PROTÓTIPO DE SISTEMA PARA AUTOMATIZAÇÃO RESIDENCIAL CONTROLADO POR COMANDO DE VOZ

Por

RONALDO ROTHER

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Mestre – Orientador, FURB
Membro:	Prof. Miguel Alexandre Wisintainer, Mestre – FURB
Membro:	Prof. Roberto Heinzle, Mestre – FURB

Blumenau, 29 de novembro de 2012.

Dedico este trabalho a minha família, amigos e colaboradores, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À minha família, pelo suporte e compreensão em todas as horas difíceis.

Aos meus amigos e colegas, principalmente os que acompanharam este projeto e vivenciaram desafios semelhantes.

Ao meu orientador, Francisco Adell Péricas, por ter acreditado na conclusão deste trabalho.

Aos professores do Departamento de Sistemas e Computação da Universidade Regional de Blumenau por suas contribuições durante os semestres letivos.

Não repete as táticas que te fizeram vencer, deixa que teus métodos sejam regulados pela infinita variedade de circunstâncias.

Sun Tzu

RESUMO

Este trabalho apresenta um sistema para ambiente *desktop*, *web* e *mobile* voltado para automação residencial, com suporte a comando por voz. O sistema foi desenvolvido com o uso das tecnologias Java, JavaServer Faces, Android e HTML5 utilizando a ferramenta Eclipse e o banco de dados MySQL. O *framework* Sphinx foi utilizado para o reconhecimento de fala. A interação com hardware real foi abstraída através de um simulador, porém o sistema oferece suporte à instalação de interfaces de dispositivo reais desenvolvidas para ele.

Palavras-chave: Domótica. Reconhecimento de fala. Dispositivos móveis. *Web*.

ABSTRACT

This work presents a system for desktop, web and mobile environments, oriented to home automation, with support for voice control. The system was developed using the Java, JavaServer Faces, Android and HTML5 using the Eclipse tool and the MySQL database. The Sphinx framework was used for speech recognition. The interaction with real hardware is abstracted through a simulator, but the system supports the installation of real device interfaces developed for it.

Key-words: Domotics. Speech recognition. Mobile devices. Web.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O mercado da automação residencial, também referida como domótica ou “casa inteligente”, emergiu do desejo de um público de minimizar o esforço empregado em tarefas comuns ao ambiente doméstico, impulsionado pela crescente integração da tecnologia ao cotidiano e pelos avanços em automação de dispositivos gerados na indústria.

Segundo Alonso (2011), dentro de algum tempo será comum haver múltiplos mecanismos em uma residência para executar tarefas rotineiras a este tipo de ambiente. Ao introduzir a automatização no contexto doméstico é necessário aproximá-la do aspecto social e de convivência humanos, favorecendo a integração a interação entre o utilizador e o sistema.

O número de dispositivos e tecnologias empregadas ao modelo de negócio residencial cresceu significativamente nos últimos anos, possibilitando o desenvolvimento de produtos e serviços baseados no esforço de integração e centralização de controle (WELLS, 2009). A multiplicidade de mecanismos disponíveis introduz a dificuldade de estabelecer interoperabilidade, desafio alvo de tentativas de padronização dos protocolos de comunicação por algumas soluções comerciais (RILEY, 2012).

1.1 OBJETIVOS DO TRABALHO

O objetivo geral do trabalho é o desenvolvimento de um sistema de automatização residencial controlado por uma interface de reconhecimento de fala.

Os objetivos específicos do trabalho proposto são:

a) disponibilizar interface para reconhecimento de fala;
b) disponibilizar interface *web* para dispositivos móveis (suportada nativamente na plataforma Android);
c) disponibilizar sistema de *feedback* e alertas ao usuário por sintetização de voz;
d) disponibilizar um simulador gráfico ao qual o sistema deve se conectar para demonstração de seu funcionamento.

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo tem-se a introdução ao tema principal deste trabalho com a apresentação da justificativa e dos objetivos.

No segundo capítulo apresenta-se a fundamentação teórica pesquisada sobre automação residencial, reconhecimento de padrões, dispositivos móveis, aplicações *web* e trabalhos correlatos.

O terceiro capítulo apresenta o desenvolvimento do sistema iniciando-se com o levantamento de informações, tendo na sequência a implementação e operacionalidade do sistema.

No quarto capítulo tem-se as conclusões deste trabalho bem como apresentam-se sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos a serem apresentados nas seções a seguir, tais como automação residencial, reconhecimento de fala, reconhecimento do locutor, além de trabalhos correlatos.

2.1 AUTOMAÇÃO RESIDENCIAL

De forma simples, automação residencial pode ser definida como um produto ou serviço que proporcione alguma funcionalidade de ação ou mensagem ao ambiente residencial sem necessidade de intervenção humana direta (RILEY, 2012). O papel da automação residencial não se restringe apenas a atender necessidades de conforto dos residentes, embora este seja o apelo principal do mercado. Por exemplo, é possível obter benefícios da redução de gastos com energia elétrica, por meio do controle do sistema de iluminação, e melhorar a segurança do ambiente, por meio de monitoramento.

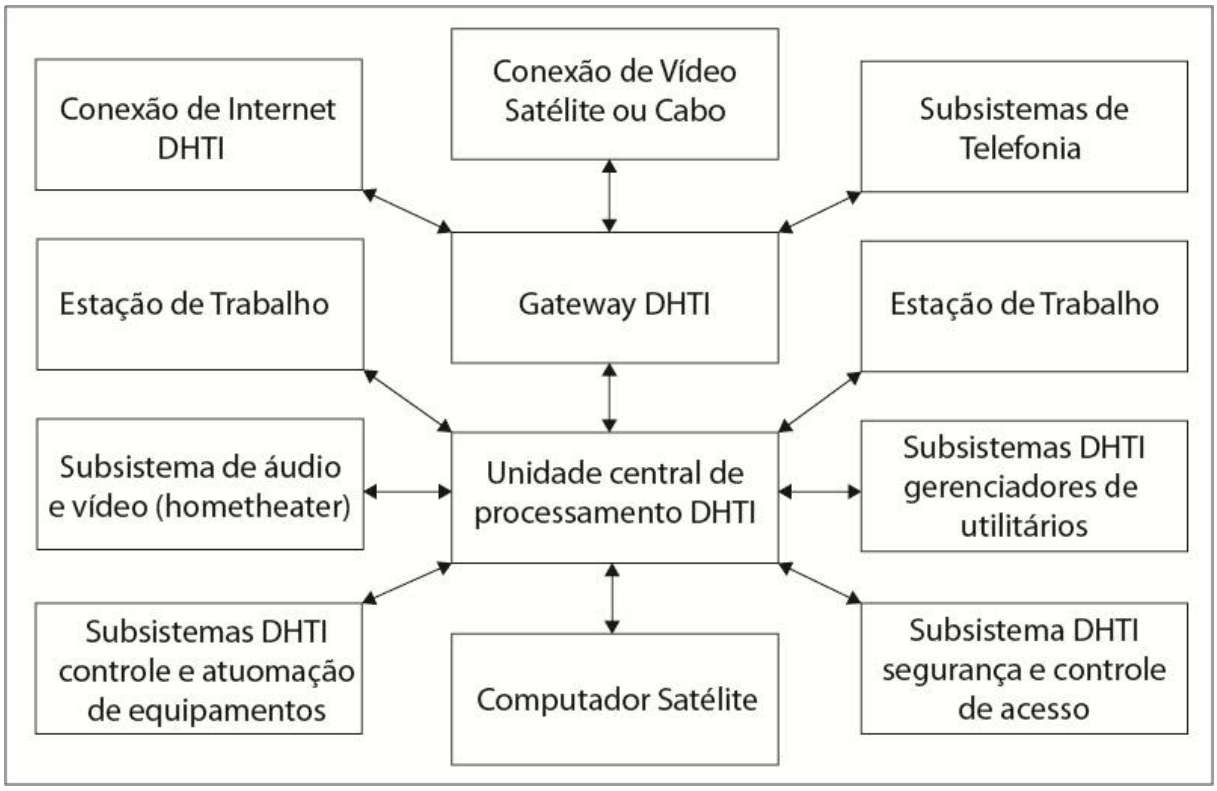
De acordo com Wells (2009), o conceito de *Digital Home Integration Technology* (DHTI) representa a integração, no ambiente residencial, dos subsistemas de distribuição de dados digitais (como audiovisual), equipamentos, utilitários domésticos e outras tecnologias. Alguns exemplos de aplicações são o controle de subsistemas de iluminação, segurança, aparelhagem multimídia e climatização.

2.1.1 Sistema para automação residencial

Segundo Wells (2009), um sistema DHTI fornece acesso a um conjunto de dados e serviços variável, agregando recursos conforme a demanda, e centraliza o gerenciamento e controle dos diversos subsistemas. Ele também pode ser responsabilizado pelo gerenciamento das rotinas de segurança e monitoramento do tráfego de rede, atuando como um *gateway* entre os recursos externos e internos.

A Figura 1 exibe a estrutura de um sistema DHTI de grande proporção, indicando

através das setas o fluxo de dados dentro do sistema e com os sistemas externos.



Fonte: Adaptado de Wells (2009).

Figura 1 – Estrutura de um sistema de automação residencial grande

Segundo Wells (2009), os sistemas de automação residencial em geral são formados conjuntos de componentes, sendo eles:

a) *processadores*, as centrais de processamento de controle do sistema, normalmente computadores conectados aos demais dispositivos e executando os softwares necessários;
b) *links de comunicação*, que permitem o tráfego de dados entre os diversos componentes do sistema e com os recursos externos à rede residencial. Conexões de com ou sem fios são uma alternativa comum, embora existam padrões de comunicação como o X10 e o *HomePlug*, não difundidos amplamente;
c) *sensores*, traduzem eventos externos em dados para o sistema. Sensores permitem que o sistema reaja a eventos e alterações no ambiente;
d) *dispositivos de controle*, dispositivos capazes de receber instruções das centrais de processamento e executar algum tipo de ação ou movimento;
e) *dispositivos de visualização e monitoramento*, telas e monitores que permitem interação com o sistema e a identificação de seu estado e funcionamento;

f) *dispositivos de gravação e armazenamento*, unidades de armazenamento de dados, como discos rígidos e gravadores de DVD.

Um sistema DHTI objetiva fornecer alguns benefícios ao usuário, como economia de tempo, automatização de certas tarefas repetitivas e eventos conhecidos e programáveis, assim como economia de recursos por controlar os sistemas de iluminação, aquecimento e água.

2.2 RECONHECIMENTO DE PADRÃO

Segundo Murty e Devi (2011), reconhecimento de padrões pode ser definido como a classificação de dados baseada em conhecimento ou em informações estatísticas extraídas do conteúdo a ser analisado. Técnicas de reconhecimento de padrões possuem várias aplicações, sendo o reconhecimento biométrico uma das mais amplamente utilizadas e conhecidas.

As técnicas de reconhecimento de padrões dividem-se entre dois paradigmas principais: reconhecimento de padrões estatístico e sintático. O campo estatístico é mais amplamente utilizado e considerado mais apropriado para aplicações que lidam com a análise de dados mais sujeitos a variações e interferências, enquanto a variante sintática é mais utilizada no campo das linguagens formais (MURTY; DEVI, 2011).

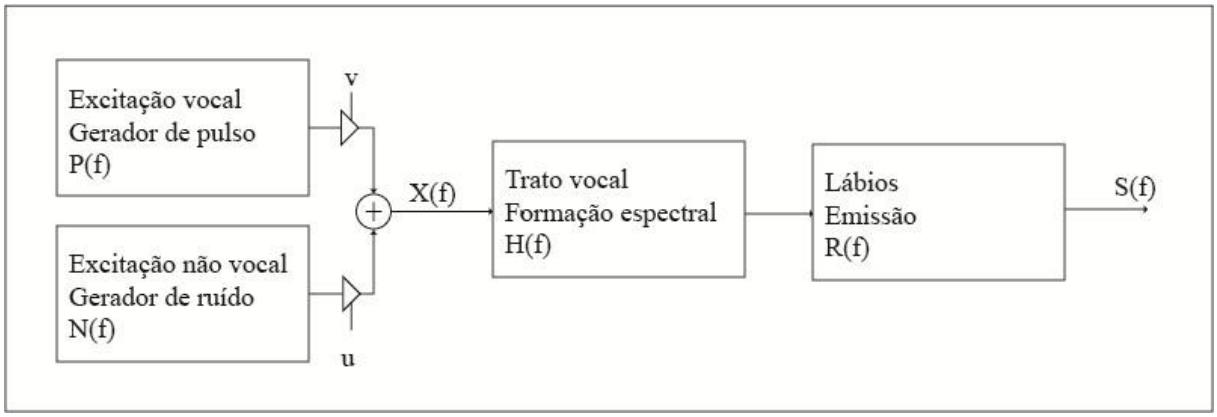
2.2.1 Funcionamento da fala

A diversidade de sons produzidos pela fala humana é originada pelo funcionamento conjunto da glote, língua, dentes, lábios e nariz. Estas estruturas funcionam como um tubo por onde o ar flui a partir dos pulmões, e as alterações em sua forma, como o fechamento parcial dos lábios e a liberação das vias nasais, modifica o espectro do sinal, permitindo moldar os sons de forma a compor a fala em toda a sua variabilidade (PLANNERER, 2005).

Segundo Plannerer (2005), um sinal de fala pode ser excitado de três formas, que podem também ser combinadas para a formação de sinais únicos:

a) *excitação vocalizada*, quando a expulsão de ar dos pulmões força a abertura da glote periodicamente. É usada principalmente na articulação de vogais, e é considerada a frequência fundamental da fala humana;
b) *excitação não vocalizada*, que gera um sinal de ruído quando o flui através de uma estrutura mais estreita. É utilizada na expressão de fonemas fricativos, como */f/* e */s/*;
c) *excitação transiente*, gerada através do aumento da pressão de ar pelo fechamento da glote ou dos lábios e subsequente abertura, diminuindo a pressão do ar e criando um espectro único através desta transição. É utilizada para formação de fonemas explosivos, como */p/*.

A banda do sinal de fala que contém a informação necessária para a compreensão é de 4 kHz, e enquanto a frequência fundamental está fica entre 80 e 350 Hz. A figura abaixo exibe uma representação simplificada do mecanismo de geração da fala humana. A excitação vocal tem o espectro do sinal dado por *P(f)* e a não vocal por *N(f)*. Ambos podem ser ajustados alterando a amplitude de sinal do gerador de pulso *v* e do gerador de ruído *u*. A modelagem do espectro fica por conta de *H(f)* enquanto a emissão dos lábios é dada por *R(f)*.



Fonte: Adaptado de Plannerer (2005).

Figura 2 – Modelo simplificado de geração da fala humana

2.2.2 Reconhecimento de fala

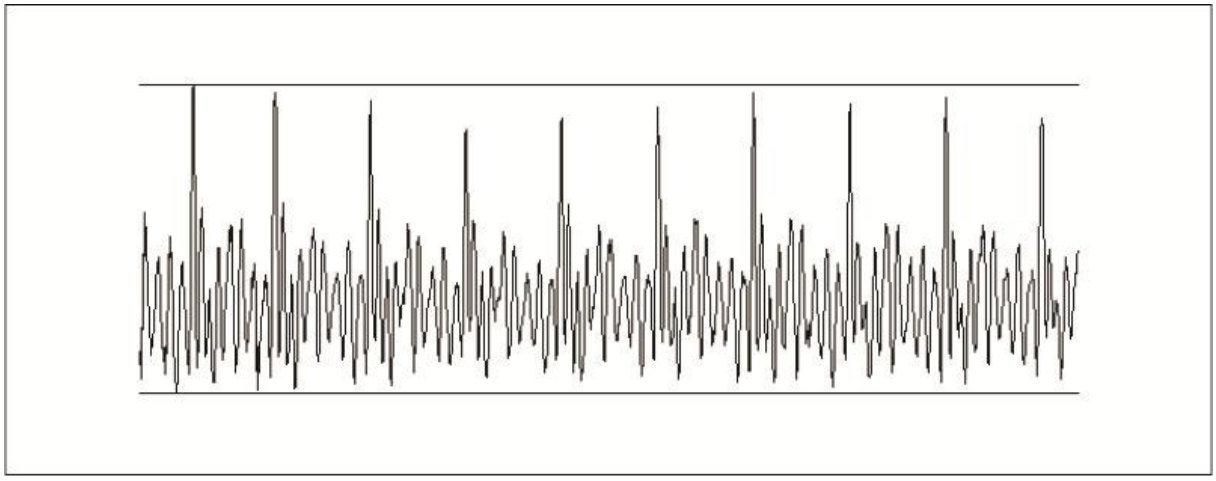
O reconhecimento de fala é uma subdivisão do domínio do reconhecimento de padrões, voltada especificamente para a extração de informações, a partir de uma entrada de

áudio pré-processada, referentes ao conteúdo identificado como semelhante a um sinal produzido por um trato vocal humano. Técnicas de reconhecimento de fala são aplicadas a programas do tipo *Multimedia Document Retrieval* (MDR), que lidam com análise de dados envolvendo áudio (MURTY; DEVI, 2011).

Aplicações do reconhecimento de fala ganharam um aumento de popularidade recentemente, principalmente no tocante aos dispositivos móveis, com o advento do Google Speech (também chamado *Voice Search*) e do Apple Siri. O último apresenta uma proposta mais ousada, sendo divulgado como um assistente pessoal que permite ao usuário expressar-se de forma comum (a princípio, somente no idioma inglês) para acessar as funcionalidades do aparelho, que aprimora sua capacidade de resposta a cada comando processado (APPLE INC, 2012).

No modelo de produção de fala exibido na Figura 2 da seção 2.2.1 podem ser observadas algumas características que devem influenciar na geração do sinal de fala: a mistura entre a excitação vocal e não vocal, a frequência fundamental, a formação do espectro e a amplitude do sinal. De acordo com Plannerer (2005), estes são os quatro parâmetros que é necessário extrair, de um sinal de entrada de áudio durante o processo de processamento acústico.

Os parâmetros do sinal vocal são extraídos a uma taxa de amostragem predefinida (por exemplo, a cada dez milissegundos) e, em uma implementação que opere diretamente sobre o espectro do sinal vocal, encaminhados para o sistema de reconhecimento de fala. Entretanto, o espectro do sinal original possui implicitamente a influência dos pulsos gerados pela excitação vocal, introduzindo picos no espectro, como demonstrado na figura 3. Sistemas de reconhecimento de fala podem, em sua camada de pré-processamento, aplicar filtros de transformação para remover tais características do espectro (PLANNERER, 2005).



Fonte: Adaptado de Plannerer (2005).

Figura 3 – Sinal gerado pela vogal */a:/*, amostra de 100 ms

2.2.3 Reconhecimento de locutor

O conceito reconhecimento de locutor (em inglês, *speaker recognition*) está associado ao reconhecimento de fala pelo fato de serem empregados paralelamente em diversas aplicações. Entretanto, embora sejam baseados em processamento sobre o mesmo tipo de mídia, o reconhecimento de locutor, também chamado de biometria de locutor (*speaker biometrics*), é focado na identificação do indivíduo originador de uma entrada de voz (BEIGI, 2011).

Segundo Beigi (2011), a disciplina de reconhecimento de locutor divide-se em seis diferentes ramificações:

a) *autenticação de locutor*, para confirmação de identidade de um usuário;
b) *identificação de locutor*, para determinação de sua identidade a partir unicamente da entrada de áudio;
c) *classificação de locutor*, para determinar o grupo ao qual o indivíduo pertence, como sexo ou faixa de idade;
d) *segmentação de locutor*, para identificar vozes de diferentes indivíduos presentes simultaneamente em um mesmo sinal de áudio;
e) *detecção de locutor*, para identificar um locutor específico;
f) *rastreamento de locutor*, para identificar diversos indivíduos ao longo de um fluxo de áudio.

2.3 APLICAÇÕES WEB

Nas últimas duas décadas a *web* passou por diversas transformações, desde o princípio da utilização do *Hipertext Markup Language* (HTML) para apresentação de conteúdo estático. Com a expansão do acesso e a presença no dia a dia das pessoas, foram criadas necessidades e oportunidades para tornar a tornar a *web* mais interativa e atrativa, mantendo o princípio do HTML para independência de navegador e plataforma (GOODMAN; MORRISON; NOVITSKI; RAYL, 2010).

Conforme Goodman, Morrison, Novitski e Rayl (2010), novas técnicas e padrões foram introduzidos com a finalidade de melhorar a experiência de navegação e possibilitar um maior nível de interatividade entre a página e o usuário. As seguintes tecnologias podem ser mencionadas como fundamentais:

a) *Cascading Style Sheets (CSS)*, um padrão estruturado pelo *World Wide Web Consortium* (W3C) para a definição de estilo (fontes, cores e posicionamento) de elementos de uma página HTML;
b) *JavaScript*, uma linguagem orientada a objetos, compilada e executada no navegador, que permite a execução de algumas operações, como validações de campos, usando a capacidade de processamento do cliente (e, portanto, liberando o servidor), manipulação da estrutura das páginas sem necessidade de recarregamento e controle da navegação.
c) *Asynchronous JavaScript And XML (AJAX)*, uma técnica de utilização do JavaScript para efetuar requisições ao servidor em segundo plano. Ele permite que uma página seja modificada, parcialmente ou não, sem necessidade de recarregamento ou redirecionamento.
d) programação no lado servidor, possibilitado por tecnologias como o *Java Server Faces* (JSF), Microsoft .NET e *PHP Hypertext Processor* (PHP), permitiram a introdução de conteúdo dinâmico, com acesso a bases de dados e demais recursos do servidor.
e) *plugins*, extensões do navegador fornecidas por terceiros que ampliam sua funcionalidade. Um dos plugins mais utilizados é o Adobe *Flash*, o qual permitiu um nível mais elevado de interatividade com aplicativos embutidos no navegador que podem, por exemplo, executar vídeos, jogos e outros programas interativos.

Com a evolução das tecnologias empregadas na *web*, esta se tornou uma plataforma muito mais adequada para finalidades mais complexas. A introdução das tecnologias do lado servidor e do AJAX tornaram possível aos sites modernos oferecer um conjunto de recursos maior com um nível de usabilidade digno de uma aplicação nativa de *desktop*. A diferença cada vez mais tênue entre algumas páginas e aplicativos de *desktop*, ao exemplo do Google Docs e do Facebook, está cada vez mais tênue.

2.3.1 HTML5

O HTML5 é uma especificação projetada para substituir o HTML 4.01, lançado em 1999, e é ainda considerado em desenvolvimento, embora a maior parte dos browsers atuais, inclusive para dispositivos móveis, suporte suas funcionalidades em algum nível. Ele surgiu do trabalho conjunto do W3C com o *Hypertext Application Technology Working Group* (WHATWG), que também desempenharam papéis chave nas versões anteriores da linguagem (W3SCHOOLS, 2012) .

O HTML5 é uma especificação criada para expandir ainda mais as capacidades da *web*, agregado uma série de ferramentas como a criação de bases de dados locais (de capacidade reduzida), manipulação de arquivos (dentro de limites seguros), operação *offline*, *multithreading* e multimídia.

2.3.2 JavaServer Faces (JSF)

JavaServer Faces é um *framework* de desenvolvimento de aplicações *web* voltado principalmente para a interface gráfica. Sua proposta é voltada para a simplificação do processo de desenvolvimento através uma arquitetura centrada em componentes, independente de cliente, com gerenciamento de estado simplificado ao longo de múltiplas requisições e é fundamentado em padrões de *design* como o *Model View Controller* (MVC).

O JSF foi concebido e tem sua evolução regida pelo *Java Community Process* (JCP), que tem a participação de diversos especialistas e empresas líderes em tecnologia, como a IBM e a Oracle. Cada melhoria é formalizada como uma requisição de especificação, ou *Java*

Specification Request (JSR), que são melhoradas, aprovadas ou reprovadas ao longo do processo.

O JSF é baseado em outras tecnologias, utilizando *Servlets* e *JavaServer Pages (JSP)* para o processamento de requisições e criação de páginas, respectivamente. Além disso, ele agrega muitas melhorias e características que são produto do refinamento proporcionado pelo aprendizado com *frameworks* e tecnologias mais antigas, como o Apache *Struts* e o *Spring* MVC (BURNS; SCHALK, 2010).

2.4 DISPOSITIVOS MÓVEIS

Há poucos anos, os telefones móveis eram limitados a aparelhos portáteis capazes de realizar chamadas telefônicas e, quando muito, forneciam alguma funcionalidade de fábrica, como calendários ou alarmes. Quando seus sistemas operacionais ofereciam alguma extensibilidade, como os adeptos da plataforma Symbian, era necessário codificar em linguagens de baixo nível com bibliotecas específicas para um modelo ou uma série de aparelhos. O maior nível de flexibilidade no que se refere a multiplataforma foi atingido com os MIDlets Java, que, entretanto, sofriam com acesso restrito ao hardware (MEIER, 2010).

Segundo Meier (2010), novas plataformas de software estão surgindo, juntamente com a crescente evolução do hardware, têm provido ambientes mais ricos e flexíveis para o desenvolvimento de aplicativos.

2.4.1 A Plataforma Android

De acordo com Meier (2010), a maior parte das plataformas mais recentes, como o Windows Mobile e o IOS, estão sujeitas a algumas limitações, possivelmente em razão de serem baseadas em tecnologia proprietária, como a priorização de aplicações nativas, restrição de comunicação entre aplicativos e restrição da distribuição de aplicativos de terceiros. A plataforma Android, desenvolvida pela Google, surgiu como uma alternativa para contornar estas limitações.

Baseado em um *kernel* Linux, o sistema oferece acesso a uma grande diversidade de hardware e abstrai dos desenvolvedores de aplicativos a maior parte das especificidades da plataforma. Como os programas são executados sobre uma máquina virtual (chamada *Dalvik*), a responsabilidade sobre a compatibilidade das *Application Programming Interfaces* (APIs) com o o sistema operacional subjacente (MEIER, 2010).

2.5 TRABALHOS CORRELATOS

Gadotti (2010) desenvolveu uma solução de automação residencial controlada pelo Twitter. A proposta principal do trabalho desenvolvido é o monitoramento de mensagens de um determinado usuário do serviço via WebService, dos quais são extraídos os comandos. A solução foi implementada sobre um hardware *Freakin´ easy* (FEZ), sem necessidade de um computador dedicado.

Venturi (2005) produziu um protótipo de sistema para controle e monitoração residencial controlado por dispositivos móveis empregando a plataforma .NET, utilizando WebServices SOAP. Para finalidades de prova do sistema, foi utilizado um simulador de circuito de ambiente residencial conectado através de uma porta paralela ao servidor.

Censi (2001) desenvolveu um sistema baseado em controle por meio de correio eletrônico, utilizando o protocolo *Post Office Protocol* (POP3). O hardware utilizado foi o kit Rabbit 2000 *Transmission Control Protocol / Internet Protocol* (TCP/IP), o qual dispensa o uso de um computador. Esta proposta especifica apenas as funcionalidades de ligar e desligar eletrodomésticos, possibilitando especificar o horário desejado para tal ação.

3 DESENVOLVIMENTO

Neste capítulo são descritos o levantamento de informações para desenvolvimento do software, suas especificação, seu processo de implementação e apresentados os resultados da discussão. Também são apresentados os diagramas de classes, casos de uso e o modelo entidade relacionamento (MER), além das especificações de requisitos funcionais e não funcionais.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Neste trabalho foi desenvolvido um sistema para automação residencial que possibilita o controle dos dispositivos instalados através de uma interface por comando de voz, assim como através de interfaces gráficas para *desktop*, em ambiente *web* e para a plataforma móvel Android. O sistema permite a visualização do estado e informações do dispositivo, assim como sua localização no ambiente previamente configurado e um mecanismo de resposta e notificação do usuário pela interface gráfica ou por sintetização de voz.

O sistema fornece o controle de acesso por usuário, com possibilidade de restrição de operações. Ele também disponibiliza recursos para gravação de conjuntos de preferências (cenas) para configuração do ambiente com um único comando.

Os dispositivos existentes no sistema são instalados através de pacotes de extensão, que são distribuídos como arquivos no formato *Java Archive* (JAR), contendo recursos como as classes de comunicação com o hardware, especificações de comandos em arquivos *eXtensible Markup Language* (XML) e ícones.

Foi desenvolvido um simulador para demonstração do funcionamento e do modelo de desenvolvimento das interfaces de dispositivo. O sistema desenvolvido é composto de dois elementos principais:

a) *o simulador*, o qual é um servidor que responde aos comandos enviados, mantendo instâncias dos dispositivos hipotéticos instalados no sistema;
b) *um pacote de interfaces simuladas*, que é instalado nos clientes e no servidor e contém a lógica de comunicação com o simulador, através da rede.

A Figura 4 demonstra, através de um diagrama simplificado, a estrutura geral da aplicação.

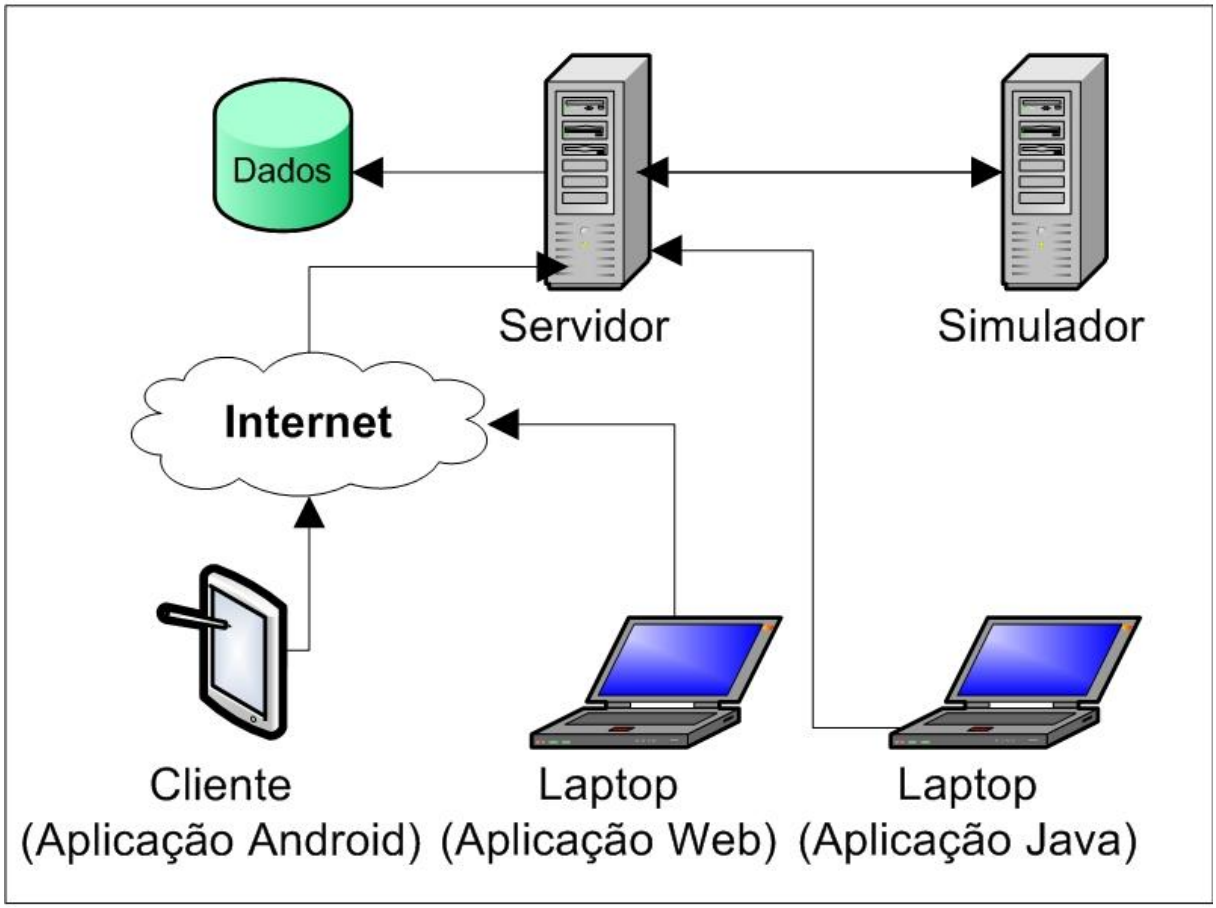


Figura 4 – Diagrama esquemático da aplicação

Foi utilizado um simulador, desenvolvido paralelamente ao sistema principal, para substituir a interação com hardware real. Esta abordagem foi selecionada para eliminar a necessidade de criar interfaces para dispositivos específicos, o que demandaria o conhecimento de um profissional experiente neste tipo de codificação, além de exigir que o programador tenha o acesso físico aos equipamentos.

3.2 ESPECIFICAÇÃO

Nesta seção será apresentada a especificação dos requisitos funcionais e não funcionais, bem como os diagramas de caso de uso e de classes documentados usando a

Unified Modeling Language (UML).

3.2.1 Requisitos do sistema

O Quadro 1 apresenta os requisitos funcionais previstos para o sistema principal e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir ao usuário enviar comandos aos dispositivos.	UC01
RF02: O sistema deverá permitir ao usuário visualizar o estado dos dispositivos instalados.	UC02
RF03: O sistema deverá permitir ao usuário visualizar a localização (geográfica) dos dispositivos.	UC02
RF04: O sistema deverá permitir que o administrador crie permissões de acesso por usuário.	UC06
RF05: O sistema deverá permitir ao administrador manter usuários.	UC07
RF06: O sistema deverá permitir ao administrador manter dispositivos.	UC03
RF07: O sistema deverá permitir ao administrador configurar o mapa de dispositivos.	UC04
RF08: O sistema deverá permitir ao usuário manter conjuntos de preferências (cenas).	UC05

Quadro 1 - Requisitos funcionais do sistema principal

O Quadro 2 apresenta os requisitos funcionais previstos para o sistema simulador de ambiente residencial e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF09: O simulador deverá receber comandos do sistema principal através das interfaces.	UC10

RF10: O simulador deverá fornecer ao sistema principal o estado dos equipamentos instalados.	UC08
RF11: O simulador deverá permitir gerar eventos simulados dos equipamentos para o sistema principal.	UC09

Quadro 2 - Requisitos funcionais do simulador de ambiente residencial

O Quadro 3 lista os requisitos não funcionais previstos para o sistema principal.

Requisitos Não Funcionais
RNF01: O sistema deverá disponibilizar interface de usuário por reconhecimento e sintetização de fala.
RNF02: O sistema deverá disponibilizar interface de usuário através de dispositivo móvel.
RNF03: O sistema deverá disponibilizar interface de usuário através de navegador *web*.

Quadro 3 - Requisitos não funcionais do sistema principal

3.2.2 Casos de uso do sistema

As Figuras 5, 6 e 7 apresentam os diagramas de caso de uso do sistema desenvolvido. O detalhamento dos principais casos de uso é apresentado no Apêndice A.

A Figura 5 apresenta os casos de uso sistema principal, referentes ao controle de dispositivos.

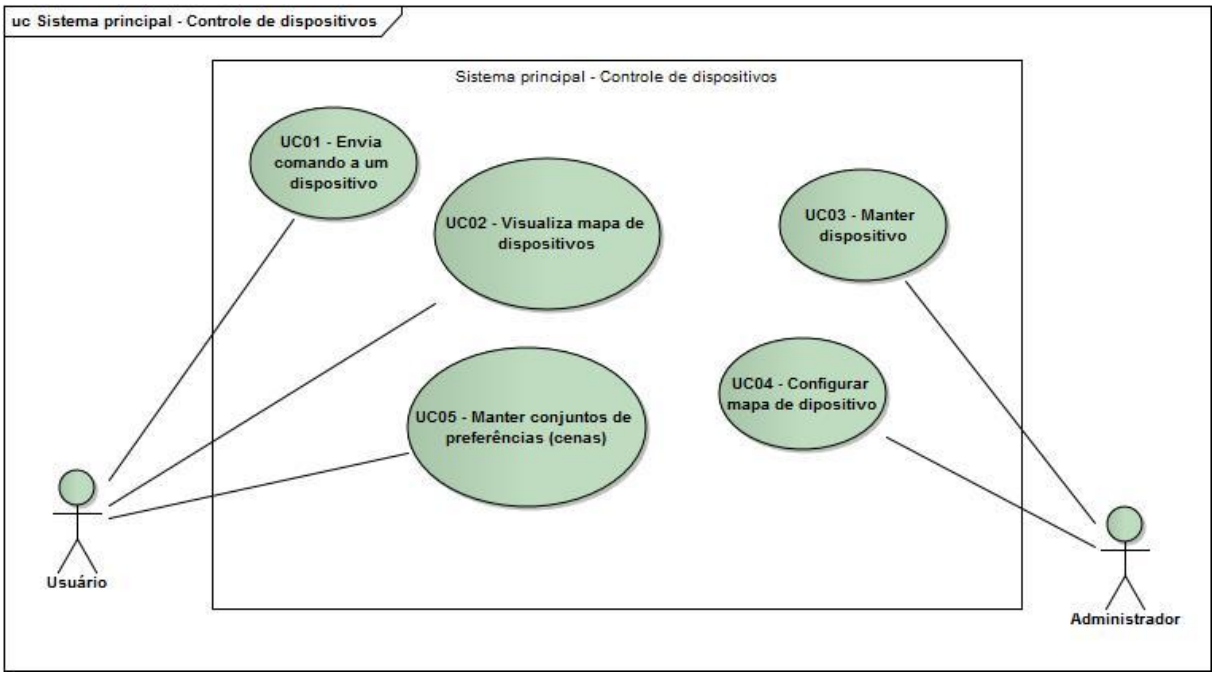


Figura 5 - Casos de uso referentes ao controle de dispositivos

A Figura 6 exibe os casos de uso do sistema principal que dizem respeito às funções gerais.

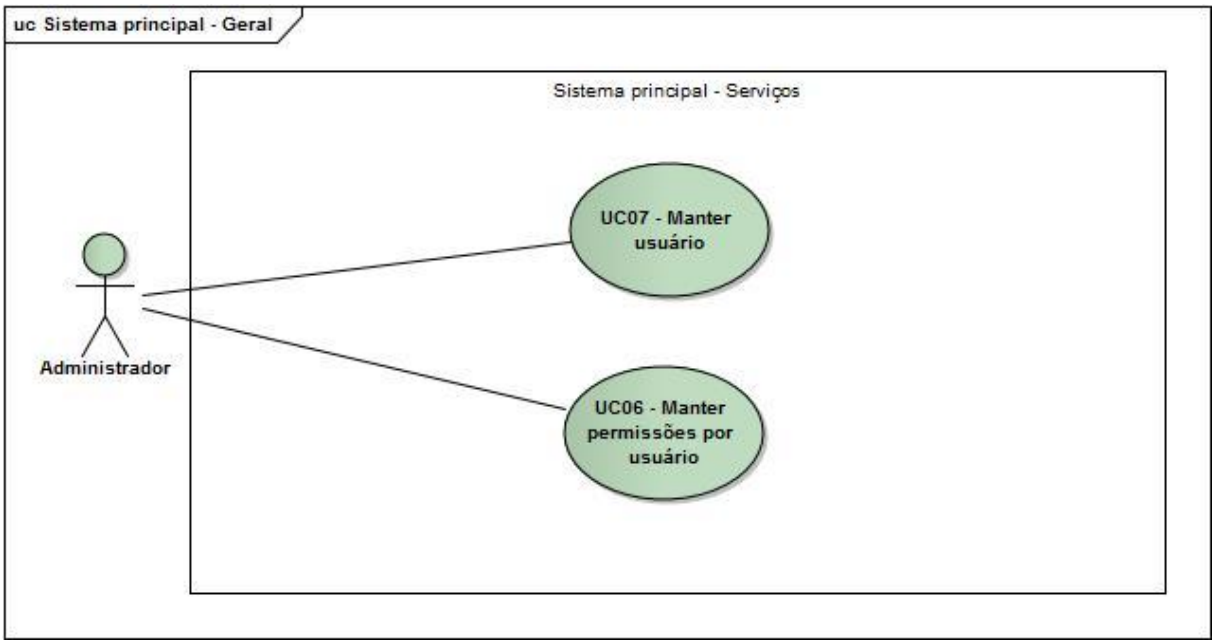


Figura 6 - Casos de uso referentes a funções gerais

A Figura 7 ilustra os casos de uso do simulador de ambiente residencial, onde o sistema principal figura como ator.

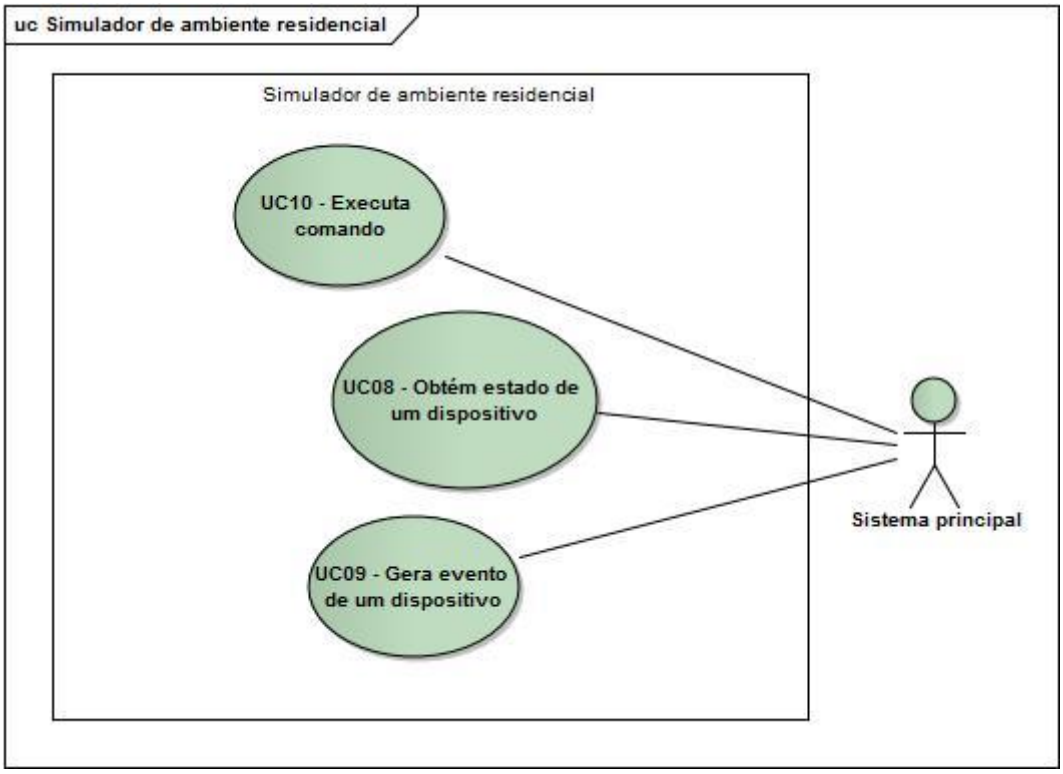


Figura 7 - Casos de uso referentes ao simulador de ambiente residencial

3.2.3 Diagrama de classes

Esta subseção apresenta os diagramas de classes do sistema desenvolvido, com uma breve descrição do papel de cada classe apresentada.

3.2.3.1 Biblioteca comum

O projeto *scha-common* é uma biblioteca contendo algumas funcionalidades comuns aos demais projetos do sistema. A Figura 8 exibe o diagrama de classes da biblioteca.

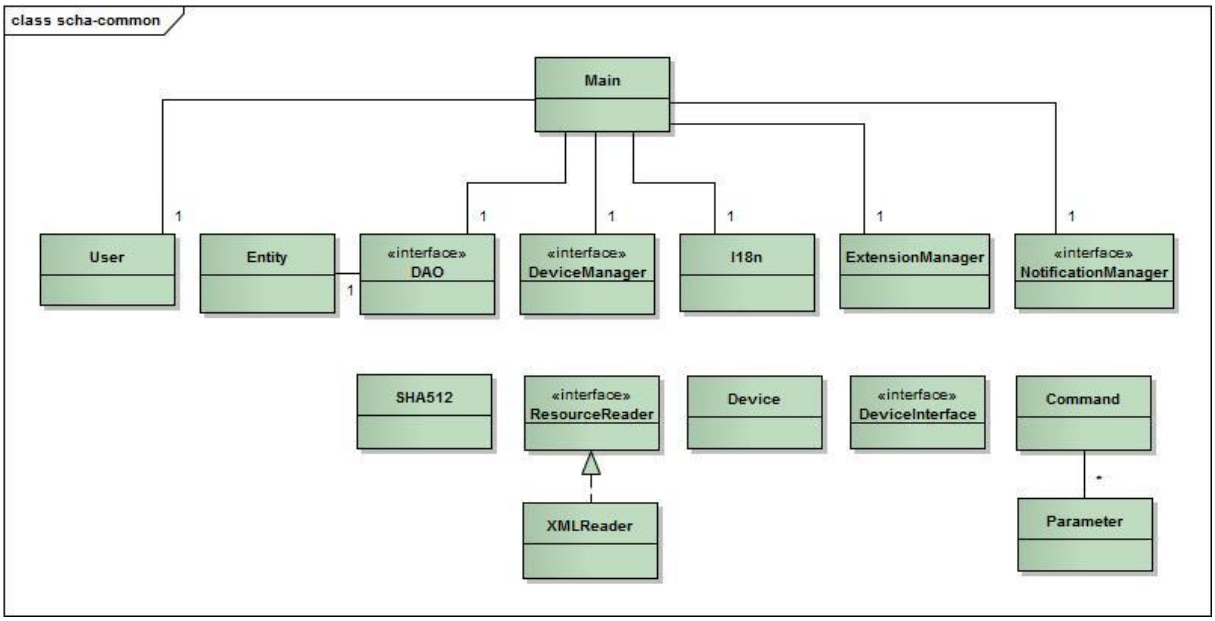


Figura 8 – Diagrama de classes do *scha-common*

a) *Main*: é a classe de ponto de entrada do sistema, inicializando os principais recursos e fornecendo alguns métodos utilitários;
b) *User*: representa a entidade *user* da base de dados;
c) *Entity*: uma classe para auxiliar na abstração da base de dados que fornece algumas operações básicas sobre os registros;
d) *DAO*: uma interface modelo para construção e objetos de acesso a bases de dados;
e) *DeviceManager*: uma interface modelo para criação do gerenciador de dispositivos, implementada no cliente e no servidor;
f) *I18n*: uma classe que fornece métodos para internacionalização de textos no sistema;
g) *ExtensionManager*: classe responsável pela importação dos pacotes de extensão do sistema;
h) *NotificationManager*: uma interface modelo para criação do gerenciador de notificações do sistema. Implementada no cliente e no servidor;
i) *SHA512*: classe para geração de *hashes*;
j) *ResourceReader*: interface modelo para criação das classes de leitura de arquivos de recurso do sistema;
k) *XMLReader*: implementação da interface *ResourceReader* para leitura de arquivos XML;

l) *Device*: representa a entidade *device* da base de dados, permitindo executar operações sobre os registros na base;
m) *DeviceInterface*: interface modelo para a criação de interfaces de dispositivos para o sistema;
n) *Command*: classe que representa um comando direcionado a um dispositivo;
o) *Parameter*: classe que representa um parâmetro de um comando.

3.2.3.2 Biblioteca comum cliente

O projeto *scha-client-common*, apresentado na Figura 9, é uma biblioteca que contém classes comuns aos demais projetos clientes do sistema, como o cliente para *desktop*, *web* e *Android*.

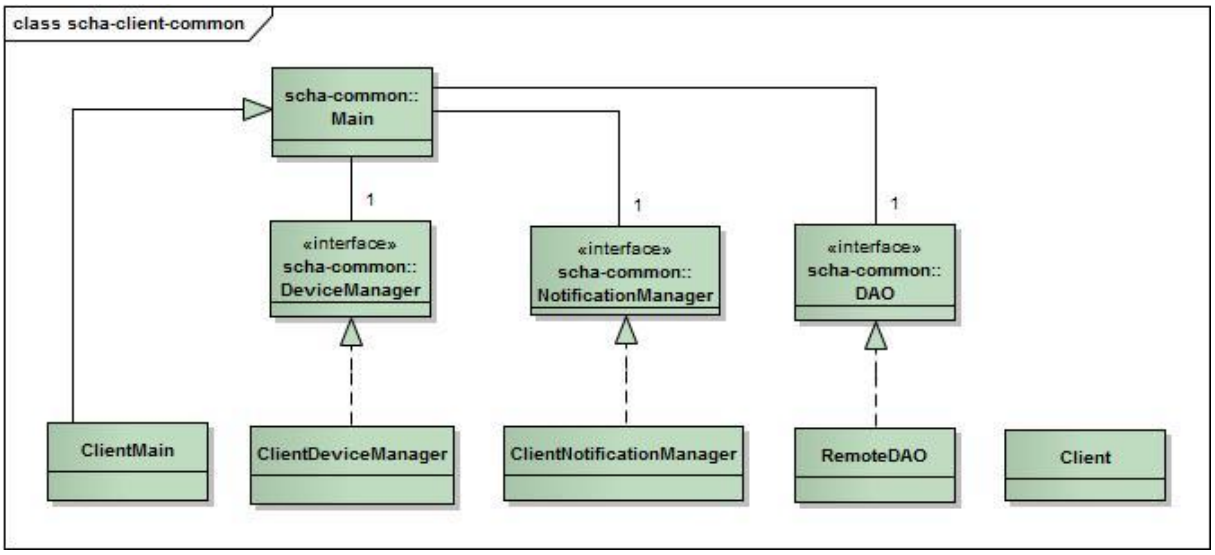


Figura 9 – Diagrama de classes do *scha-client-common*

As classes presentes neste pacote são descritas a seguir:

a) *ClientMain*: é a classe de ponto de entrada do sistema, inicializando os principais recursos e fornecendo alguns métodos utilitários;
b) *ClientDeviceManager*: implementação do gerenciador de dispositivos no cliente;
c) *ClientNotificationManager*: implementação do gerenciador de notificações do sistema;

d) *RemoteDAO*: implementação do objeto de acesso a dados no cliente. Ele obtém os dados da base através de uma conexão com o servidor;
e) *Client*: classe responsável por estabelecer comunicação com o servidor.

3.2.3.3 Servidor

O projeto *scha-server*, apresentado na Figura 10, é o lado servidor do sistema, a partir do qual é efetuada a comunicação com o banco de dados e com os dispositivos.

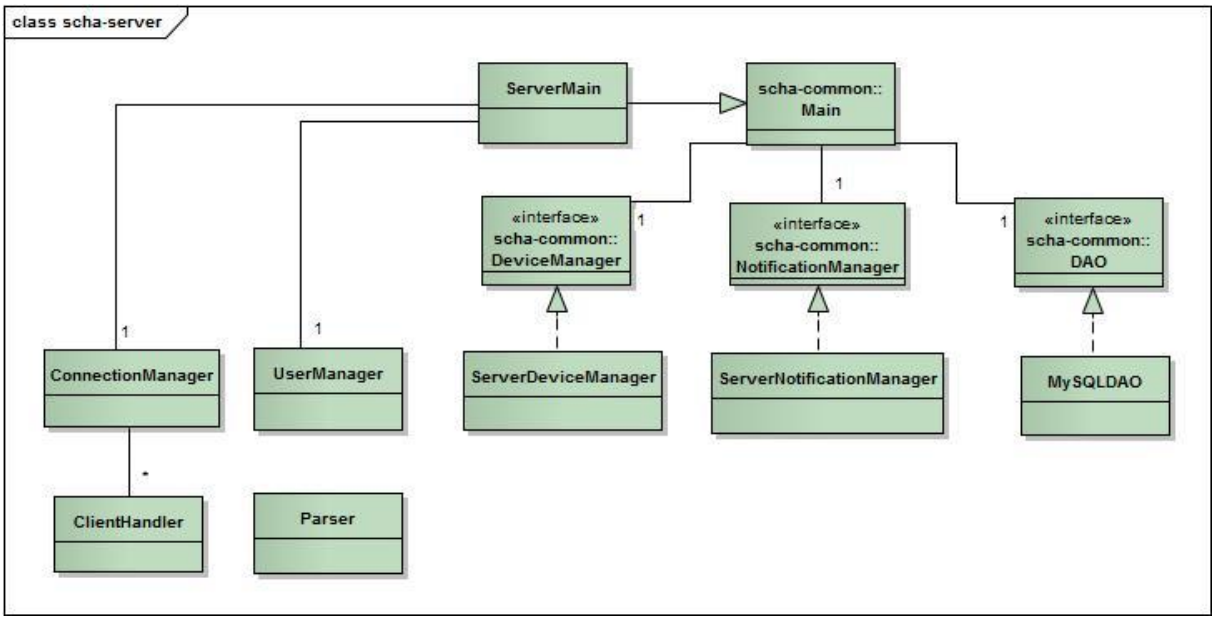


Figura 10 – Diagrama de classes do *scha-server*

As classes presentes neste pacote são descritas a seguir:

a) *ServerMain*: extensão da classe principal no servidor. É o ponto de entrada do programa e fornece métodos utilitários;
b) *ConnectionManager*: gerenciador de conexões do servidor. Estende a classe *ServerSocket*, do Java;
c) *ClientHandler*: classe utilizada para tratar uma requisição recebida em um *thread* separado. Cada conexão recebida pelo *ConnectionManager* é delegada a um *ClientHandler*;

d) *UserManager*: classe utilizada para autenticação e registro de usuários autenticados no sistema;
e) *ServerNotificationManager*: implementação do gerenciador de notificações no servidor;
f) *MySQLDAO*: implementação do objeto de acesso a dados no servidor. Esta classe é especializada na comunicação com o banco de dados *MySQL*;
g) *Parser*: classe responsável pela interpretação de comandos em texto puro, recebidos do cliente e gerados através da interface de reconhecimento de fala.

3.2.3.4 Cliente para *desktop*

O projeto *scha-client-desktop*, apresentado na Figura 11, é o cliente para *desktop* do sistema.

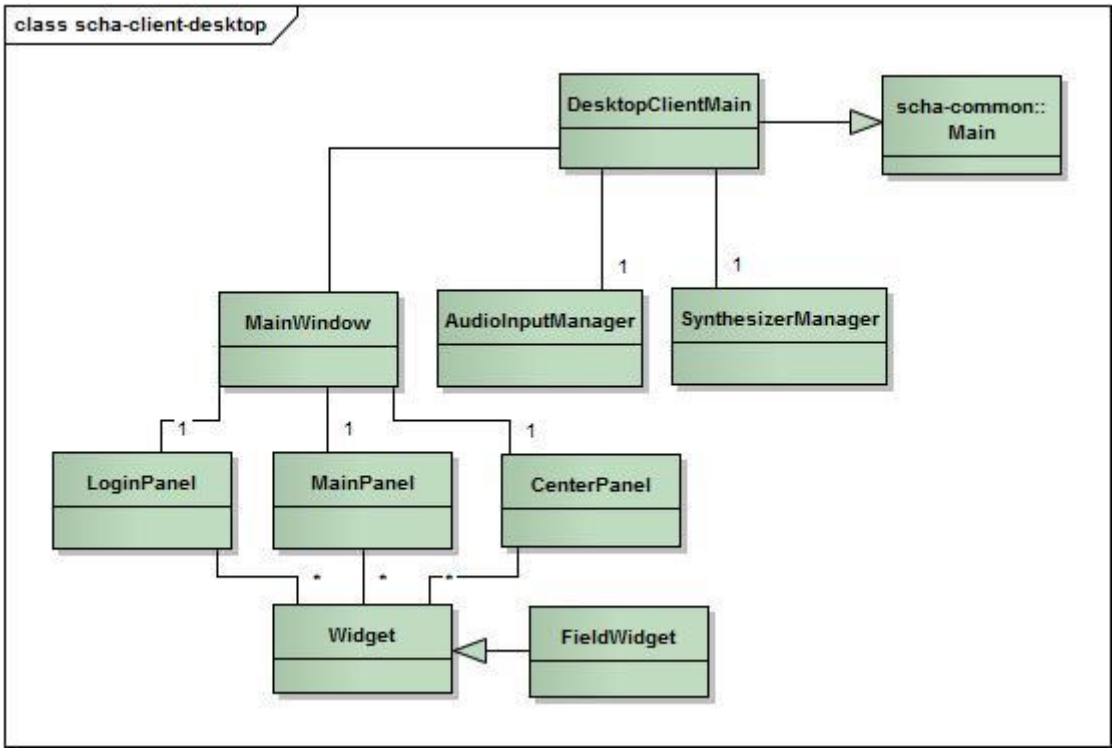


Figura 11 – Diagrama de classes do *scha-client-desktop*

As classes presentes neste pacote são descritas a seguir:

a) *DesktopClientMain*: extensão da classe *Main* no aplicativo para *desktop*;
b) *MainWindow*: a janela do aplicativo;
c) *LoginPanel*: a tela inicial de *login*;
d) *MainPanel*: a tela principal do aplicativo;
e) *CenterPanel*: o painel exibido no centro da tela principal;
f) *Widget*: classe que representa os componentes de interface do sistema;
g) *FieldWidget*: classe que representa os componentes de interface que são campos para entrada de dados;
h) *AudioInputManager*: classe responsável pelo reconhecimento de fala no aplicativo;
i) *SynthesizerManager*: classe responsável pela sintetização de fala no aplicativo.

3.2.3.5 Cliente para *web*

O projeto *scha-client-web*, apresentado na Figura 12, é o aplicativo *web* do sistema.

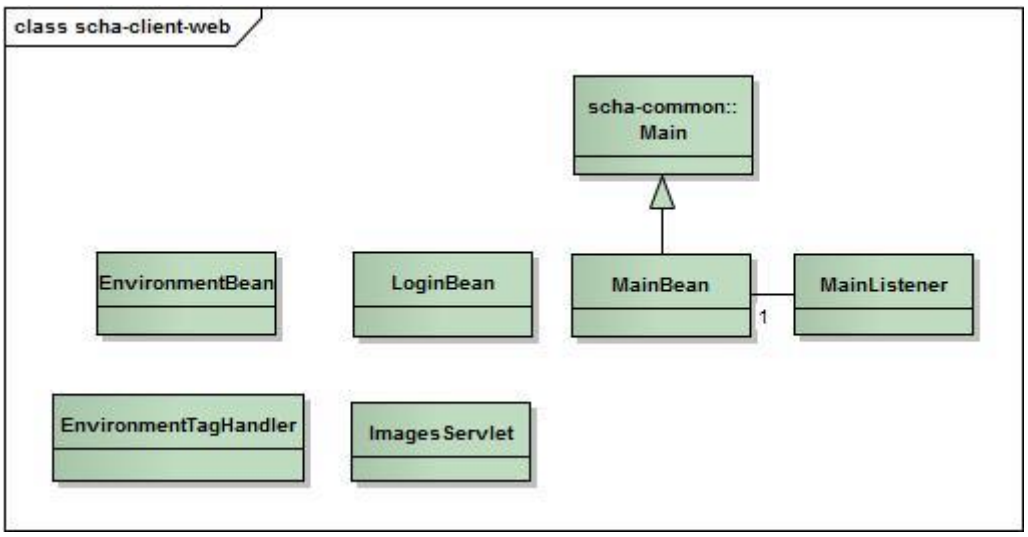


Figura 12 – Diagrama de classes do *scha-client-web*

As classes presentes neste pacote são descritas a seguir:

a) *MainBean*: estensão da classe *Main* no aplicativo para *web*;
b) *LoginBean*: classe que contém a lógica de autenticação do aplicativo *web*;

c) *MainListener*: receptor de eventos ligado ao JSF que é acionado ao iniciar a aplicação para executar operações que serão necessárias ao longo de todo o seu ciclo de vida;
d) *EnvironmentBean*: classe responsável pela lógica de interação com o mapa de dispositivos no lado servidor;
e) *EnvironmentTagHandler*: classe responsável pela lógica de renderização da *tag environment*, para uso em uma página JSF;
f) *ImagesServlet*: um servlet especializado em servir requisições de imagens para as páginas JSF.

3.2.3.6 Simulador

O projeto *scha-simulator*, apresentado na Figura 13, é o simulador de dispositivos do sistema.

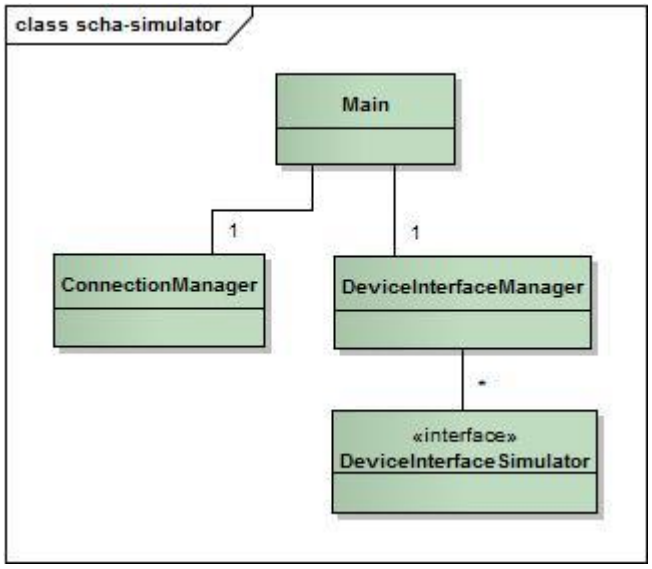


Figura 13 – Diagrama de classes do *scha-simulator*

As classes presentes neste pacote são descritas a seguir:

a) *Main*: ponto de entrada do aplicativo;
b) *ConnectionManager*: classe que atua como servidor do simulador, recebendo e tratando as requisições de rede;

c) *DeviceInterfaceManager*: classe responsável por agrupar os dispositivos simulados do sistema e controlar a geração de eventos para eles;
d) *DeviceInterfaceSimulator*: uma interface modelo para criação dos simuladores de dispositivos.

3.2.3.7 Cliente para *Android*

O projeto *scha-client-android*, apresentado na Figura 14, é o aplicativo cliente para a plataforma *Android*.

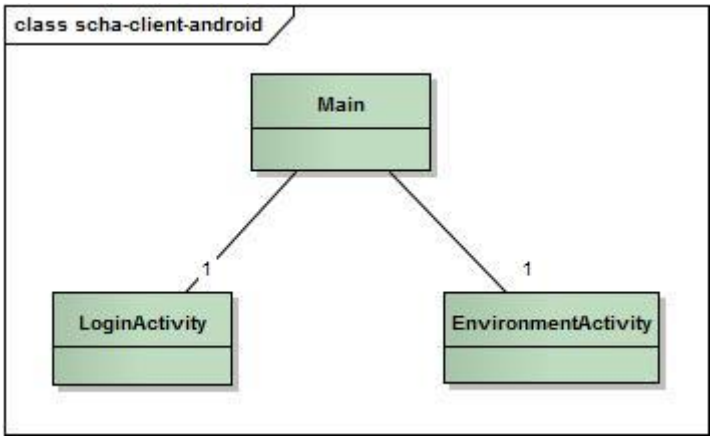


Figura 14 – Diagrama de classes do *scha-client-android*

As classes presentes neste pacote são descritas a seguir:

a) *Main*: classe principais funcionalidades do aplicativo;
b) *LoginActivity*: tela de *login* do sistema;
c) *EnvironmentActivity*: tela do mapa de dispositivos do sistema.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A tecnologia base para desenvolvimento deste sistema foi o Java, sendo utilizada a versão 1.7 do *Java Development Kit*. O ambiente de desenvolvimento, ou *Integrated Development Environment* (IDE), inicialmente selecionado para as atividades de codificação foi o Eclipse na versão *Indigo* 3.7, posteriormente migrando para a versão *Juno* 4.2 devido à instabilidade após a instalação do *plugin Web Tools Platform* (WTP).

3.3.1.1 Persistência de dados

Para a persistência de dados foi selecionado o banco de dados MySQL, na versão 5.5, e utilizado a biblioteca MySQL *Connector* na versão 5.1.1.19 para conexão. O padrão de desenvolvimento adotado para esta funcionalidade foi o *Data Acess Object* (DAO).

A Figura 15 exibe um trecho da interface DAO, que é o modelo de objeto de acesso a dados do sistema desenvolvido. Ela define alguns métodos procurando limitar-se às quatro operações básicas da persistência de dados: *create*, *read*, *update*, e *delete* (CRUD). Esta interface é implementada no servidor pela classe MySQLDAO, especializada na comunicação com o MySQL.

```
36⊕     * Cria um novo registro no repositório de dados.
42      public void create(String entity, Map<String, Object> attributes) throws Exception;
43
45⊕     * Retorna um consjunto de registros obtidos do repositório de dados,
59⊕     public List<Map<String, Object>> read(String entity, List<String> attributes,
61
63⊕     * Retorna um consjunto de registros obtidos do repositório de dados,
73⊕     public List<Map<String, Object>> read(String entity, List<String> attributes,
75
77⊕     * Retorna um consjunto de registros obtidos do repositório de dados,
85      public List<Map<String, Object>> read(String entity, Map<String, Object[]> filter) throws Exception;
86
88⊕     * Retorna um consjunto de registros obtidos do repositório de dados.
93      public List<Map<String, Object>> read(String entity) throws Exception;
94
96⊕     * Atualiza um registro ou consjunto de registros obtidos do repositório
105⊕    public void update(String entity, Map<String, Object> attributes,
107
109⊕    * Atualiza um registro do repositório de dados, correspondente ao {@code id},
117     public void update(String entity, Map<String, Object> attributes, String id) throws Exception;
118
120⊕    * Remove os referidos registros do repositório de dados.
126     public void delete(String entity, Map<String, Object[]> filter) throws Exception;
127
129⊕    * Remove os referidos registros do repositório de dados.
134     public void delete(String entity, String id) throws Exception;
```

Figura 15 – Principais métodos da interface DAO

Com a finalidade de criar uma pequena camada de abstração de base de dados, foi

criada a classe *Entity* (Entidade), que pode ser estendida por outras classes que representem entidades da base para fornecer acesso aos dados.

A Figura 16 exibe a declaração da classe *Entity*. Esta classe implementa a interface *Iterable*, para que possa ser alvo de uma instrução de iteração. Foi adotado como padrão de modelagem o uso de uma coluna nomeada *id* como chave primária para cada entidade do modelo, portanto é declarada uma constante contendo este identificador.

Na linha 20 pode-se observar que uma instância de um objeto DAO é mantida, a qual é utilizada para o efetivo acesso ao banco, e a variável é marcada como transiente para que, quando o objeto for enviado do servidor para o cliente, este valor seja "nulificado". Isto é necessário, pois o cliente não possui a implementação do DAO requerida, desta forma restringido o acesso à base unicamente ao servidor.

```
10
11 /**
12  * Classe de mapeamento de uma entidade do repositório de dados.
13  * @author Rother
14  */
15 public class Entity<T> implements Iterable<T>, Serializable {
16
17     public static final String ENTITY_ATTR_ID = "id";
18     private static final long serialVersionUID = -5049663922274427126L;
19     private String name;
20     private transient DAO dao;
21     private int current;
22     private Map<String, Object[]> lastFilter;
23
24     //Objetos usados para manipulação de registros.
25     private List<Map<String, Object>> data;
```

Figura 16 – Declaração da classe *Entity*

3.3.1.2 Comunicação de rede por *sockets*

A comunicação entre cliente e servidor foi construída com o uso de *sockets*. As classes responsáveis pela comunicação são o *Client* (cliente), *ConnectionManager* (servidor) e *ClientHandler* (trata e responde à requisição do cliente).

A Figura 17 mostra o método responsável pelo início de uma requisição ao servidor. A lógica da requisição é executada em um novo *thread*, e o programa aguarda uma notificação desta para prosseguir. Esta abordagem foi adotada, pois, embora também bloqueie a continuidade da execução do programa, requisições de rede não devem ser efetuadas a partir do *thread* principal.

```java
/**
 * Envia uma requisição para o servidor.
 * @param className o nome da classe a acessar no servidor.
 * @param methodName o nome do método a executar no servidor.
 * @param args os parâmetros do método a executar no servidor.
 * @return a resposta para a requisição.
 */
public Object request(String className, String methodName, Object ... args) {
    try {
        this.className = className;
        this.methodName = methodName;
        this.args = args;
        Thread thread = new Thread(this);
        thread.start();
        synchronized(thread) {
            thread.wait();
        }
        return response;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 17 – Mecanismo de requisição da classe *Client*

3.3.1.3 CMU *Sphinx*

Para efetuar o reconhecimento de fala do sistema foi utilizado o framework *CMU Sphinx*, desenvolvido pela Universidade Carnegie Mellon. O *Sphinx* é um framework para reconhecimento de fala, de código fonte aberto, implementado totalmente na linguagem Java e de redistribuição permitida. Além disso, o *framework* implementa a especificação *Java Speech Application Programming Interface* (JSAPI).

O *Sphinx* atua como um motor de reconhecimento de fala estatístico baseado em *Hidden Markov Models* (HMM), porém depende de um modelo acústico externo, que permite que ele seja facilmente adaptado para diversos idiomas. Foi utilizado um modelo do Laboratório de Processamento de Sinais da Universidade Federal do Pará (UFPA), construído com um corpo de voz baseado na leitura de trechos da constituição brasileira por um indivíduo adulto do sexo masculino.

A Figura 18 mostra o início do processo de reconhecimento de fala. Dentro de um loop indefinido é iniciada a gravação usando um microfone previamente alocado. Um resultado é obtido através do método *recognize* do reconhecedor. A detecção do início e do fim da fala do usuário é gerenciada internamente pelo *Sphinx*. Após o reconhecimento o microfone é fechado até a próxima iteração e é realizada uma chamada do sistema para reciclagem de memória. Se for obtido algum resultado, uma linha de texto contendo o melhor resultado é extraída.

```
42    @Override
43    public void run() {
44        while (true) {
45            microphone.clear();
46            microphone.startRecording();
47            Result result = recognizer.recognize();
48            microphone.stopRecording();
49            System.gc();
50            if (result != null) {
51                String resultText = result.getBestResultNoFiller();
```

Figura 18 – Inicialização da operação do *framework* Sphinx

A Figura 19 mostra um fragmento do arquivo de configuração do *Sphinx*. Este arquivo é o mais frequentemente alterado durante o processo de desenvolvimento, pois contém toda a parametrização do sistema de reconhecimento. O fragmento contém a definição do caminho do dicionário utilizado, que faz parte do modelo acústico.

```
97    <component name="dictionary"
98        type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
99        <property name="dictionaryPath"
100                 value="resource:/cd_cont_1000_16/cd_cont_1000/constituicao.dic"/>
101       <property name="fillerPath"
102             value="resource:/cd_cont_1000_16/cd_cont_1000/noisedict"/>
103       <property name="addSilEndingPronunciation" value="false"/>
104       <property name="wordReplacement" value="&lt;sil&gt;"/>
105       <property name="unitManager" value="unitManager"/>
106   </component>
```

Figura 19 – Configuração do dicionário

3.3.1.4 *Java Server Faces* (JSF)

Como base para desenvolvimento da aplicação *web* foi selecionado o *framework* Java *Server Faces* (JSF). O JSF é um *framework* orientado ao padrão *Model-View-Controller* (MVC) que favorece a integração entre a lógica da aplicação (no lado servidor) e a camada de visualização.

Na Figura 20 é exibida a codificação da página de acesso ao sistema, que contém um formulário padrão de autenticação por usuário e senha. É possível observar a utilização de bibliotecas de *tags* específicas do JSF. Também é possível observar no atributo *value* de cada campo a utilização de um componente para internacionalização dos textos apresentados na interface. Por fim é adicionado um botão de ação que submete o formulário para ser processado em um método no *bean login*.

```
17⊖        <div class="floating-dialog" style="width: 300px; height: 250px;">
18⊖            <div class="error-label" style="padding-top: 130px; margin-left: 5px;">
19                <h:outputText value="#{login.error}"/></div>
20⊖            <h:form>
21⊖                <div style="height: 30px; padding-top: 10px;">
22                    <h:outputText value="#{i18n.user}" styleClass="standard-label"/>
23                    :
24                    <h:inputText styleClass="standard-field" value="#{login.name}"/>
25                </div>
26⊖                <div style="height: 30px;">
27                    <h:outputText value="#{i18n.password}" styleClass="standard-label"/>
28                    :
29                    <h:inputSecret styleClass="standard-field"
30                        value="#{login.password}" />
31                </div>
32⊖                <div>
33                    <h:commandButton styleClass="centered" type="submit" value="#{i18n.login}"
34                        action="#{login.authenticateByUserPassword}"/>
35                </div>
```

Figura 20 – Implementação da tela de *login* do sistema

3.3.2 Operacionalidade da implementação

Esta subseção apresenta as principais telas do sistema e trechos de código relevantes.

3.3.2.1 Cliente para *desktop*

Ao abrir o sistema, o usuário é apresentado à tela de autenticação do usuário. Além de um formulário padrão de *login*, o sistema apresenta uma opção de acesso com o usuário “Convidado”, que não necessita de autenticação, caso esta configuração esteja habilitada no servidor. A Figura 21 mostra a tela de *login* do sistema.

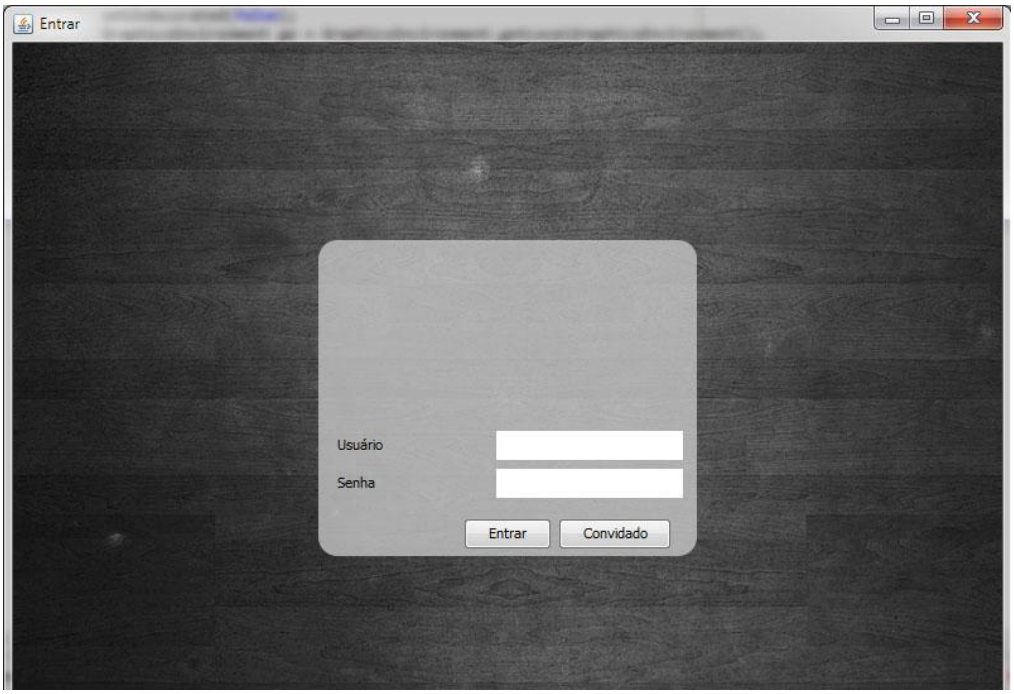


Figura 21 – Tela de *login* do sistema

Após a autenticação o usuário é redirecionado à tela principal do sistema, cuja visualização inicial é o mapa do ambiente com os respectivos dispositivos instalados. Dependendo de suas permissões o usuário visualiza as opções acessíveis no menu no lado esquerdo da tela, como observável na Figura 22.

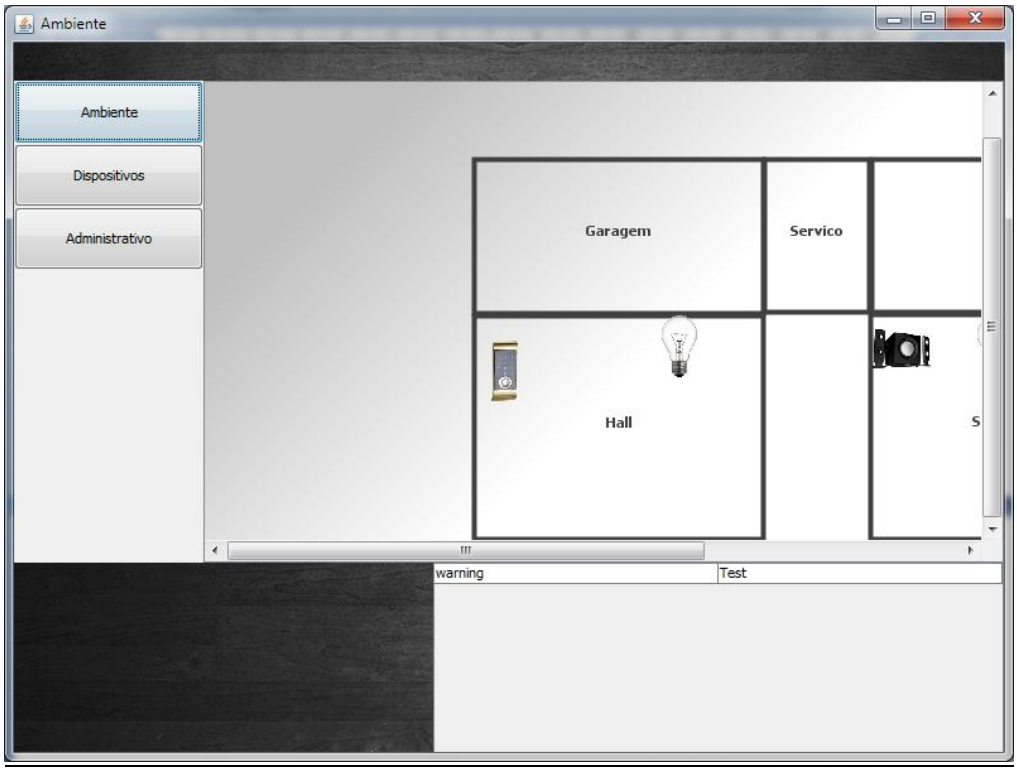


Figura 22 – Mapa do ambiente

Se o usuário clicar em um dos dispositivos posicionados no mapa, serão apresentados os comandos disponíveis para este dispositivo, conforme exibido na Figura 23.

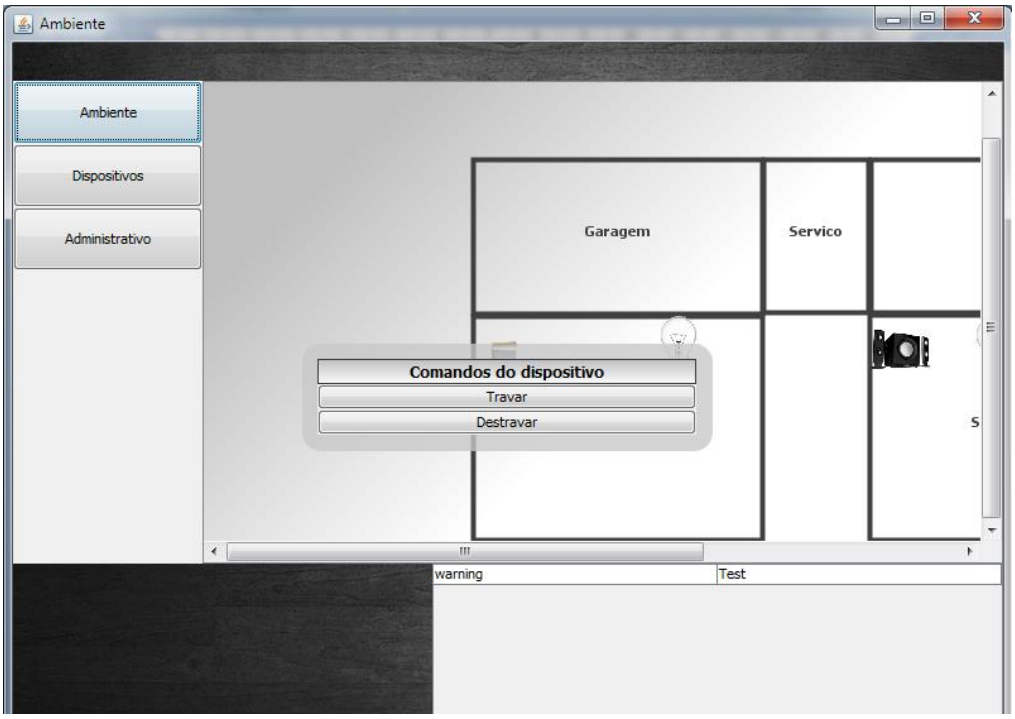


Figura 23 – Seleção de comandos do dispositivo

Ao selecionar um comando, o sistema apresentará as opções (parâmetros) disponíveis para ele. A Figura 24 exibe os parâmetros do comando “Travar” do dispositivo simulado de trava de portas, que se resume ao horário em que a trava será acionada. O usuário emite o comando clicando no botão de confirmação (em verde).

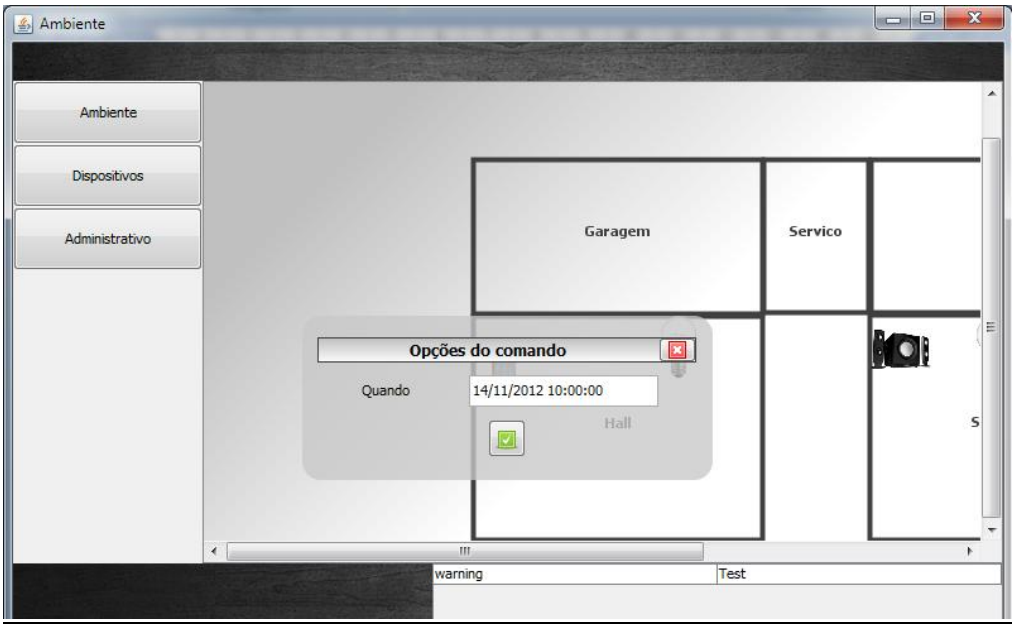


Figura 24 – Opções do comando

Um usuário com a permissão adequada pode acessar o cadastro de usuários. Isso pode ser realizado através do item “Usuários”, sob o menu “Administrativo”. A Figura 25 mostra o cadastro de usuários.

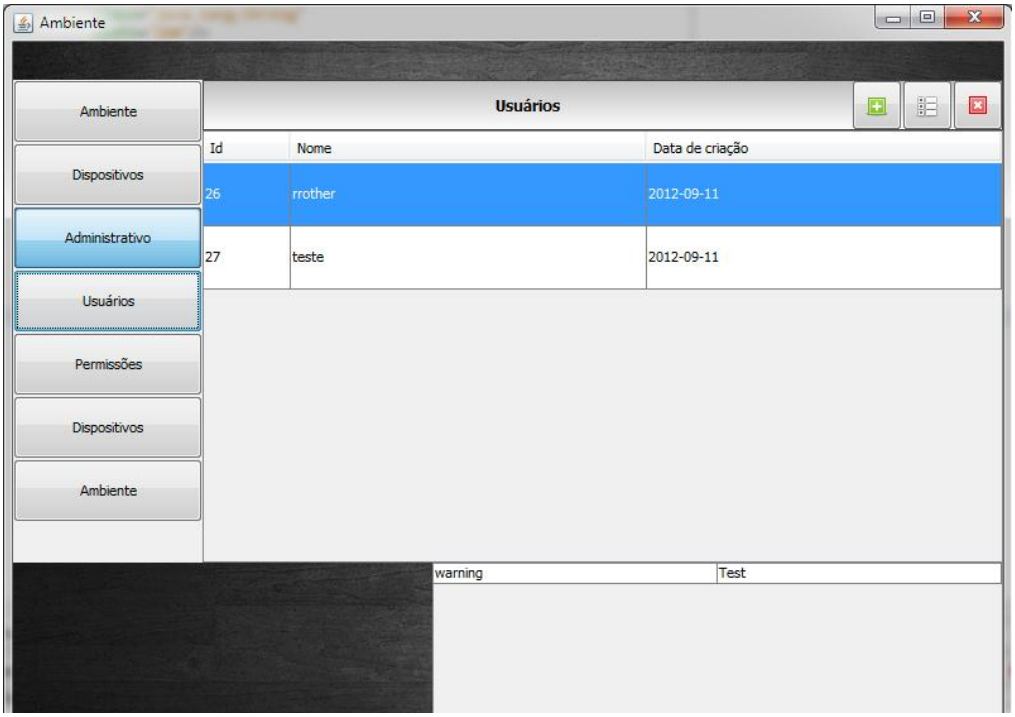


Figura 25 – Cadastro de usuários

É possível cadastrar novos usuários através do ícone em verde no canto superior direito da tela, como exibido na Figura 26.

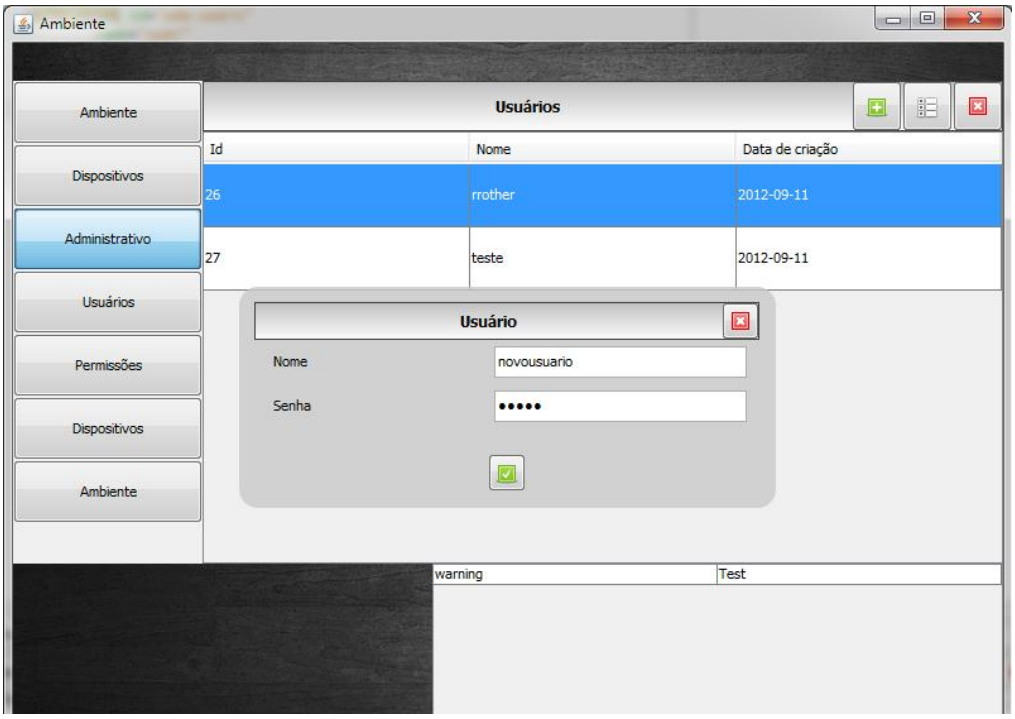


Figura 26 – Cadastrando um novo usuário

O usuário com a devida permissão também pode alterar a configuração do ambiente através da opção *Ambiente* sob o menu *Administrativo*. A Figura 27 exibe a tela de edição do ambiente, na qual é possível adicionar novas áreas e dispositivos, através das opções da barra lateral direita, além de mover os redimensionar os elementos existentes. A área desatacada em vermelho foi selecionada pelo usuário, e está apta para redimensionamento usando as âncoras em cada um dos quatro lados.

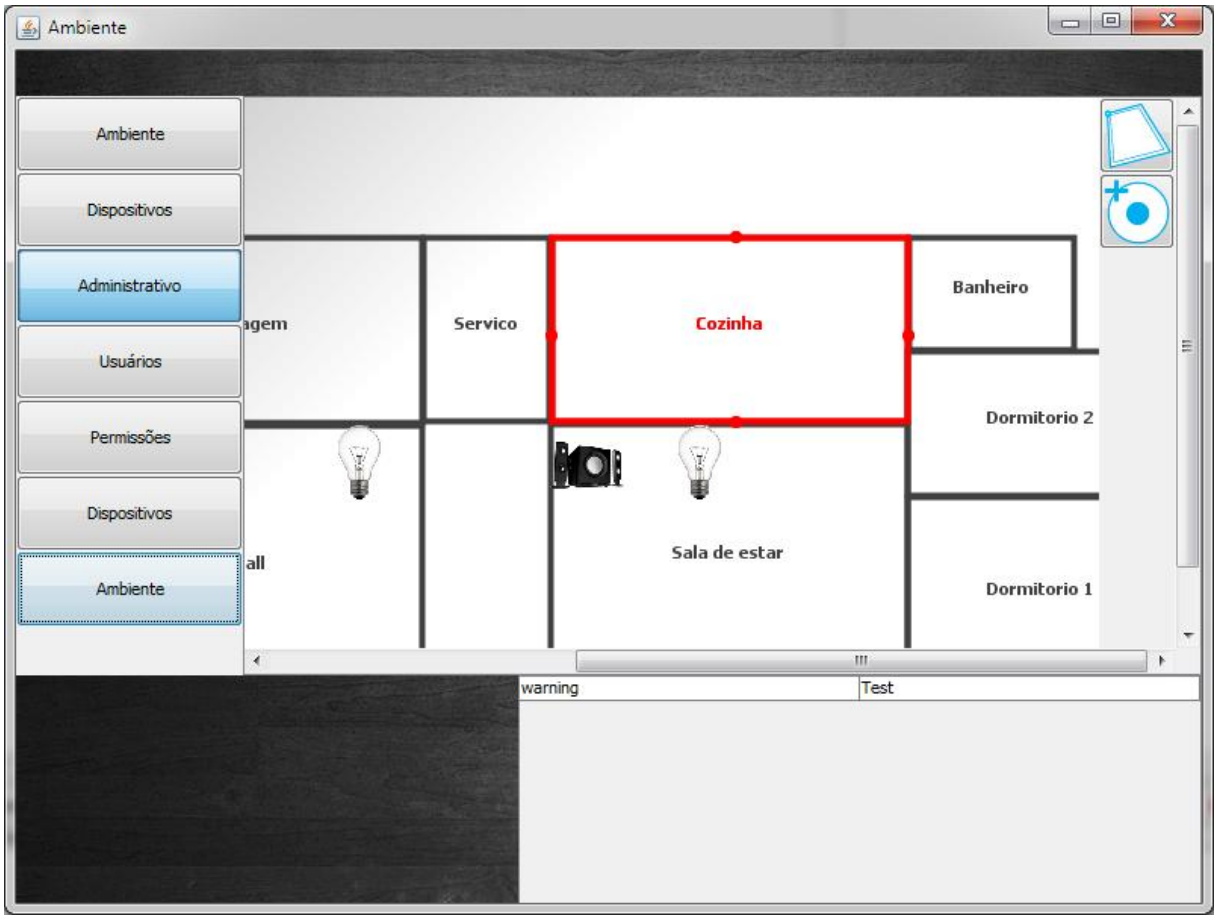


Figura 27 – Tela de configuração do ambiente

3.3.2.2 Cliente para *web*

Ao acessar a página inicial do sistema, o usuário visualiza o formulário de *login*, como ilustrado na Figura 28. A funcionalidade é bastante semelhante à do cliente para *desktop*, apresentada na seção 3.3.2.1.

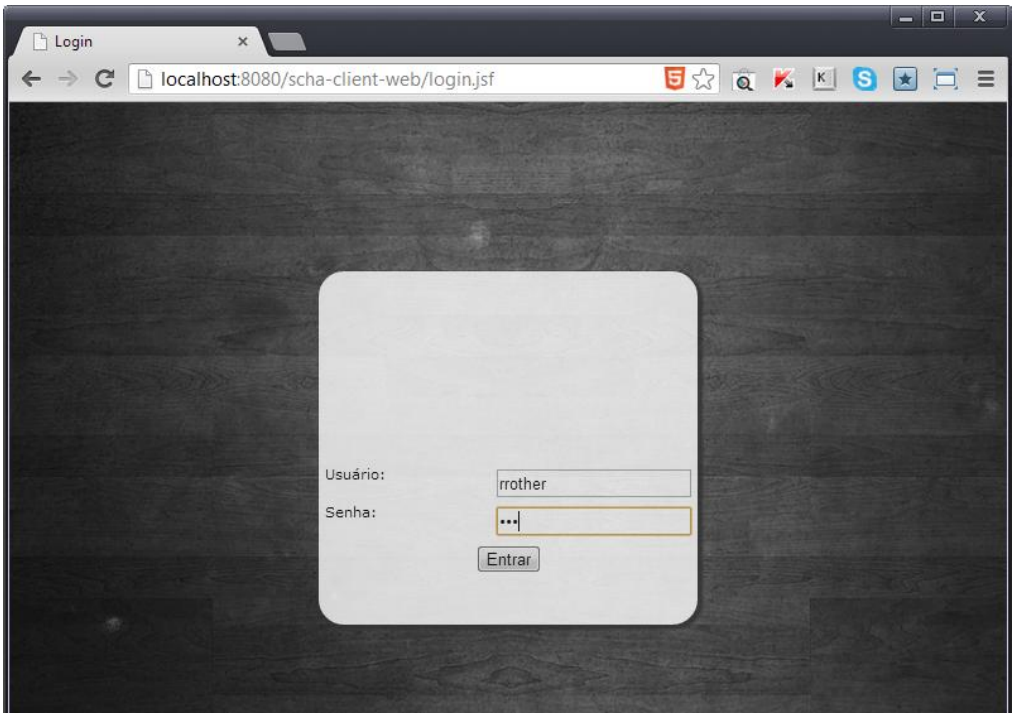


Figura 28 – Tela de login na interface *web*

Após a autenticação o usuário é direcionado para a página contendo o mapa do ambiente, como mostrado na figura 29. A interface *web* não contempla todas as funcionalidades do sistema, apenas o envio de comandos para os dispositivos instalados.

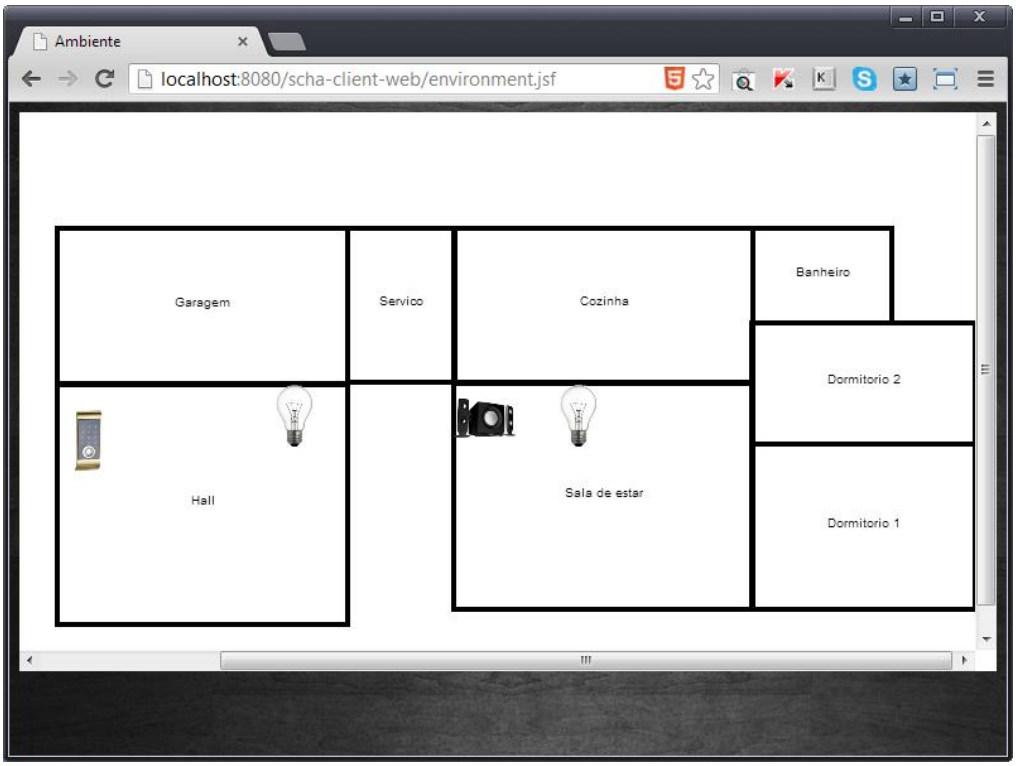


Figura 29 – Mapa do ambiente na interface *web*

Através da interface *web* também é possível enviar comandos clicando sobre os ícones dos dispositivos, como ilustrado na Figura 30.

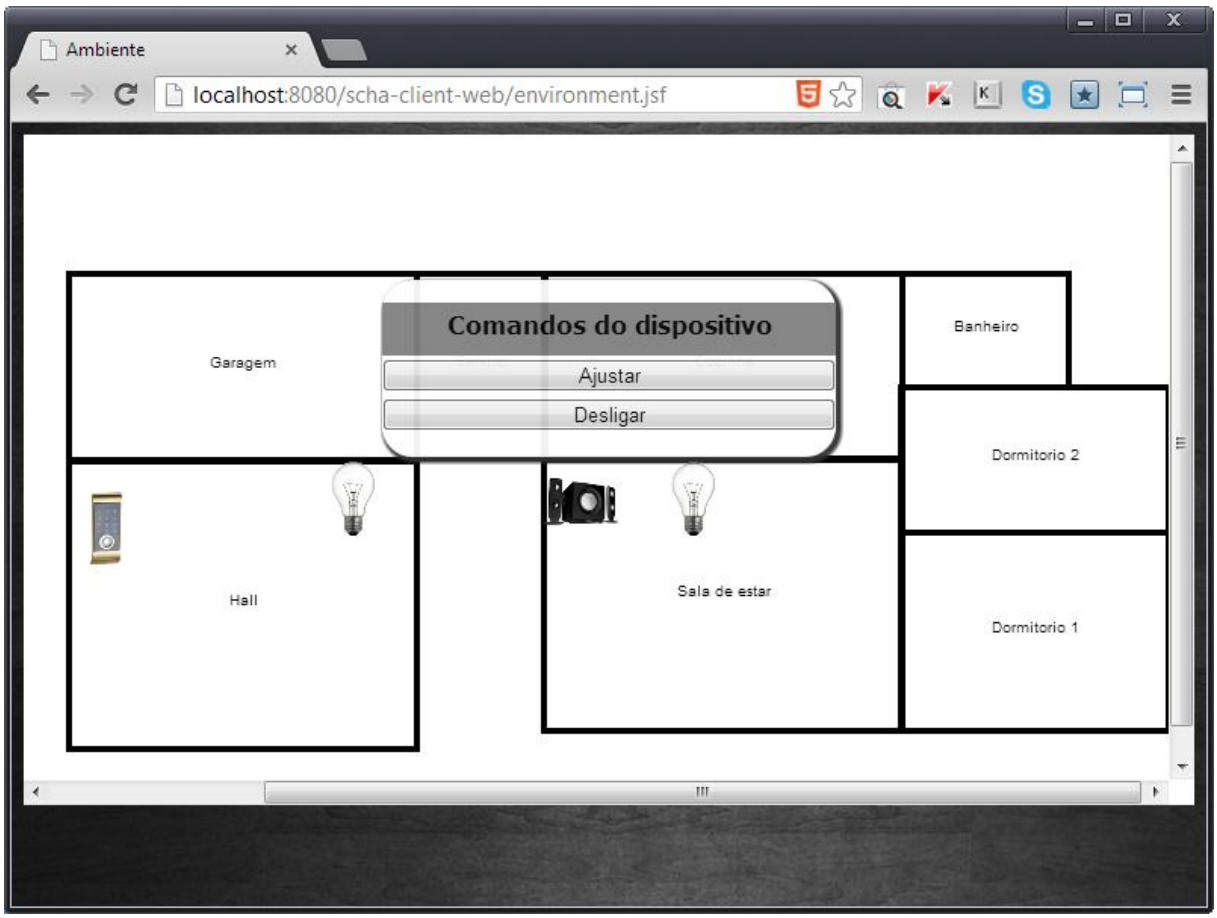


Figura 30 – Envio de comando pelo ambiente *web*

3.4 RESULTADOS E DISCUSSÃO

A proposta inicial de um software para automação residencial com múltiplas interfaces de usuário e capacidade de recepção de comandos de voz foi atendida. O protótipo final deste projeto fornece uma base adequada para interfaces de comunicação com dispositivos reais, que possam ser instaladas sem necessidade de modificações no programa após sua distribuição, para permitir a operacionalidade em um ambiente residencial padrão.

Para o desenvolvimento do sistema, não foi utilizado hardware real. Esta necessidade foi abstraída por um simulador, desenvolvido em conjunto com o sistema e um pacote de interfaces simuladas para comunicação com ele, com a finalidade de demonstrar seu

funcionamento. Para efetuar a comunicação com dispositivos reais, o sistema suporta a importação de pacotes de extensão *Java ARchive* (JAR) contendo classes Java e arquivos XML, a exemplo do pacote “scha-pack-simulated.jar”, distribuído com a aplicação.

4 CONCLUSÕES

Este trabalho mostrou ser possível a integração de múltiplas tecnologias, em ambientes e plataformas diferentes, para composição de um sistema que possa ser portado e estendido com relativa facilidade. É provável que o número de usuários de sistemas de automação residencial continue crescendo nos próximos anos, fazendo com que este mercado também cresça e exija soluções de software cada vez mais flexíveis.

Em relação às técnicas de reconhecimento de fala, a pesquisa realizada enfrentou dificuldades na seleção de uma ferramenta de software livre para esta finalidade, pois, para conferir maior portabilidade ao sistema, foi valorizado o critério da independência de plataforma. Neste ponto, o *framework* Sphinx destacou-se por ser totalmente implementado em Java, multi plataforma, de utilização relativamente simples e de código fonte aberto. O principal ponto negativo é a escassez de modelos acústicos no idioma brasileiro que sejam adequados para reconhecimento de múltiplos indivíduos e de redistribuição livre, o que refletiu no desempenho do sistema desenvolvido no que diz respeito ao reconhecimento de fala.

A tecnologia JSF 2.0 provou ser uma muito amigável em relação ao seu aprendizado e fornecer uma abstração de alguns aspectos da programação em ambiente *web* que simplificam a rotina do programador. O uso de AJAX integrado foi uma melhoria significativa em relação à versão anterior do *framework*.

O HTML5, principalmente no que compete ao elemento *canvas*, utilizado neste sistema, mostrou-se adequado para a composição de interfaces ricas e animações em uma página *web*, eliminando a necessidade de extensões adicionais no navegador.

O principal obstáculo encontrado foi a eficiência do sistema de reconhecimento de fala, que apresentou desempenho muito aquém do esperado devido a pouca disponibilidade de recursos para o idioma português. Insto se refletiu na incapacidade do sistema de efetuar o reconhecimento com precisão suficiente para compor comandos. Devido a esta limitação não foi possível tornar operacional a funcionalidade de reconhecimento de fala.

4.1 EXTENSÕES

Este sistema pode ser melhorado em trabalhos futuros agregando novas funcionalidades ou utilizando técnicas e ferramentas melhores para a execução de algumas tarefas, entre elas:

a) desenvolvimento de um pacote de interfaces para comunicação com dispositivos reais;
b) utilização de técnicas de interpretação semântica de comandos para que o usuário possa utilizar linguagem natural;
c) introduzir técnicas de criptografia na comunicação entre cliente e servidor.

Estas melhorias tornariam o sistema mais próximo de se tornar um produto apto à comercialização.

REFERÊNCIAS

ALONSO, Ignacio G. Service robotics. **Intelligent systems, control and automation**, New York, v. 53, p. 89-114, 2011.

APPLE INC**. IPhone Siri.** [S.l.], 2012. Disponível em: < http://www.apple.com/iphone/features/siri.html>. Acesso em: 07 abr. 2012.

BEIGI, Homayoon. **Fundamentals of speaker recognition**. New York: Springer, 2011.

BURNS, Ed; SCHALK, Chris. **JavaServer Faces 2.0:** The Complete Reference. New York: McGraw Hill, 2010.

CENSI, Angela. **Sistema para automação e controle residencial via e-mail.** 2001. 71 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GADOTTI, Eduardo F. **Sistema para automação e controle residencial via twitter.** 2010. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GOODMAN, Danny; MORRISON, Michael; NOVITSY, Paul; RAYL, Tia G. **JavaScript Bible.** 7th ed. Indianapolis: Wiley, 2010.

MEIER, Reto. **Professional Android 2 application development.** Indianapolis: Wiley, 2010.

MURTY, M. Narashimha; DEVI, V. Susheela. **Pattern recognition:** An Algorithmic Approach. [New York]: Springer, [2011].

PLANNERER, Bernd. **An introduction to speech recognition.** Munique: [s.n], 2005.

RILEY, Mike. **Programming your home:** Automate with Arduino, Android, and your computer. [S.l.]: The Pragmatic Programmers, 2012.

VENTURI, Eli. **Protótipo de um sistema para controle e monitoração residencial através de dispositivos móveis utilizando a plataforma .net.** 2005. 69 f. Trabalho de Conclusão de

Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

W3SCHOOLS. **HTML5 Introduction**. [S.l.], 2012. Disponível em: < http://www.w3schools.com/html/html5_intro.asp>. Acesso em: 07 set. 2012.

WELLS, Quentin. **Guide to digital home technology integration**. New York: Delmar, 2009.

APÊNDICE A – DETALHAMENTO DOS CASOS DE USO

Este Apêndice apresenta a descrição dos casos de uso conforme previstos no(s) diagrama(s) apresentado(s) na subseção 3.3.1.

UC01 Envia comando a um dispositivo
Permite que o usuário, através de uma das interfaces disponibilizadas, envie um comando a um dispositivo no intuito de alterar seu estado ou funcionamento.

Constraints
Pré-condição. O usuário solicitante deve estar cadastrado no sistema.

Cenários
Executar Comando {Principal}.
1. Usuário envia comando ao sistema.
2. Sistema valida comando.
3. Sistema valida permissão do usuário para execução do comando.
4. Sistema valida estado do dispositivo alvo.
5. Sistema envia comando para a interface do dispositivo.
6. Sistema confirma execução do comando.

Comando não reconhecido {Exceção}.
No passo 2, caso o comando não possa ser interpretado, o sistema informa o usuário a respeito da falha.

Usuário não possui permissão {Exceção}.
No passo 3, caso o usuário não disponha de permissões para a execução do comando, o sistema informa o usuário a respeito da falha.

Dispositivo em estado não acessível {Exceção}.
No passo 4, caso o dispositivo não esteja em estado normal de operação, o sistema informa o usuário a respeito da falha.

UC02 Visualiza mapa de dispositivos
Permite que o usuário, através de uma interface gráfica, visualize o posicionamento relativo de cada dispositivo em relação ao posicionamento e dimensões da residência. Também é possível visualizar o estado e enviar comandos aos dispositivos exibidos.

Constraints
Pré-condição. O mapa de dispositivos deve estar configurado no sistema.

Cenários
Consultar Mapa {Principal}.
1. Usuário opta por exibir mapa de dispositivos.
2. Sistema exibe mapa.
3. Usuário opta por uma operação ou encerra caso de uso.

Executar Comando {Alternativo}.
A partir do passo 2, o usuário pode selecionar um dispositivo afim de executar um comando.
2.1. O usuário seleciona um dispositivo dentre os apresentados no mapa.
2.2. O sistema apresenta as operações disponíveis para o dispositivo selecionado.
2.3. O usuário seleciona uma operação.
2.4. Procede-se conforme o UC01.

UC03 Manter dispositivo
Permite ao administrador, através de uma interface gráfica, consultar os dispositivos cadastrados, assim como cadastrar, alterar e remover dispositivos.

Cenários
Consultar dispositivos {Principal}.
1. O administrador acessa o cadastro de dispositivos através do painel administrativo do sistema.
2. O sistema apresenta a lista de dispositivos cadastrados.
3. O administrador opta por visualizar os detalhes do dispositivo.
4. O sistema exibe a tela de detalhes do dispositivo.
5. O administrador opta por uma operação ou encerra o caso de uso.

Incluir dispositivo {Alternativo}.
A partir do passo 2, o administrador pode optar por incluir um dispositivo.
2.1. O administrador opta por incluir um dispositivo.
2.2. O sistema apresenta a tela de inclusão das informações do dispositivo.
2.3. O administrador insere as informações do dispositivo.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Editar dispositivo {Alternativo}.
A partir do passo 2, o administrador pode optar por editar um dispositivo.
2.1. O administrador opta por editar um dispositivo.
2.2. O sistema apresenta a tela de edição das informações do dispositivo.
2.3. O administrador edita as informações do dispositivo.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Excluir dispositivo {Alternativo}.
A partir do passo 2, o administrador pode optar por excluir um dispositivo.
2.1. O administrador opta por excluir um dispositivo.
2.2. O sistema solicita confirmação da operação.
2.3. O usuário confirma a operação.
2.4. O sistema exclui o registro do dispositivo.

UC04 Configurar mapa de dispositivo
Permite ao administrador, através de uma interface gráfica, configurar o mapa de dispositivos do sistema.

Cenários
Editar mapa {Principal}.
1. O administrador solicita a edição do mapa através do painel administrativo do sistema.
2. O sistema apresenta a tela de edição do mapa.
3. O administrador efetua alterações no mapa.
4. O administrador confirma as alterações.
5. O sistema persiste as informações.

UC05 Manter permissões de usuário
Permite ao administrador, através de uma interface gráfica, consultar as permissões definidas por usuário, assim como incluir, editar e remover cadastros.

Cenários
Consultar permissões {Principal}.
1. O administrador acessa o cadastro de perfis através do painel administrativo do sistema.
2. O sistema apresenta a s permissões por usuário cadastrado.
4. O sistema exibe a tela de detalhes da permissão.
5. O administrador opta por uma operação ou encerra o caso de uso.

Incluir permissão {Alternativo}.
A partir do passo 2, o administrador pode optar por incluir uma permissão.
2.1. O administrador opta por incluir uma permissão.
2.2. O sistema apresenta a tela de inclusão de permissões.
2.3. O administrador seleciona a permissão desejada.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Editar permissão {Alternativo}.
A partir do passo 2, o administrador pode optar por editar uma permissão.
2.1. O administrador opta por editar uma permissão.
2.2. O sistema apresenta a tela de edição das informações da permissão.
2.3. O administrador seleciona uma permissão.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Excluir perfil {Alternativo}.
A partir do passo 2, o administrador pode optar por excluir uma permissão.
2.1. O administrador opta por excluir uma permissão.
2.2. O sistema solicita confirmação da operação.
2.3. O administrador confirma a operação.
2.4. O sistema valida a operação.
2.5. O sistema exclui o registro da permissão.

UC06 Manter conjuntos de preferências (cenas)

Permite ao usuário, através de uma das interfaces disponibilizadas, consultar os conjuntos de preferências cadastrados, assim como incluir, editar e excluir cadastros..

Cenários
Consultar conjuntos de preferências {Principal}.
1. O usuário solicita a consulta de conjuntos de preferências.
2. O sistema apresenta a lista de conjuntos de preferências cadastrados.
3. O usuário opta por visualizar os detalhes do conjunto de preferências.
4. O sistema exibe a tela de detalhes do conjunto de preferências.
5. O usuário opta por uma operação ou encerra o caso de uso.

Incluir conjunto de preferências {Alternativo}.
A partir do passo 2, o usuário pode optar por incluir um conjunto de preferências.
2.1. O usuário opta por incluir um conjunto de preferências.
2.2. O sistema apresenta a tela de inclusão das informações do conjunto de preferências.
2.3. O usuário insere as informações do conjunto de preferências.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Editar conjunto de preferências {Alternativo}.
A partir do passo 2, o usuário pode optar por editar um conjunto de preferências.
2.1. O administrador opta por editar um conjunto de preferências.
2.2. O sistema apresenta a tela de edição das informações do conjunto de preferências.
2.3. O usuário edita as informações do conjunto de preferências.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Excluir conjunto de preferências {Alternativo}.
A partir do passo 2, o usuário pode optar por excluir um conjunto de preferências.
2.1. O usuário opta por excluir um conjunto de preferências.
2.2. O sistema solicita confirmação da operação.
2.3. O usuário confirma a operação.
2.4. O sistema valida a operação.
2.5. O sistema exclui o registro do conjunto de preferências.

UC07 Manter usuário
Permite ao administrador, através de uma interface gráfica, consultar os usuários cadastrados, assim como incluir, editar e excluir cadastros..

Cenários
Consultar usuários {Principal}.
1. O administrador solicita a consulta de usuários.
2. O sistema apresenta a lista de usuários cadastrados.
3. O administrador opta por visualizar os detalhes do usuário.
4. O sistema exibe a tela de detalhes do usuário.
5. O administrador opta por uma operação ou encerra o caso de uso.

Incluir usuário {Alternativo}.

A partir do passo 2, o administrador pode optar por incluir um usuário.
2.1. O administrador opta por incluir um usuário.
2.2. O sistema apresenta a tela de inclusão das informações do usuário.
2.3. O administrador insere as informações do usuário.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Editar usuário {Alternativo}.
A partir do passo 2, o administrador pode optar por editar um usuário.
2.1. O administrador opta por editar um usuário.
2.2. O sistema apresenta a tela de edição das informações usuário.
2.3. O administrador edita as informações do usuário.
2.4. O sistema valida as informações.
2.5. O sistema persiste as informações.

Excluir usuário {Alternativo}.
A partir do passo 2, o administrador pode optar por excluir um usuário.
2.1. O administrador opta por excluir um usuário.
2.2. O sistema solicita confirmação da operação.
2.3. O administrador confirma a operação.
2.4. O sistema valida a operação.
2.5. O sistema exclui o registro do usuário.

UC10 Executa comando
Permite que o sistema principal, através das interfaces de dispositivo ativas, envie comandos ao simulador de ambiente residencial para que sejam executados.

Constraints
Pré-condição. O comando deve ter sido validado na interface do dispositivo.
Pré-condição. Estado do dispositivo deve ter sido validado na interface.

Cenários
Executar Comando {Principal}.
1. Sistema envia comando através de interface do dispositivo.
2. Simulador executa comando em dispositivo simulado.
3. Simulador retorna resultado do comando ao sistema.
4. Sistema exibe informações de retorno do comando.

UC08 Obtém estado de um dispositivo
Permite que o sistema principal, através das interfaces de dispositivo ativas, obtenha o estado atual de um dispositivo simulado.

UC09 Gera evento de um dispositivo
Permite que o dispositivo simulado envie ao sistema principal, através do simulador de ambiente residencial, eventos simulados.