UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GEOVANI FABRICIO WELTER

SISTEMA PARA INTERAÇÃO COM O EQUIPAMENTO ROUTERBOARD

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2012

GEOVANI FABRICIO WELTER

SISTEMA PARA INTERAÇÃO COM O EQUIPAMENTO ROUTERBOARD

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2012

ATA Nº: **200**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **GEOVANI FABRíCIO WELTER**.

Às 10:20 hrs do dia 11 de outubro de 2012, Bloco S da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Robison Cris Brito (Convidado) e Alisson Andrey Puska (Convidado), para avaliar o Trabalho de Diplomação do aluno Geovani Fabrício Welter, matrícula 1030736, sob o título **Sistema para Interação com o Equipamento Routerboard**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 11:35 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Prof. Alisson Andrey Puska, Esp.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

WELTER, Geovani Fabricio. Sistema para interação com o equipamento RouterBoard. 2012. 79 f. Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2012.

A orientação a objetos possibilita entender e representar um sistema como um conjunto de objetos. Esses objetos possuem atributos que os caracterizam e realizam ações. As ações são realizadas tendo como base esses atributos. Esses objetos também se comunicam por meio da troca de mensagens. O uso de conceitos de orientação a objetos facilita o entendimento do sistema para o qual será elaborada uma solução computacional (software ou aplicativo, dentre outras denominações) a ser desenvolvida a partir do seu conceito de negócio. Isso porque o mundo real é composto por objetos sejam eles físicos ou conceituais. Em termos de análise e projeto de sistemas computacionais é comum que sejam criados objetos que não possuem correspondência direta com objetos do mundo real. Visando utilizar as facilidades e os recursos para modelagem de sistemas computacionais oferecidos pelos conceitos de orientação a objetos, este relatório de estágio apresenta a análise e o projeto de um aplicativo que será desenvolvido para interagir por meio de uma API com um equipamento de rede denominado RouterBoard.

Palavras-chave: Orientação a objetos. Controle do equipamento RouterBoard. UML.

ABSTRACT

WELTER, Geovani Fabricio. System to interact with RouterBoard. 2012. 79 f. Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2012.

Object orientation make to understand and represent a system as a set of objetct possible. These objects have attributes that characterize them and perform actions. The actions are performed based on these attributes. The objects communicate by exchanging messages. Using concepts of object orientation facilitates the understanding of the system for which a computational solution (software or application, among other names) will be developed from its business processes. The real world is composed of objects whether physical or conceptual. In terms of analysis and design of computer systems is common that objects have no direct correspondence with real world objects. Aiming to use the facilities and resources for modeling computational systems offered by concepts of object orientation, this text presents the development of an application that is designed to interact through an API with a network equipment called RouterBoard.

Palavras-chave: Object orientation. RouterBoard. UML.

LISTA DE QUADROS

LISTA DE FIGURAS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
API	*Application Programming Interface*
BGP	*Border Gateway Protocol*
CDC	*Connected Device Configuration*
CLDC	*Connected Limited Device Configuration*
DLL	*Dynamic-Link Library*
HTTP	*Hypertext Transfer Protoco*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JDK	*Java Development Kit*
Java EE	*Java Enterprise Edition*
Java ME	*Java Micro Edition*
JRE	*Java Runtime Edition*
Java SE	*Java Standard Edition*
MAC	*Media Access Control*
MPLS	*Multiprotocol Label Switching*
OMG	*Object Management Group*
OSPF	*Open Shortest Path First*
PPPoe	*Point-to-Point Protocol over Ethernet*
RIP	*Routing Information Protocol*
SPF	*Small Form-Factor Pluggable*
SQL	*Structured Query Language*
SSH	*Secure Shell*
UML	*Unified Modeling Language*
VPLS	*Virtual Private LAN Service*
VPN	*Virtual Private Network*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma visão geral do trabalho, os objetivos e a justificativa. Por fim está a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

As empresas precisam ser mais competitivas para expandir-se ou mesmo manter-se no mercado. Uma das formas de aumentar a competitividade é por meio da agregação de valor e eficiência em atender seus clientes pelo uso de sistemas computacionais. As tecnologias de informação e comunicação têm facilitado e contribuído para a melhora dos produtos, o atendimento e o oferecimento de soluções aos usuários. São equipamentos e sistemas computacionais mais eficientes, seja em termos de desempenho, segurança ou de funcionalidades.

Uma das formas, denominada também de paradigma, de definir e implementar esses sistemas é por meio do uso de conceitos da orientação a objetos. Cada objeto possui identidade permitindo individualizá-lo dos demais objetos. Esses são caracterizados por atributos e possuem operações que são realizadas utilizando esses atributos. Assim, os dados e a manipulação desses dados ficam incorporados ao objeto. É uma forma de proteger os dados, assegurando que os mesmos não sejam indevidamente manipulados. Apesar desse encapsulamento de dados e operações, esses objetos se comunicam por meio da troca de mensagens. O objeto destinatário da mensagem torna explícito o formato dessa mensagem indicando parâmetros e retorno da mesma.

Visando utilizar conceitos de orientação a objetos na análise e no projeto de um sistema, esse trabalho apresenta conceitos de análise orientada a objetos e da UML *(Unified Modeling Language)* aplicados no desenvolvimento de um sistema para manipular ações e informações dentro de um RouterBoard utilizado em provedores de acesso a internet. A UML é utilizada como linguagem para representar um sistema computacional utilizando conceitos da orientação a objetos.

O RouterBoard utiliza o sistema operacional RouterOS que é baseado em Linux e voltado para soluções em redes de computadores. Muitos provedores de acesso à Internet utilizam esse sistema operacional, pelo mesmo apresentar uma gama muito grande de soluções para diversos tipos de serviço e pela facilidade que sua interface gráfica proporciona.

A Mikrotik, empresa da Letônia fabricante do equipamento RouterBoard e desenvolvedora do sistema operacional RouterOS disponibiliza em seu site uma API (*Application Programming Interface*) para comunicação com o RouterBoard em Java.

Os provedores de acesso à Internet estão buscando sempre novas soluções e a World Line Net, pioneira na utilização desse sistema operacional, está buscando agregar valor aos serviços oferecidos aos seus clientes por meio de um sistema computadorizado para manipular ações e informações dentro do sistema RouterOS de forma segura e padronizada.

1.2 OBJETIVOS

O objetivo geral se refere ao resultado principal pretendido com a realização deste trabalho e os objetivos específicos o complementam.

1.2.1 Objetivo Geral

- Implementar sistema, aplicando conceitos de análise orientados a objeto, para manipular ações e informações dentro de um equipamento eletrônico denominado RouterBoard que executa um sistema operacional RouterOS.

1.2.2 Objetivos Específicos

- Apresentar os conceitos básicos de orientação a objetos e sua modelagem utilizando a UML.
- Apresentar a modelagem de um sistema computacional para execução em um equipamento RouterBoard.
- Desenvolver o sistema, incluindo o uso da *Application Programming Interface* em Java para comunicação com o equipametno RouterBoard

1.3 JUSTIFICATIVA

A sistema implementado tem como base a necessidade verificada dentro da empresa World Line Net de possuir um sistema de controle de ações e manipulação de informações dentro de um equipamento de rede. Esse equipamento é o RouterBoard.

Com um sistema informatizado para manipular ações e informações dentro de um equipamento de rede a empresa terá um ganho considerável em segurança e padrão de configuração de equipamentos em diferentes cidades. Um funcionário inexperiente, por exemplo, que atualmente precisa de permissão de administrador para trabalhar dentro do ambiente RouterOS contará com o auxílio de um sistema computacional. Esse sistema permitirá que o funcionário realize as atividades de forma segura e eficaz, inibindo possíveis erros na configuração , já que atualmente este processo é feito manualmente.

A implementação do sistema será feita utilizando a linguagem Java que foi escolhida como ferramenta de programação. A escolha dessa linguagem decorre dos recursos que a mesma possui e de ser baseada em orientação a objetos, por trabalhar em arquiteturas distintas necessitando apenas de uma JDK (*Java Development Kit*), ter compiladores gratuitos que atendem as necessidades do sistema a ser implementado e que facilitam a implementação orientada a objetos.

A análise orientada a objetos foi utilizada por facilitar a modelagem e a implementação do sistema propriamente dito. Além de proporcionar a reutilização e entendimento do código no próprio projeto e posteriormente por outros desenvolvedores.

1.4 ORGANIZAÇÃO DO TEXTO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a introdução, com os objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico sobre orientação a objetos e a linguagem de modelagem UML. No Capítulo 3 é apresentado o material e o método utilizados na realização das atividades para alcançar os objetivos definidos. O Capítulo 4 contém a modelagem do sistema. No Capítulo 5 está a conclusão com as considerações finais.

2 MODELAGEM DE SISTEMAS UTILIZANDO ORIENTAÇÃO A OBJETOS

A fundamentação teórica do trabalho tem como base a orientação a objetos e a modelagem de sistemas utilizando a UML.

2.1 ORIENTAÇÃO A OBJETOS

Durante os anos 1960 foi introduzido um novo conceito de programação orientada a objetos através da implementação de uma linguagem de programação chamada Simula (FERRANTE; RODRIGUEZ, 2000). Esse conceito representa como os dados são tratados dentro de um software.

Durante anos o termo orientado a objetos foi usado para denotar uma abordagem de desenvolvimento de software que utiliza linguagens orientadas a objetos. Atualmente o paradigma de orientação a objetos abrange um enfoque completo de engenharia de *software* (PRESSMAN, 2002). Isso significa que orientação a objetos está envolvida desde a representação do negócio (objetos de negócio) e não somente na implementação do sistema. E como ressalta Berard (1993) citado em Pressman (2002), os benefícios da orientação a objetos são ampliados se ela é adotada no inicio e ao longo de todo o processo de engenharia de software.

Na orientação a objetos, os dados são tratados como objetos ou um conjunto de objetos que possuem características ou atributos definidos pelos usuários. Isso porque esses objetos representam entidades do domínio de negócio. Para que possa ser utilizada com efetividade a visão orientada objetos exige uma abordagem evolutiva (PRESSMAN, 2002), como a fornecida pelo processo iterativo e incremental (SCOTT, 2003). Um modelo evolutivo vinculado a uma abordagem que facilita e incentiva o reuso é o melhor paradigma para a engenharia de software orientada a objetos (PRESSMAN, 2002).

Se os objetos representam entidades ou “coisas” do cenário ou contexto do que está sendo modelado, a orientação a objetos, como técnica para modelagem de sistemas, diminui a diferença semântica entre e a realidade sendo modelada o os modelos construídos (BEZERRA, 2006). A realidade representa o que será informatizado.

Em termos de código, o reuso está vinculado à orientação a objetos pelo mecanismo de herança. A orientação a objeto permite criar objetos a partir de outros objetos e cada qual pode ter atributos e comportamento específicos. Essa técnica se preocupa, principalmente,

com o comportamento do objeto. Contudo, a orientação a objetos facilita o reuso de componentes de código, entendidos como a implementação de funcionalidades quer estejam sob a forma de DLLs (*Dynamic-link library*), bibliotecas, rotinas ou outros.

O encapsulamento padroniza a forma de interação com esses componentes, protegendo os seus dados. O reuso também se aplica ao reuso de artefatos de análise e projeto que também é facilitado pelo uso da orientação a objetos. Artefatos se referem a tudo o que é produzido durante o ciclo de vida, seja direta ou indiretamente vinculado ao resultado. Documento de requisitos, diagramas de classe, scripts de banco de dados e planos de testes são considerados diretamente relacionados. Padronizações para documentação de código e *check-list* para verificação de requisitos são considerados indiretamente relacionados.

Utilizando diagramas da UML, as entidades que definem o problema e a solução são documentadas de maneira que elas possam mais facilmente ser reusados em outros projetos. Isso porque essas entidades são definidas como agrupamentos relacionados de dados associados ao seu comportamento.

Os conceitos principais relacionados à orientação a objetos: objeto e classe (incluindo os conceitos de identidade, atributo, método e mensagem), encapsulamento, polimorfismo e herança. A notação utilizada para representar graficamente esses conceitos é proveniente da UML. Outro conceito bastante importante na orientação a objetos é o de abstração.

Uma abstração significa um modelo que representa uma realidade de acordo com determinados interesses. Nesse sentido são representados apenas os aspectos considerados importantes ou relevantes para o objetivo da representação. A abstração permite gerenciar a complexidade e concentrar a atenção nas características essenciais dos objetos, ou seja, do que está sendo modelado utilizando o paradigma da orientação a objetos (BEZERRA, 2006).

a) Objeto

O mundo real é composto por elementos físicos ou conceituais, como, clientes, pedidos de compra, fórmula de cálculo de juros e carros, dentre muitos outros. Na terminologia da orientação a objetos, essas coisas do mundo real são denominadas objetos (BEZERRA, 2006).

Um objeto é “um conceito, abstração ou coisa com identidade que possui significado para uma aplicação” (BLAHA; RUMBAUGH, 2006, p. 23). Um objeto em termos de uma aplicação computacional pode ter uma representação correspondente no mundo real, como, por exemplo, João da Silva que representa um cliente para o sistema computacional em uma determinada empresa. Um objeto pode não ter representação correspondente no mundo real como a fórmula para o cálculo de juros compostos utilizada em um sistema financeiro. Os

objetos podem ser bastante distintos entre si e representados de maneiras distintas, mas todos os objetos possuem identidade e são distinguíveis uns dos outros.

Booch (1998) caracteriza um objeto por possuir estado, exibir comportamento bem definido e ter uma identidade única. O estado descreve as variáveis que contêm os dados que caracterizam um objeto. O comportamento representa os métodos ou as ações realizados sobre os dados. E a identidade está relacionada a fato que cada objeto é único e que existe uma forma de individualizá-lo. Por exemplo, uma empresa pode ter milhares de clientes, mas há algum dado (variável) que compõe o seu estado que permite individualizar cada cliente dos demais.

b) **Classe**

Uma classe descreve um grupo de objetos com as mesmas propriedades (atributos), comportamento (operações, métodos), tipos de relacionamentos e semântica (BLAHA; RUMBAUGH, 2006). A interpretação semântica depende do objetivo da classe em relação à aplicação. Por exemplo: pessoa e carro podem ser considerados como pertencentes a mesma classe em um sistema para seguradora. Isso porque ambos possuem um determinado valor de seguro. Em um sistema para uma locadora de veículos ou mesmo uma revendedora de automóveis, pessoa pode representar cliente ou mesmo funcionário e veículo ser uma classe completamente distinta porque é o objeto que será locado ou vendido/comprado.

Um objeto é uma instância ou uma ocorrência de uma classe. Uma classe instancia um objeto ou a partir de uma classe são originados os objetos. A classe define como o objeto é. E o objeto personifica os atributos e o comportamento definidos na classe. Assim, uma classe pode ser considerada um conjunto de objetos que possuem as características definidas na classe.

Na Figura 1 está representada uma classe denominada Cliente com os atributos nome, cpf, email, telefone endereço, numero e complemento. E os métodos: verificarcpf() e incluircliente(), excluircliente().

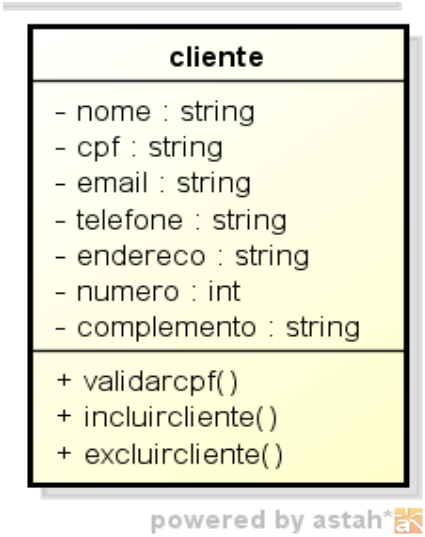

Figura 1 – Exemplo de Classe

Os atributos são os elementos que definem a estrutura de uma classe e armazenam as características dos objetos instanciados. Os atributos também são conhecidos como variáveis de classe. Cada atributo pode ser seguido por dados opcionais como o tipo e o valor padrão. A Figura 2 apresenta uma classe denominada “pessoa” com um atributo “nome”. Esse atributo possui o valor padrão “Nome 1”.

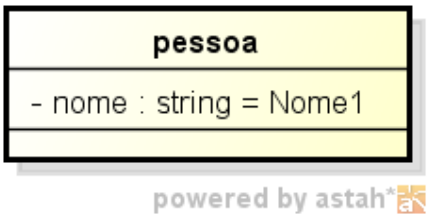

Figura 2 – Exemplo de Atributo

Os métodos definem as ações dos objetos. A ação só vai ocorrer se existir um objeto instanciado e se este objeto invocar (chamar ou acessar) o método. Os métodos manipulam os dados do objeto.

Os objetos se comunicam por meio de mensagens. Essas mensagens podem dizer que um método de um objeto necessita utilizar outro método de um objeto distinto. As mensagens podem ser utilizadas para passar objetos por parâmetro ou algum resultado obtido com a execução de um método. As mensagens podem invocar métodos, porém elas não podem dizer quando e como eles devem ser executados. Uma mensagem é uma solicitação para executar uma operação indicada sobre determinado objeto e retornar um resultado (MARTIN, 1995).

b) Encapsulamento

Encapsulamento significa que os dados e as operações que trabalham com esses dados são agrupados pelo objeto ou a classe. O encapsulamento é uma forma de proteger os dados de um objeto instanciados na classe. Além de controlar o acesso aos atributos e métodos, o encapsulamento torna os atributos e métodos de um objeto ocultos aos outros objetos.

O encapsulamento oferece os seguintes benefícios (PRESSMAN, 2002):

- Os detalhes internos da implementação dos dados e procedimentos (operações) são ocultados de acesso externo;
- As estruturas de dados e as operações que as manipulam são agrupadas em uma única entidade, a classe. O que facilita o reuso.
- As interfaces entre objetos encapsulados são simplificadas. Um objeto que envia uma mensagem não precisa ter conhecimento da estrutura interna dos dados.

c) Polimorfismo

O polimorfismo "indica a capacidade de abstrair várias implementações diferentes em uma única interface" (BEZERRA, 2006, p. 10). Interface estabelece a forma de comunicação, os parâmetros para troca de mensagens, entre os objetos. Isso no sentido de definir a interação com os objetos.

O polimorfismo ocorre quando há duas ou mais classes derivadas de uma superclasse e as classes derivadas invocam os métodos de classe abstrata (superclasse), porém com comportamentos diferentes em cada classe derivada (subclasse). Os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

Na Figura 3 está representada uma classe abstrata chamada "Impressão" e duas classes derivadas. As classes derivadas herdarão o método gerarimpressao() da classe pai e podem ter algumas características próprias. Se o método gerarimpressao() for invocado por um objeto do tipo "ArquivoPDF" a ação apesar de ser o mesmo método da classe "Impressora" terá suas características, representando o polimorfismo.

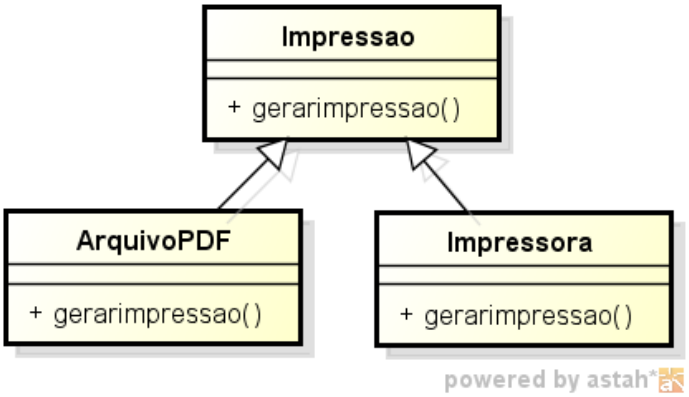

Figura 3 – Exemplo de Polimorfismo

h) Herança

A herança é utilizada com o propósito de reaproveitar atributos e métodos através de compartilhamento entre classes. O uso de herança em uma linguagem de programação orientada a objetos possibilita criar classes derivadas a partir de classes já existentes. Na herança, classes semelhantes são agrupadas em hierarquias, em que cada nível de uma hierarquia pode ser visto como um nível de abstração (BEZERRA, 2006).

Os termos herança, generalização e especialização referem-se a uma mesma ideia. Generalização e especialização tratam de um relacionamento entre classes e assumem pontos de vista opostos, da superclasse ou das subclasses. E generalização se refere a que a superclasse generaliza as subclasses (BLAHA; RUMBAUGH, 2006). E, ainda, segundo esses autores a especialização indica que as subclasses refinam ou especializam a superclasse. Ao passo que herança se refere ao mecanismo para compartilhamento de atributos, operações e associações por meio do relacionamento de generalização/especialização.

A Figura 4 apresenta um exemplo de herança. Nesse exemplo a classe “PessoaFisica” e a classe “PessoaJuridica”, além de definir atributos específicos, herdam os atributos da classe “Cliente”.

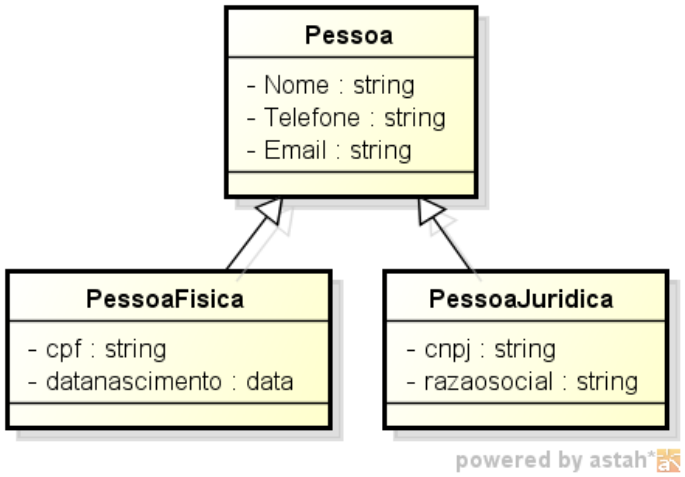

Figura 4 – Exemplo de Herança

2.2 Unified Modeling Language

A *Unified Modeling Language* é uma linguagem visual para especificar, visualizar, construir e documentar modelos de software orientados a objetos (BOOCH; RUMBAUGH; JACOBSON, 2000). E sendo uma linguagem visual ela é constituída de elementos gráficos que permitem representar os conceitos da orientação a objetos na modelagem de sistemas.

A UML é composta por elementos de modelo que representam as diferentes partes e funcionalidades de um sistema de software. Esses elementos representam, por meio de diagramas e desenhos, visões distintas do sistema, de acordo com os interesses dos envolvidos no projeto. Assim, cada diagrama representa pontos de vista diferentes do sistema. Cada elemento possui uma sintaxe (uma forma predeterminada de representar o elemento) e uma semântica (o significado definido para cada elemento e a definição do seu uso) (BEZERRA, 2006).

Essas visões permitem representar abstrações distintas de uma mesma realidade. No desenvolvimento de sistemas orientados a objetos essa realidade é o sistema modelado e posteriormente implementado em uma linguagem de programação. Modelos distintos são importantes porque se complementam na representação do sistema, mas atendem a interesses específicos, embora se refiram ao mesmo propósito.

2.1.1 Diagramas da UML

A UML permite representar diferentes visões de um sistema por meio de diagramas. Cada diagrama representa determinados aspectos do sistema, mas os diagramas possuem dependência entre si. A relação entre os diagramas deve manter a coerência do projeto. Blaha e Rumbaugh (2006) definem a modelagem de um sistema a partir de três pontos de vista distintos, mas relacionados, cada qual representando aspectos relevantes do sistema, mas todos necessários para uma descrição completa. Esses pontos de vista são: modelo de classes, modelo de estados e modelo de interações.

O modelo de classes representa os aspectos estáticos, estruturais, de dados de um sistema. As classes descrevem a estrutura dos objetos de um sistema: sua identidade, relacionamentos com outros objetos, seus atributos e operações. Esse modelo fornece o contexto para os modelos de estados e de interações (BLAHA; RUMBAUGH, 2006). A generalização permite que as classes compartilhem estrutura (dados) e comportamento (operações) e as associações relacionam as classes entre si.

O modelo de estados representa os aspectos temporais, comportamentais, de controle de um sistema. Esse modelo descreve os aspectos dos objetos que tratam do tempo e da sequência de operações e captura o controle (BLAHA; RUMBAUGH, 2006). Os estados são representados por diagramas de máquinas de estados que mostram a sequência de estados e eventos em permitidas em um sistema para uma classe de objetos.

O modelo de interações representa a colaboração de objetos de forma a alcançar o comportamento do sistema como um todo. O modelo de interações pode ser documentado por meio de casos de uso, diagramas de sequência e de atividades (BLAHA; RUMBAUGH, 2006), dentre outros.

A Figura 5 mostra os diagramas da UML. Essa representação hierárquica dos modelos foi feita de acordo com o exposto na especificação da superestrutura da UML definida pela OMG (*Object Management Group*). Nessa figura os diagramas de estados e de interações são representados como diagramas comportamentais.

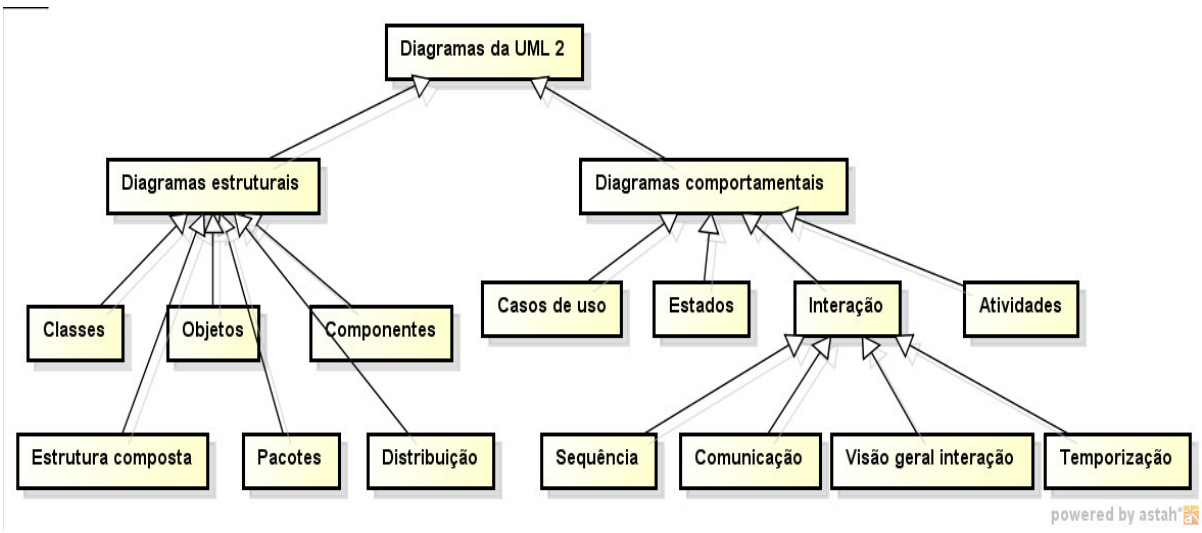

Figura 5 – Diagramas da UML 2

Fonte: composto a partir da especificação OMG 2.4.1 (2011).

A seguir são exemplificados alguns dos diagramas representados na Figura 5.

a) Diagrama de casos de uso

Diagramas de caso de uso descrevem relacionamentos e dependências entre um grupo de caso de uso e os atores participantes. Caso de uso representa uma funcionalidade completa de um sistema, subsistema ou classe, que é externamente perceptível (SILVA, 2007). O Ator representa qualquer entidade que interage com o sistema. Pode ser uma pessoa, um dispositivo fisco ou outro sistema. São características dos atores (MACORATTI, 2012):

- Não é parte do sistema. Representa os papéis que o usuário do sistema pode desempenhar;
- Pode interagir ativamente com o sistema;
- Pode ser um receptor passivo de informação.

Os casos de uso também podem se relacionar entre si. Os quatro tipos de relacionamento entre casos de uso são (BOOCH; RUMBAUGH; JACOBSON, 2000; SILVA, 2009)

- Associação - representa a interação do ator com o caso de uso. Essa associação representa participação da entidade externa modelada pelo ator no caso e uso a ele associado (SILVA, 2009).
- Generalização - um caso de herda o comportamento e o significado de outro caso de uso e o comportamento herdado deverá ser acrescido ou sobrescrito (BOOCH; RUMBAUGH; JACOBSON, 2006).

- Extensão - o caso de uso base incorpora implicitamente o comportamento de outro caso de uso em um local especificado indiretamente pelo caso de uso estendido (BOOCH; RUMBAUGH; JACOBSON, 2006). O caso de uso poderá permanecer isolado, mas sob certas condições seu comportamento poderá ser estendido pelo outro caso de uso. A extensão tem a conotação de ser ser relacionamento opcional (SILVA, 2009).
- Inclusão - Indica que um caso de uso base incorpora explicitamente o comportamento de outro caso de uso em uma localização especificada (BOOCH; RUMBAUGH; JACOBSON, 2006). Um relacionamento de inclusão tem a conotação de obrigatoriedade (SILVA, 2009).

A Figura 6 representa um exemplo de diagrama de caso de uso.

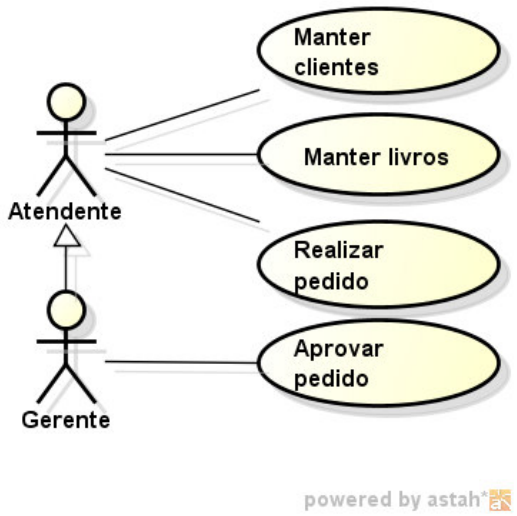

Figura 6 – Diagrama de casos de uso

b) Diagrama de Classes

Um diagrama de classes define as classes que compõe um sistema e como elas se relacionam. É um diagrama que representa a visão estática do sistema (BOOCH, RUMBAUGH; JACOBSON, 2000) porque representa na sua estrutura as informações dos métodos, atributos, nome das funções bem como os relacionamentos entre as classes.

As relações entre as classes podem ser (BOOCH; RUMBAUGH; JACOBSON, 2000):

- Herança - quando uma classe herda todos os atributos e as operações de outra classe. Esses atributos e operações herdados podem ser sobrepostos e modificados e podem ser adicionados outros atributos e operações próprias.
- Associação - representa um relacionamento entre classes.

- Agregação - é um tipo especial de associação em que o objeto é parte do outro sem necessariamente ser todo.
- Composição - é uma associação que representa a parte não existe sem o todo.

A Figura 7 mostra as classes Cliente e Livro que representam os cadastros básicos. A classe Pessoa representa uma generalização da classe Cliente, incluindo seu Endereço. A classe Pedido registra as compras realizadas pelos clientes. A classe ItemPedido representa os pedidos de Livros realizados por um cliente em um pedido.

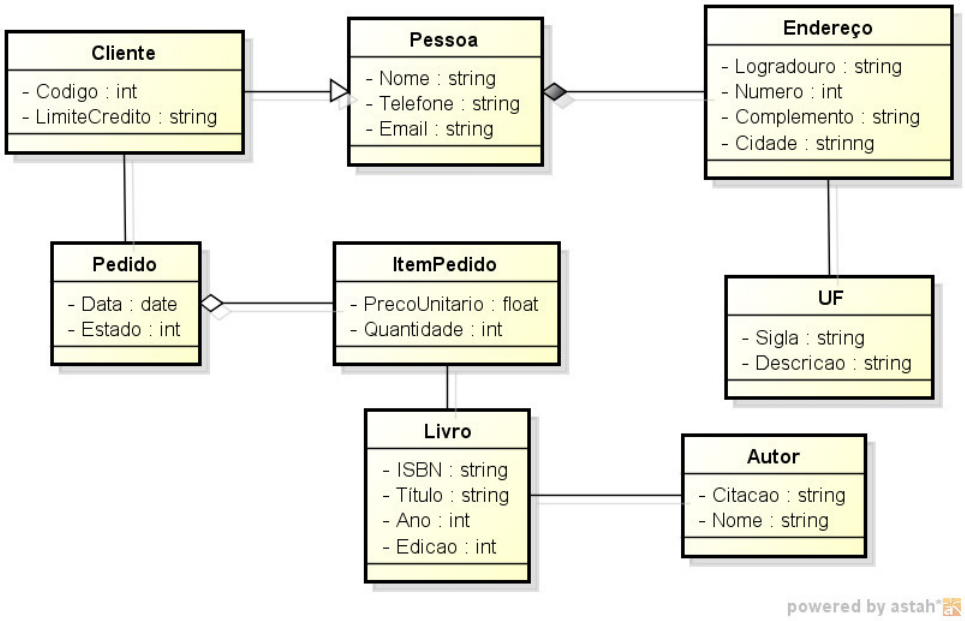

Figura 7 – Diagrama de classes

g) Diagrama de Objetos

Um diagrama de objetos mostra um conjunto de objetos e seus relacionamentos em um ponto no tempo (BOOCH; RUMBAUGH; JACONSON, 2000). Na Figura 8, os objetos Livro e Autor são mostrados após terem sido instanciados, apresentando os atributos valorizados.

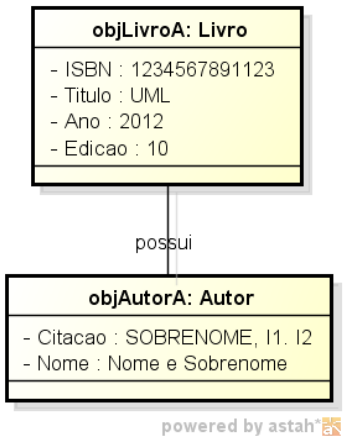

Figura 8 – Diagrama de objetos

c) Diagrama de atividades

Um diagrama de atividades mostra o fluxo de uma atividade para outra (BOOCH; RUMBAUGH; JACOBSON, 2000). É uma representação de um modelo sequencial de ações. Esse diagrama é utilizado para modelar o aspecto comportamental dos processos. É usado para apresentar uma sequência de atividades que são conectadas entre si por transações que mostram as dependências entre elas.

Na Figura 9 está representado um diagrama de atividades para processamento de um pedido de compras. O estado inicial da atividade como um círculo em cor preta. Em seguida, está a ação de receber o pedido. Dessa ação são originadas duas outras ações em execução em paralelo. Elas representam as ações possíveis de serem tomadas a partir do recebimento do pedido. Independentemente da ação tomada elas seguem para uma mesma ação na sequência. Isso e determinado por um ponto de junção. Da ação “Solicitar confirmação do pedido” duas ações podem ser decorrentes de uma decisão. Ou o pedido é confirmado ou é cancelado. Dependendo ação realizada a ação seguinte é tomada. Um ponto de tomada de decisões que confirma ou cancela o pedido, chegando ao estado final da atividade que é representado por um círculo de cor preta com borda dupla.

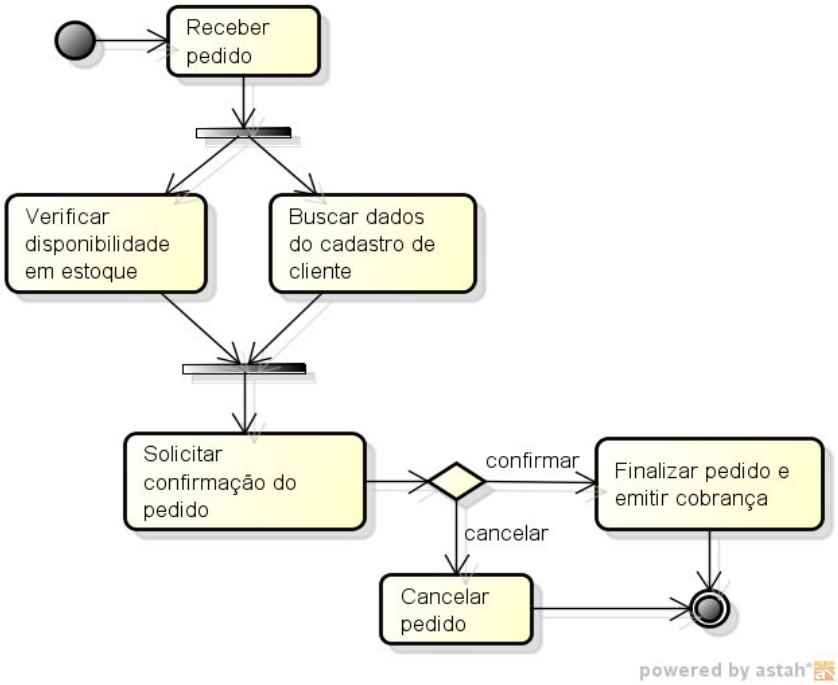

Figura 9 – Diagrama de atividades

d) Diagrama de sequência

O diagrama de sequência representa a sequência dos processos de um sistema de forma simples e lógica através de mensagens que são passadas entres os objetos. Esse diagrama mostra os participantes nas interações e a sequência das mensagens entre eles. Um diagrama de sequência mostra as interações de um sistema com os seus atores para realizar casos de uso completos ou partes dos mesmos (BLAHA; RUMBAUGH, 2006). A Figura 10 representa a troca de mensagens do cliente com os objetos e a sequência delas.

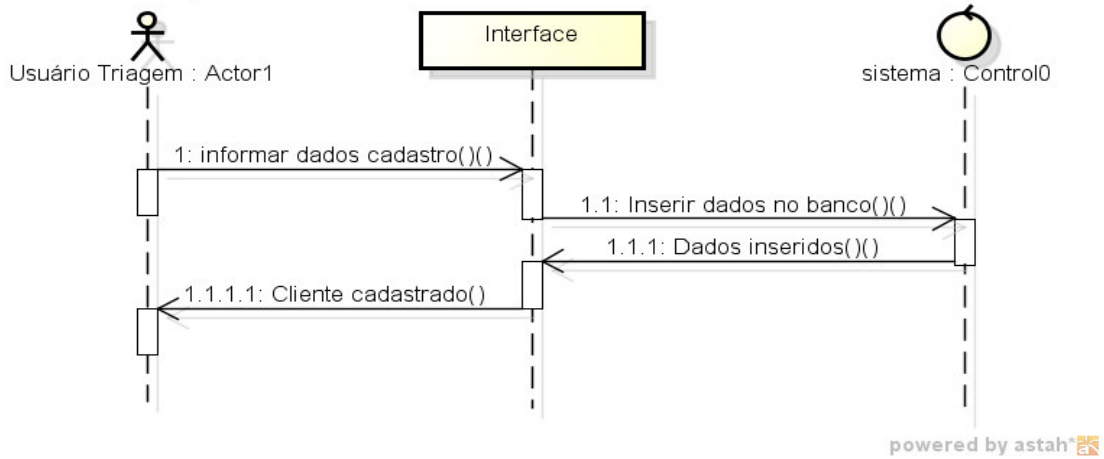

Figura 10 – Diagrama de sequência

e) Diagrama de Estado

O diagrama de estado, ou de máquina de estados, representa o ciclo de vida dos objetos e como eles são afetados pelos eventos (BOOCH; RUMABUGH; JACOBSON, 2000). Na Figura 11 é representado o ciclo de um pedido com seus possíveis estados e a transição de desligada para ligada.

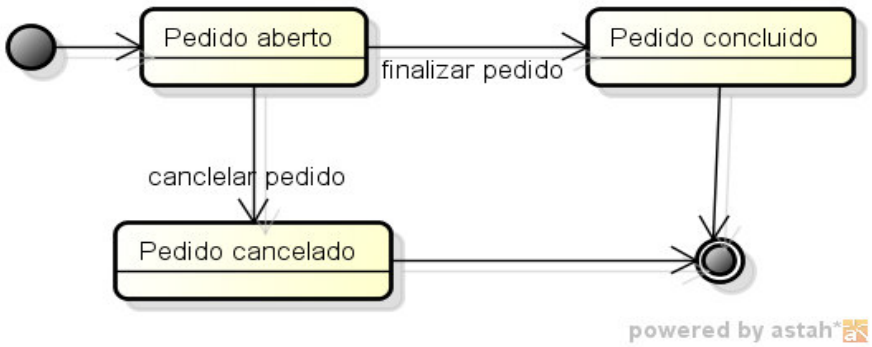

Figura 11 – Diagrama de máquina de estados

f) Diagrama de Pacotes

O Diagrama de pacotes descreve os agrupamentos lógicos do sistema mostrando as dependências entre eles, basicamente ilustra a arquitetura de um sistema mostrando o agrupamento de suas classes. Um pacote pode ser composto por vários elementos entre eles: classes, interfaces, componentes, casos de uso. Um pacote pode ser composto por outros pacotes. A Figura 12 mostra um exemplo de diagrama de pacotes.

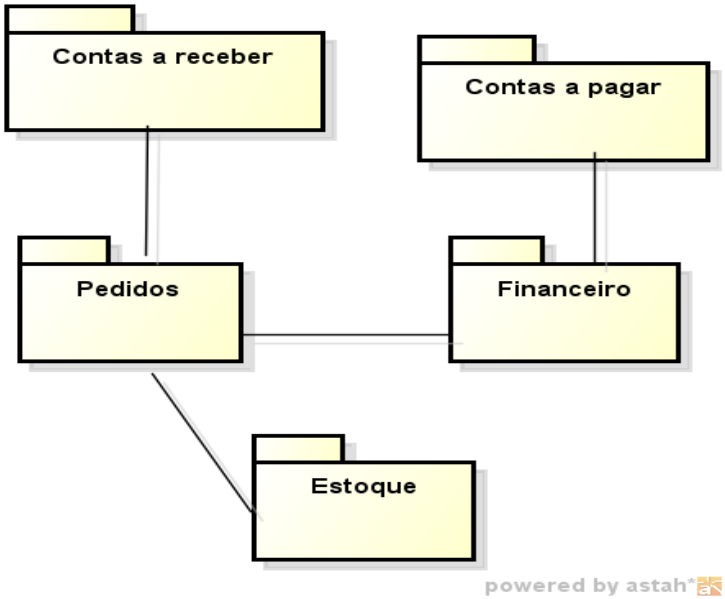

Figura 12 – Diagrama de pacotes

g) Diagrama de Componentes

O Diagrama de componentes ilustra a organização das classes e tem o objetivo de apresentar a disposição dos componentes físicos dentro de um sistema. A Figura 13 apresenta o tipo do componente representado pelo estereótipo da UML.

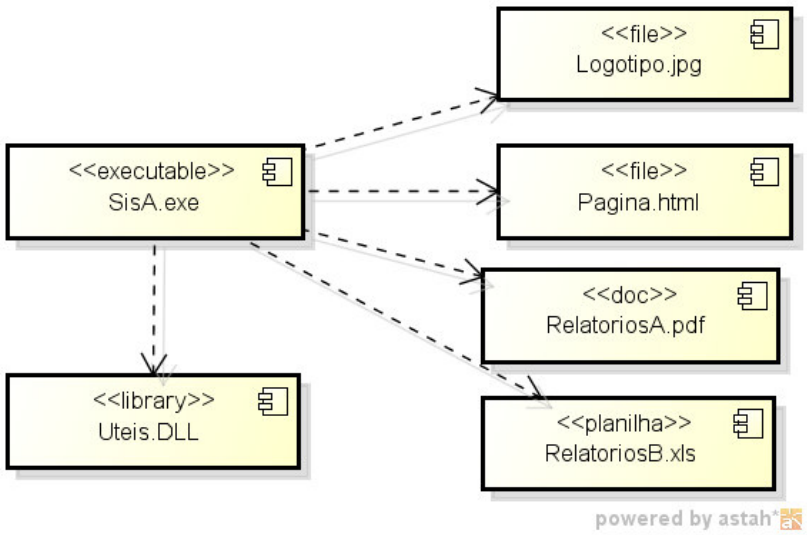

Figura 13 – Diagrama de componentes

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais utilizados no projeto e o método empregado para a realização das atividades.

3.1 MATERIAIS

As ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema foram as seguintes.

a) AstahCommunity - para modelagem do sistema;

b) NetBeans 7.0.1 - IDE (*Integrated Development Environment*) para desenvolvimento;

c) Java - linguagem de programação;

d) MySQL - banco de dados;

e) MySQL - Front - gerenciador do banco de dados;

f) Equipamento RouterBoard 433 com sistema RouterBoard.

3.3.1 Astah Community

Astah Community (ASTAH, 2012) é uma ferramenta gratuita para modelagem de sistemas orientados a objeto. Essa ferramenta é baseada em diagramas definidos com a notação da UML 2.0.

Com a ferramenta Astah é possível desenhar os diagramas necessários para representar um projeto orientado a objetos de forma fácil. A escolha da ferramenta Astah deu-se devido a esse motivo e por ser um software gratuito.

A Figura 14 apresenta a tela principal da ferramenta Astah Community utilizada para modelagem de projetos.

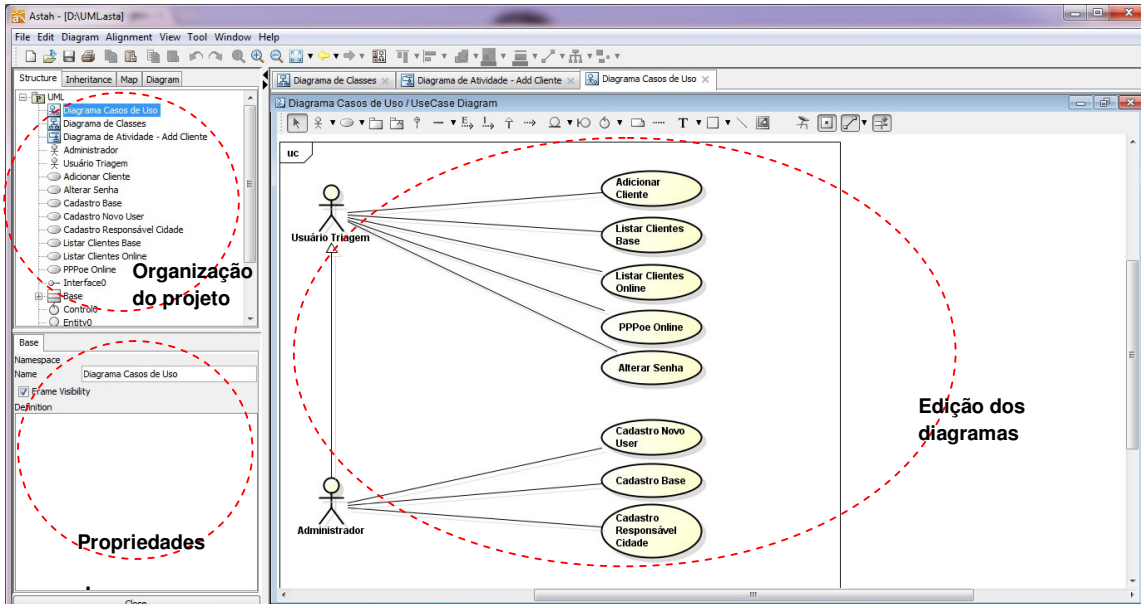

Figura 14 – Tela principal do Astah Community

As partes principais da ferramenta Astah Community marcadas na Figura 14 são:

a) Organização do projeto – apresenta os arquivos e os elementos do projeto de acordo com visões distintas que são:

a.1) structure – é a árvore de estrutura do projeto composta por pastas, diagramas e elementos de diagramas;

a.2) *inheritance* – exibe as heranças identificadas entre os elementos definidos;

a.3) *map* – apresenta todo o diagrama em edição e possibilita definir a área a ser visualizada;

a.4) *diagram* – apresenta a lista de diagramas do projeto em edição.

b) Propriedades dos elementos – apresentada as propriedades do elemento selecionado do diagrama.

c) Editor dos diagramas – é a área para composição dos diagramas e para edição de diagramas. A barra de ferramentas que fica na parte superior dessa área apresenta os elementos para composição diagramas. Esses elementos são apresentados de acordo com o tipo de diagrama selecionado para edição.

3.3.2 NetBeans

A IDE Netbeans (NETBEANS, 2012) é um ambiente de desenvolvimento utilizado para produzir código fonte de acordo com a sintaxe e a semântica de linguagens como Java,

JavaScript, PHP e Phtyton, embora esta seja na maioria das vezes utilizada para o desenvolvimento. Embora está muito associada ao desenvolvimento Java. É um ambiente de desenvolvimento multiplataforma com recursos para editar, compilar e depurar programas e gerar instaladores. A IDE NetBeans facilita o trabalho de desenvolvimento de aplicativos por possuir um grande conjunto de bibliotecas, módulos e APIs que são basicamente um conjunto de rotinas, protocolos e ferramentas para desenvolvimento. A Figura 15 apresenta a tela principal da IDE NetBeans.

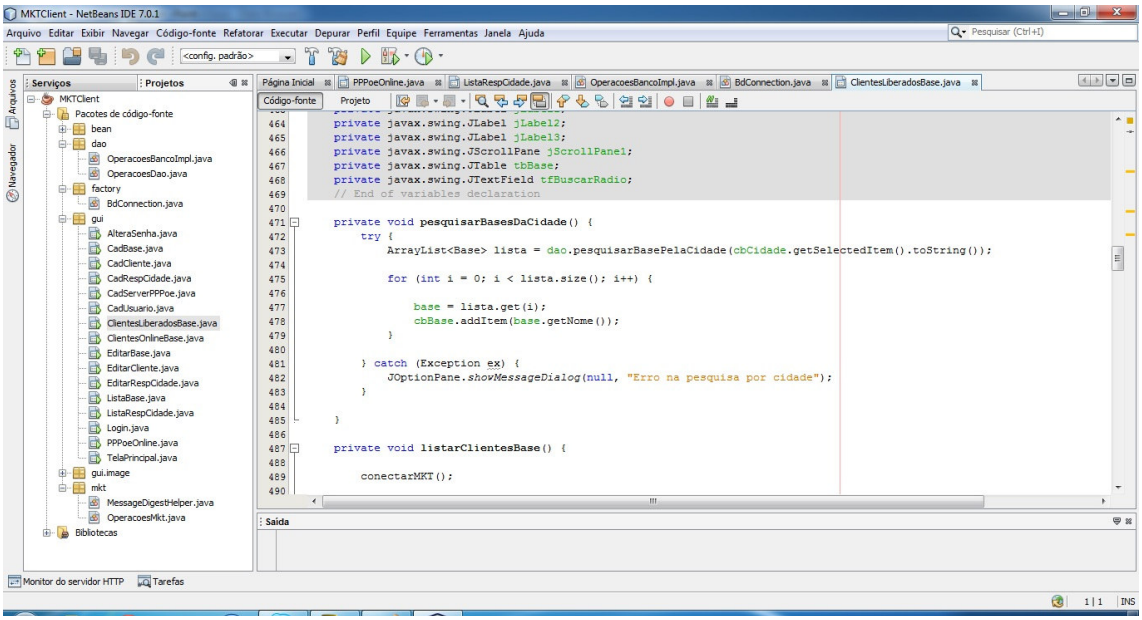

Figura 15 – IDE NetBeans

Na Figura 15, na área mais a esquerda estão os artefatos de código produzido. Eles ficam organizados em pastas. Na área central e a direita da figura está o editor de código. E na parte superior da área de edição está uma barra de ferramentas e abas com os arquivos em edição.

3.3.3 Linguagem Java

Java é uma linguagem orientada a objetos em que a maior parte dos elementos de um programa Java são objetos, também é denominada plataforma ou tecnologia. Os tipos básicos, como *int* e o *float* não são objetos. O código é organizado em classes, que podem estabelecer somente relacionamentos de herança simples entre si. Como os programas Java executam em

uma máquina virtual, os programas criados nessa linguagem podem ser executados em qualquer sistema operacional que suporte a linguagem C++. A máquina virtual Java é um compilador C/C++.

A linguagem Java provê o gerenciamento de memória por meio de *garbage collection* (coleta de lixo). Sua função é a de verificar a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Esse procedimento pode deixar o sistema de *software* com execução mais demorada por manter uma *thread* paralela à execução do programa. Porém, evita problemas como referências perdidas e avisos de falta de memória quando ainda há memória disponível na máquina.

A tecnologia Java é dividida em três versões básicas:

a) *Java Standard Edition* (Java SE) - é a tecnologia Java para computadores pessoais, portáteis e arquiteturas com capacidade de processamento e memória consideráveis. Várias APIs acompanham essa versão e tantas outras podem ser obtidas no site da Sun (http://www.sun.com). JSE possui duas divisões:

a.1) *Java Development Kit* (JDK) ou *Standard Development Kit* (SDK) - um conjunto padrão de recursos para desenvolvimento em Java.

a.2) *Java Runtime Edition* (JRE) - uma versão do JDK que executará os sistemas construídos com o SDK.

b) *Java Micro Edition* (Java ME) – é a tecnologia Java para dispositivos móveis com limitações de memória e/ou processamento. Possui APIs que visam otimizar o uso de espaço, de memória e de processamento. Java ME se divide em dois grupos de bibliotecas:

b.1) *Connected Limited Device Configuration* (CLDC) – para dispositivos móveis mais limitados em recursos como celulares e *smartphones*.

b.2) *Connected Device Configuration* (CDC) – para dispositivos com mais recursos como *palmtops* e *pocket pcs*.

c) *Java Enterprise Edition* (Java EE) - para aplicações corporativas que podem estar na Internet. JEE possui um grande número de APIs. É ideal para servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros.

3.3.4 MySQL

O MySQL é um servidor e gerenciador de banco de dados relacional que utiliza a linguagem SQL (*Structured Query Language*) (MYSQL, 2012). Ele possui licença livre e

paga. E foi inicialmente projetado para trabalhar com aplicações de pequeno e médio portes, mas atualmente atende também aplicações de grande porte.

As principais características incorporadas na versão 5 do MySQL (MILANI, 2007) são:

a) Visões - são consultas pré-programadas ao banco de dados que permitem unir duas ou mais tabelas e retornar uma única tabela como resultado. As visões podem ser utilizadas para filtrar informações de tabelas. É possível também utilizá-las para permitir que determinados usuários acessem dados de uma visão, mas não as tabelas utilizadas para compor a referida visão, restringindo acesso a informações.

b) Cursores - cursores possibilitam a navegação em conjuntos de resultados. É possível navegar pelos registros de uma tabela a partir de estruturas de repetição, permitindo realizar operações necessárias e transações à parte para cada linha da tabela.

c) *Information Schema* - são tabelas responsáveis apenas pela organização dos recursos do banco de dados, conhecidos como dicionário de dados, ou metadados.

d) Transações distribuídas XA - são uma espécie de extensão da ACID (Atomicidade, Consistência, Isolamento, Durabilidade) que fornece a possibilidade de gerenciamento das transações realizadas com a união de bancos de dados (transações distribuídas) para a execução de uma mesma transação.

e) Integridade referencial - os relacionamentos entre tabelas são gerenciados pelo banco de dados na realização de operações de inclusão, alteração e exclusão.

f) Clusterização - a clusterização é baseada na integração e sincronismo de dois ou mais servidores para dividirem a demanda de execuções entre si e o balanceamento de carga.

3.3.5 MySQL-Front

MySQL-Front (MYSQL-FRONT, 2012) é um software Windows *front-end* para o servidor de banco de dados MySQL. A estrutura da base de dados e os dados podem ser manipulados por meio de diálogos ou comandos SQL. É possível importar e exportar dados em diversos formatos. O acesso ao MySQL Server pode ser feito diretamente ou por meio de HTTP (*Hypertext Transfer Protoco*). A Figura 16 apresenta a tela principal da ferramenta MySQL-Front.

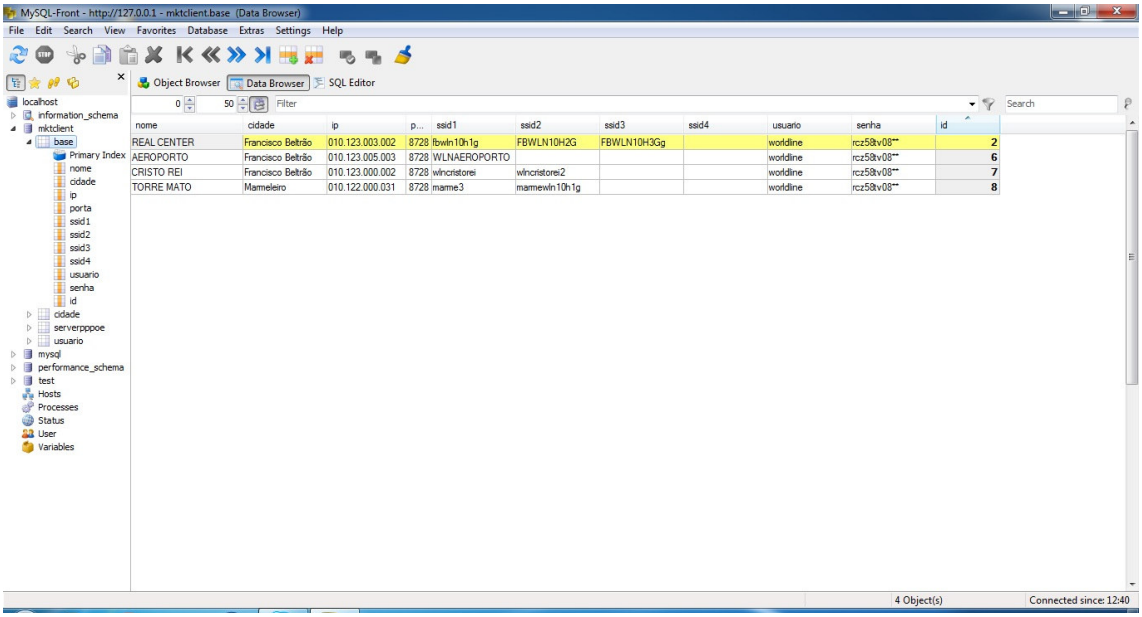

Figura 16 – IDE MySQL-Front

Na área à esquerda da Figura 16 fica a estrutura de pastas com os arquivos. Os arquivos são tabelas do banco de dados, por exemplo. Na área à direita está a região de edição das tabelas.

3.3.6 Equipamento RouterBoard

Fundada em 1995, a Mikrotik é uma empresa da Letônia que fabrica equipamentos para redes de computadores, esses equipamentos são muito utilizados por provedores de acesso a internet. O principal produto da empresa é o sistema operacional baseado em Linux chamado de RouterOS. Esse sistema operacional permite que os equipamentos fabricados pela empresa ou qualquer computador se torne um roteador com funções como *firewall*, VPN (*Virtual Private Network*), Proxy, QoS, protocolos de roteamento do tipo BGP (*Border Gateway Protocol*), RIP (*Routing Information Protocol*), OSPF (*Open Shortest Path First*), MPLS (Multiprotocol Label Switching), VPLS (*Virtual Private LAN Service*). O nível da licença do sistema determinará as funções liberadas para configuração.

O sistema operacional RouterOS pode ser administrado através de terminais SSH, Telnet, página *web* ou pelo software de configuração WinBox. A Figura 17 apresenta a interface do software Winbox desenvolvido pela empresa Mikrotik.

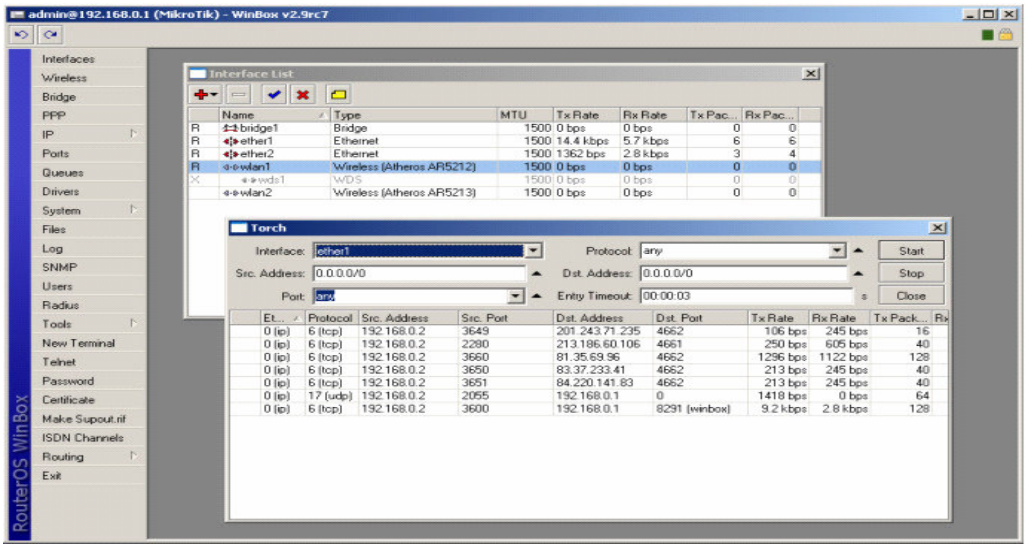

Figura 17 – IDE Winbox da Mikrotik

Algumas opções de configuração do Winbox não são disponíveis nessa interface apenas em terminais Telnet ou SSH (*Secure Shell*). O equipamento fabricado pela empresa combinado com o sistema operacional RouterOS leva o nome de RouterBoard. A Figura 18 representa um RouterBoard 433.

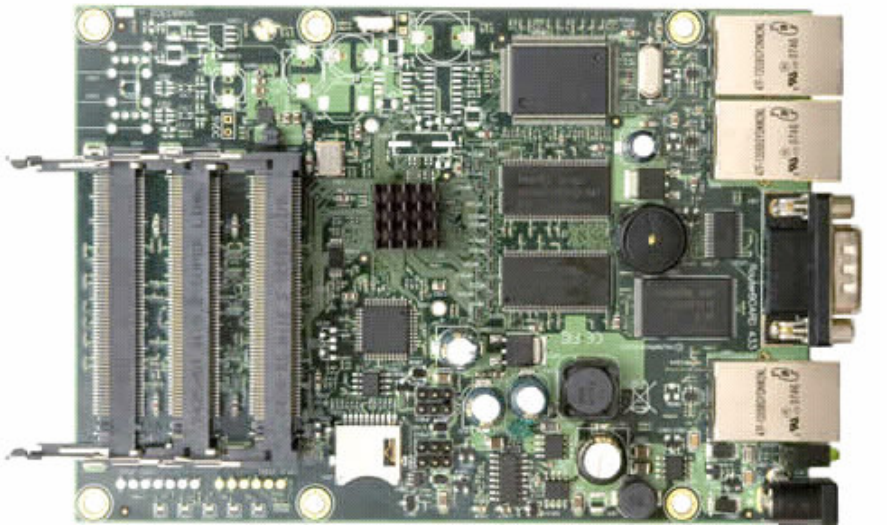

Figura 18 – Equipamento RouterBoard

Esse equipamento é muito utilizado pelos provedores de acesso para fazer *links wireless*. O equipamento possui três *slots* MiniPCI onde são conectados os cartões *wireless* para fazer *links* de rádio. A empresa também possui várias outras soluções em equipamentos

como antenas, roteadores com portas SFP (*Small Form-Factor Pluggable*) entre outros. Todos os produtos RouterBoard são agregados ao Sistema Operacional RouterOS.

3.2 MÉTODO

O projeto para a modelagem do sistema para interação com o equipamento RouterBoard foi desenvolvido tendo como base a empresa onde o autor deste trabalho é funcionário. Para a realização do trabalho algumas etapas foram organizadas e seguidas, sendo elas planejamento, embasamento teórico, levantamento de requisitos, análise e projeto do sistema.

O planejamento esteve voltado para a definição da forma de levantamento dos requisitos e do referencial teórico necessário para fundamentar o sistema que seria modelado.

O embasamento teórico ou levantamento bibliográfico foi fundamental para o desenvolvimento do projeto, porque permitiu a revisão de conceitos de orientação a objetos.

Os diagramas criados a partir da ferramenta Astah facilitaram a compreensão do projeto porque forneceu a visão geral do projeto a partir de pontos de vista distintos. A UML foi bastante útil para organizar e documentar o projeto. Os diagramas de casos de uso, classes e atividades forneceram esses pontos de vista. Considerando que a linguagem Java foi a escolhida para implementar o projeto, o estudo da orientação a objeto trouxe o conhecimento necessário para saber lidar com os objetos de *software*.

O levantamento de requisitos ocorreu por meio de entrevistas com funcionários da área na qual o software vai abranger e observações de processos. E pelo conhecimento do autor deste trabalho, que também é funcionário da referida empresa.

A empresa World Line Net atua na região do Sudoeste do Paraná como provedor de acesso à Internet. O autor sendo funcionário dessa empresa identificou falhas de segurança e a dificuldade dos funcionários do setor de triagem em lidar com um sistema proprietário Mikrotik chamado de RouterOS que normalmente é instalado em dispositivos eletrônicos de rede.

Buscando colher informações, o autor deste trabalho conversou com funcionários da triagem de ordem de serviço que serão os principais beneficiados com o desenvolvimento do projeto. Essas pessoas foram indagadas sobre procedimentos utilizados para realizar a sua atividade.

O autor procurou saber quais as maiores dificuldades desses funcionários na realização das suas funções e como um *software* poderia auxiliá-los. Todas as informações adquiridas foram utilizadas para identificar os requisitos do sistema.

A observação dos processos foi realizada por meio de acompanhamento das atividades e da sequência de processos. A observação também auxiliou na identificação dos requisitos.

Diagramas de casos de uso foram modelados com base nos requisitos levantados apresentando uma visão geral do sistema. Os diagramas de classes, atividades e sequência também foram criados para melhor visualizar o sistema que será desenvolvido.

4 MODELAGEM DO SISTEMA

Neste capítulo é apresentada a modelagem resultante das atividades de análise e projeto realizadas. Essa modelagem representa o software que manipulará as informações e as ações em um equipamento denominado RouterBoard. Esse equipamento possui o sistema operacional RouterOS e é proprietário da empresa Mikrotik.

4.1 A EMPRESA

A empresa World Line Net foi o primeiro provedor de acesso à internet de Francisco Beltrão. Desde sua fundação em 1996 a empresa sempre buscou as melhores soluções em internet para atender seus clientes de forma satisfatória e eficiente. Atualmente levando o nome de WI Telecom a empresa se tornou o maior provedor de internet banda larga do Paraná atuando em mais de 160 cidades.

Preocupada com a qualidade dos serviços oferecidos a empresa World Line Net está em constante atualização das tecnologias utilizadas e capacitação de seus funcionários.

A Internet por ondas de rádio é uma forma eficaz de levar internet em locais mais distantes dos grandes centros com ótimas opções de velocidades. Soluções de *firewall*, hospedagem, *e-mail*, *links* dedicados através de fibra óptica também são oferecidos pelo WLN/W.i.

4.2 MODELAGEM DO SISTEMA

O sistema modelado tem como objetivo manipular as ações e as informações dentro de um equipamento que rede, o RouterBoard, que possui um sistema operacional proprietário da Mikrotik, baseado em Linux, chamado de RouterOS.

Por intermédio do sistema a ser desenvolvido o usuário poderá adicionar, remover e editar clientes de uma lista de acesso que fica na memória do equipamento citado, bem como obter informações de clientes conectados. Cadastros de responsáveis técnicos e equipamentos também serão criados e salvos em banco de dados.

A Figura 17 apresenta o diagrama de casos de uso. Foram identificados três autores. O usuário Administrador com acesso a todas as funcionalidades do sistema. O usuário triagem com funcionalidades relacionadas a clientes e o usuário Routerboard (o próprio equipamento).

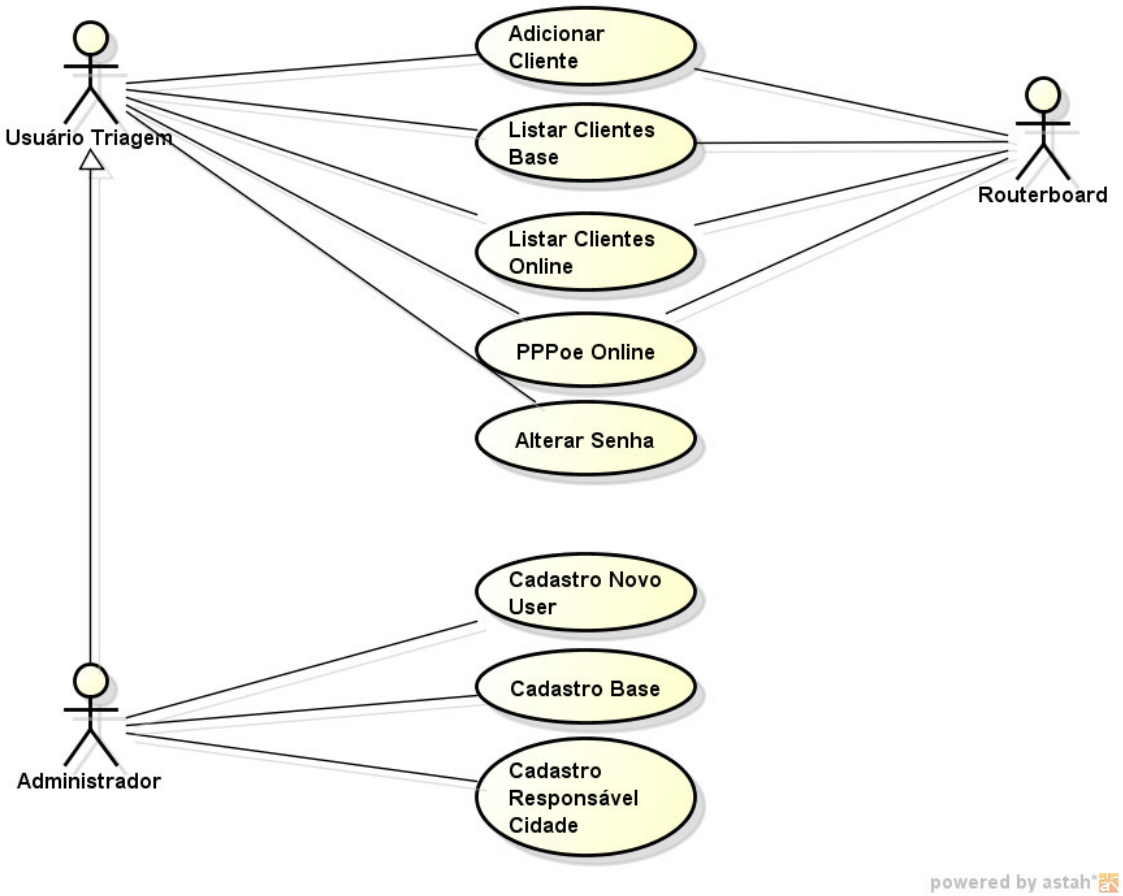

Figura 19 – Diagrama de casos de uso

No Quadro 1 está a descrição de cada caso de uso do modelo de sistema.

Caso de uso	Descrição
Adicionar Clientes	Através da interface do sistema o usuário vai escolher a cidade, base e interface para cadastrar o MAC (*Media Access Control*) e nome do cliente no equipamento que está na torre. A base se refere a um equipamento RouterBoard. O MAC identifica o equipamento do cliente. Inclusão – Incluir MAC e nome do cliente no *access list* do equipamento *routerboard* que esta na torre.
Listar Clientes Base	O usuário identifica a cidade e a base, o sistema deve conectar no equipamento que esta na torre e listar o nome dos clientes e respectivas MACs que estão liberados no *access list* do equipamento. Editar – Editar o registro de algum cliente dentro do *access list* do equipamento. Exclusão – Excluir o registro do cliente do *access list*. Habilitar – Habilitar o cliente no *access list*. Desabilitar – Desabilitar o cliente no *access list*. Busca – Buscar algum cliente por nome.
Listar Clientes Online	O usuário identifica a cidade e a base, o sistema deve conectar no

	equipamento que esta na torre e listar o nome dos clientes e respectivas MACs que estão conectados na base neste momento. Derrubar – Derrubar a conexão do cliente. Buscar – Buscar algum cliente por nome.
PPPoe Online	O sistema deve listar na tela os Servidores de PPPoe (*Point-to-Point Protocol over Ethernet*) que estão cadastrados, selecionando um servidor PPPoe o sistema deve listar na interface os clientes PPPoe que estão conectados no momento. Inclusão – Cadastrar servidores PPPoe. Exclusão – Excluir servidores PPPoe. Buscar – Buscar algum cliente PPPoe por nome.
Alterar Senha	Alteração da senha pessoal de acesso ao sistema.
Cadastro de Usuário	O sistema deverá cadastrar os usuários que utilizarão o sistema, definindo se administrador ou não. Inclusão – Cadastrar usuários no sistema.
Cadastro Base	O sistema deverá cadastrar as bases no sistema Inclusão – cadastrar base no sistema. Listar – Listar as bases cadastradas. Exclusão – Excluir bases do sistema. Alteração – Alterar algum dado da base no sistema. Detalhe – Apresentar alguma informação extra da base.
Cadastro Responsável Cidade	O sistema devera cadastrar o responsável pela cidade. Inclusão – cadastrar responsável pela cidade no sistema. Listar – Listar responsáveis. Exclusão – Excluir responsável pela cidade do sistema. Alteração – Alterar algum dado do responsável pela cidade no sistema. Detalhe – Apresentar alguma informação extra do responsável pela cidade.

Quadro 1 – Casos de uso do sistema

Nos quadros a seguir são apresentados os requisitos funcionais e não-funcionais detalhados de todos os casos de uso do sistema.

O Quadro 2 mostra o requisito funcional Cadastrar MAC de Clientes. Esse requisito funcional possibilitará ao usuário adicionar a MAC e o nome do cliente a lista de acessos do equipamento escolhido na interface. Os requisitos não funcionais também estão representados.

F1 Cadastrar MAC de Clientes	**Oculto ( )**
Descrição: O sistema apresentará na interface da tela um GroupBox Cidade, escolhendo a cidade o sistema preencherá outro GroupBox Base com as bases dessa cidade. O usuário escolhe a base e a interface da base, preenche os campos de MAC e nome do usuário. O sistema deverá conectar no equipamento escolhido por uma API (*Application Programming Interface*) e cadastrar a MAC e usuário no *access list* do equipamento escolhido. O sistema cadastrará o	

nome do usuário e MAC em banco de dados.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 1.1 Identificação de cidade, base e interface.	O sistema deve identificar a cidade escolhida dentro do GroupBox, as bases que esta cidade possui e dentro do GroupBox interface deverá preencher com os SSIDs da base escolhida no campo anterior.	Segurança	( )	(X)
NF 1.2 Conexão	O sistema deverá conectar na base através de uma API utilizando de um IP (*Internet Protocol*) e porta especifica para cada base.	Segurança	( )	(X)
NF 1.3 Campo em branco	O sistema não permitirá que seja cadastrado um cliente com o campo MAC em branco ou com MAC inválida.	Segurança	( )	(X)
NF 1.4 Janela Única	Todas as funções relacionadas ao cadastro de clientes devem ser efetuadas em uma única janela.	Interface	(X)	( )

Quadro 2 – Requisito cadastro de clientes

O Quadro 3 apresenta o requisito funcional Listar Clientes Base. Esse requisito possibilitará ao usuário listar os clientes que estão liberados dentro da lista de acesso do equipamento e será possível editar e excluir clientes através dessa lista. Adicional ao requisito funcional está à lista de requisitos não-funcionais associados.

F2 Listar Clientes Base		**Oculto** ( )		
Descrição: O usuário selecionará a cidade e a base que será acessada. O sistema acessará essa base através de uma API com um IP e porta especifica da base. O sistema listará as MACs e informações de cada cliente que esta cadastrado no *access list* do equipamento em questão.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**

NF 2.1 Identificação base.	O sistema deverá identificar qual foi a base escolhida dentro do GroupBox para fazer o acesso.	Segurança	( )	(X)
NF 2.2 Conexão	O sistema deverá conectar na base através de uma API utilizando de um IP e porta especifica da base.	Segurança	( )	(X)
NF 2.3 Busca de Cliente por Nome	O sistema permitirá fazer uma busca por nome dentro dos clientes listados na tela.	Segurança	( )	(X)
NF 2.4 Editar e Excluir cadastro do cliente.	O sistema deve permitir alterar e excluir cadastros que estão dentro do *access list* do equipamento.	Segurança	( )	(X)
NF 2.5 Janela Única	Todas as funções relacionadas à lista de clientes devem ser efetuadas em uma única janela.	Interface	(X)	( )

Quadro 3 – Requisito listar clientes base

O requisito funcional Listar Clientes Online está descrito no Quadro 4. O usuário poderá verificar quais são os clientes que estão *online* no equipamento escolhido na interface do sistema. A lista dos requisitos não-funcionais é apresentada nesse mesmo quadro.

F3 Listar Clientes Online	**Oculto** ( )			
Descrição: O usuário selecionará a cidade e a base que será acessada. O sistema acessará essa base através de uma API com um IP e porta especifica da base e listar as MACs e informações de cada cliente que está conectado na base no momento.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 3.1 Identificação base.	O sistema deverá identificar qual foi a base escolhida dentro do GroupBox para fazer o acesso.	Segurança	( )	(X)
NF 3.2 Conexão	O sistema deverá conectar na base através de uma API utilizando de um IP e porta especifica da base.	Segurança	( )	(X)

NF 3.3 Busca de Cliente por Nome	O sistema permitirá fazer uma busca por nome dentro dos clientes listados na tela.	Segurança	( )	(X)
NF 3.4 Derrubar conexão	O sistema deve derrubar a conexão do cliente quando clicado no botão correspondente.	Segurança	( )	(X)
NF 3.5 Janela Única	Todas as funções relacionadas à lista de clientes *online* devem ser efetuadas em uma única janela.	Interface	(X)	( )

Quadro 4 – Requisito listar clientes online

O Quadro 5 descreve os requisitos funcionais e não-funcionais do PPPoe *online*.

F4 PPPoe Online	**Oculto** ( )			
Descrição: O sistema listará na tela os Servidores de PPPoe que estão cadastrados, selecionando um servidor PPPoe. O sistema deve listar na interface os clientes PPPoe que estão conectados no momento. Cadastro e exclusão de servidores são possíveis dentro dessa interface. O sistema terá um mecanismo de busca por nome aos clientes PPPoe.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 4.1 Identificação do servidor selecionado	O sistema deve identificar o servidor selecionado e listar na interface os clientes PPPoe que estão conectados nesse servidor.	Segurança	( )	(X)
NF 4.2 Conexão	O sistema deverá conectar no Servidor PPPoe através de uma API utilizando de um IP e porta especifica.	Segurança	( )	(X)
NF 4.3 Adicionar, Editar e Excluir cadastro de Servidores PPPoe	O sistema deve permitir adicionar, alterar e excluir cadastros de Servidores PPPoe.	Segurança	( )	(X)
NF 4.4 Janela Única	Todas as funções relacionadas ao cadastro de clientes devem ser efetuadas em uma única janela.	Interface	(X)	( )

NF 4.5 Validar campo IP.	O sistema deve validar o IP digitado na hora do cadastramento do servidor PPPoe.	Segurança	( )	(X)

Quadro 5 – Requisito PPPoe online

O usuário pode alterar a senha de seu acesso através do sistema. O Quadro 6 contém o requisito funcional e os requisitos não-funcionais para Alterar a Senha.

F5 Alterar Senha		**Oculto** ( )		
Descrição: O sistema poderá alterar a senha do usuário logado.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 5.1 Identificar usuário logado	O sistema deverá identificar qual é o usuário que está logado e solicitando alteração de senha.	Segurança	( )	(X)
NF 5.2 Verificar senha atual	O sistema deverá verificar se a senha atual digitada corresponde à senha do usuário logado.	Segurança	( )	(X)
NF 5.3 Confirmar nova senha	O sistema deverá verificar se a nova senha digitada é igual nos campos senha e confirmação de senha.	Segurança	( )	(X)
NF 5.4 Janela Única	Todas as funções relacionadas à alteração de senha devem ser efetuadas em uma única janela.	Interface	(X)	( )

Quadro 6 – Requisito alterar senha

O requisito funcional Cadastrar Usuário está apresentado no Quadro 7. Esse requisito possibilitará ao administrador cadastrar novos usuários para o sistema. Os requisitos não funcionais também estão representados nesse quadro.

F6 Cadastro de Usuário	**Oculto** ( )
Descrição: O sistema poderá cadastrar um novo usuário para usar o sistema. Este usuário poderá ser administrador ou usuário padrão.	

Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 6.1 Confirmar nova senha	O sistema deverá verificar se a nova senha digitada é igual nos campos senha e confirmação de senha.	Segurança	( )	(X)
NF 6.2 Janela Única	Todas as funções relacionadas a cadastro de usuário devem ser efetuadas em uma única janela.	Interface	(X)	( )

Quadro 7 – Requisito cadastrar usuário

O Quadro 8 apresenta o requisito funcional Cadastrar Base. Esse requisito possibilitará ao usuário cadastrar as bases existentes em cada cidade. Adicional ao requisito funcional está a lista de requisitos não-funcionais associados.

F7 Cadastrar Base	**Oculto** ( )			
Descrição: O sistema vai permitir cadastrar as bases no banco de dados.				
Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 7.1 Validar campo IP.	O sistema deve validar o IP digitado no momento do cadastramento para verificar se o mesmo é válido.	Segurança	( )	(X)
NF 7.2 Listar Bases Existentes	O sistema deve, através de um botão, listar em uma nova tela as bases que já estão cadastradas no sistema.	Segurança	( )	(X)
NF 7.3 Editar e Excluir cadastros	O sistema deve permitir a alteração e exclusão de cadastros das bases.	Segurança	( )	(X)

Quadro 8 – Requisito cadastrar base

No Quadro 9 está o requisito funcional Cadastrar Responsável Cidade e seus respectivos requisitos não-funcionais.

F8 Cadastrar Responsável Cidade	**Oculto** ( )
Descrição: O sistema permitirá cadastrar os responsáveis pelas cidades onde a WLN atua no banco de dados.	

Requisitos não-funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 8.1 Campos formatados.	Os campos de telefone estão formatados para evitar erros de digitação.	Interface	( )	(X)
NF 8.2 Listar Responsáveis Existentes	O sistema deve através de um botão listar em uma nova tela os responsáveis que já estão cadastradas no sistema.	Segurança	( )	(X)
NF 8.3 Editar e Excluir cadastros	O sistema deve permitir a alteração e exclusão de cadastros dos responsáveis.	Segurança	( )	(X)

Quadro 9 – Requisito cadastrar responsável cidade

Os requisitos suplementares são requisitos que definem atributos de qualidade do sistema, tais como desempenho, usabilidade, funcional, suportabilidade, confiabilidade e restrições. O Quadro 10 mostra alguns requisitos suplementares definidos para o sistema.

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
S1 Tipo de Interface	As Interfaces do sistema devem ser simples e funcionais.	Interface	( )	(X)
S2 Informações das bases	As informações dos cadastros das bases (IP, porta, usuário e senha) devem ser tratadas como informações sigilosas.	Segurança	( )	(X)
S3 Acesso aos equipamentos	O sistema deve esperar no máximo 30 segundos para obter acesso a alguma base, caso contrário deve abortar a execução e informar ao usuário "sem conexão com a base".	Desempenho	( )	(X)
S4 Perfis de usuário	Os perfis de usuário para acesso ao sistema são: 1 – Administrador – pode efetuar todas as operações. 2 – Triagem – pode efetuar as operações básicas do sistema.	Segurança	( )	(X)

Quadro 10 – Requisitos suplementares

O Quadro 11 lista as interfaces definidas para o sistema, com os atores e os casos de uso relacionados, todos com uma descrição complementar.

No.	Nome	Ator	Caso de Uso	Descrição
01	Tela de Login	Usuário Triagem / Administrador	-	Interface para *login* de acesso ao sistema
02	Tela Principal	Usuário Triagem / Administrador	-	Interface para escolha que qual módulo deseja-se utilizar
03	Tela de Cadastro de Usuários	Administrador	Cadastro de Usuário	Interface para inclusão de novos usuários
04	Tela para Alterar Senha	Usuário Triagem / Administrador	Alterar Senha	Interface para alterar senha de Usuário
05	Tela PPPoe Online	Usuário Triagem / Administrador	PPPoe Online	Interface para inclusão, alteração e exclusão de servidores PPPoe. Também lista todos os servidores e seus clientes conectados, faz busca por nome de cliente
06	Tela para Cadastro de Responsável Cidade	Administrador	Cadastro de Responsável Cidade	Interface para inclusão, alteração e exclusão de Responsável Cidade. Também lista os responsáveis cadastrados
07	Tela para Cadastro de Base	Administrador	Cadastro de Base	Interface para inclusão, alteração e exclusão de Base. Também lista as Bases cadastrados
08	Tela para Adicionar Cliente	Usuário Triagem / Administrador	Cadastro de MAC Clientes	Interface para inclusão de clientes na base escolhida
09	Tela para Listar Clientes Base	Usuário Triagem / Administrador	Listar Clientes Base	Interface para listar, editar e excluir clientes cadastrados no *access list* do equipamento acessado.
10	Tela para Listar Clientes Online	Usuário Triagem / Administrador	Listar Clientes Online	Interface para listar os clientes que estão conectados no equipamento acessado

Quadro 11 – Interfaces de usuário

O diagrama de classes representa o sistema de acordo com os conceitos da orientação a objetos, tendo em vista que ele descreve os objetos e os relacionamentos que existem entre eles. O diagrama de classes descreve a estrutura e as relações entre as classes, uma visão estática do sistema em termos de classes e relacionamento entre as classes. A Figura 18 representa o diagrama de classes do sistema com seus respectivos relacionamentos.

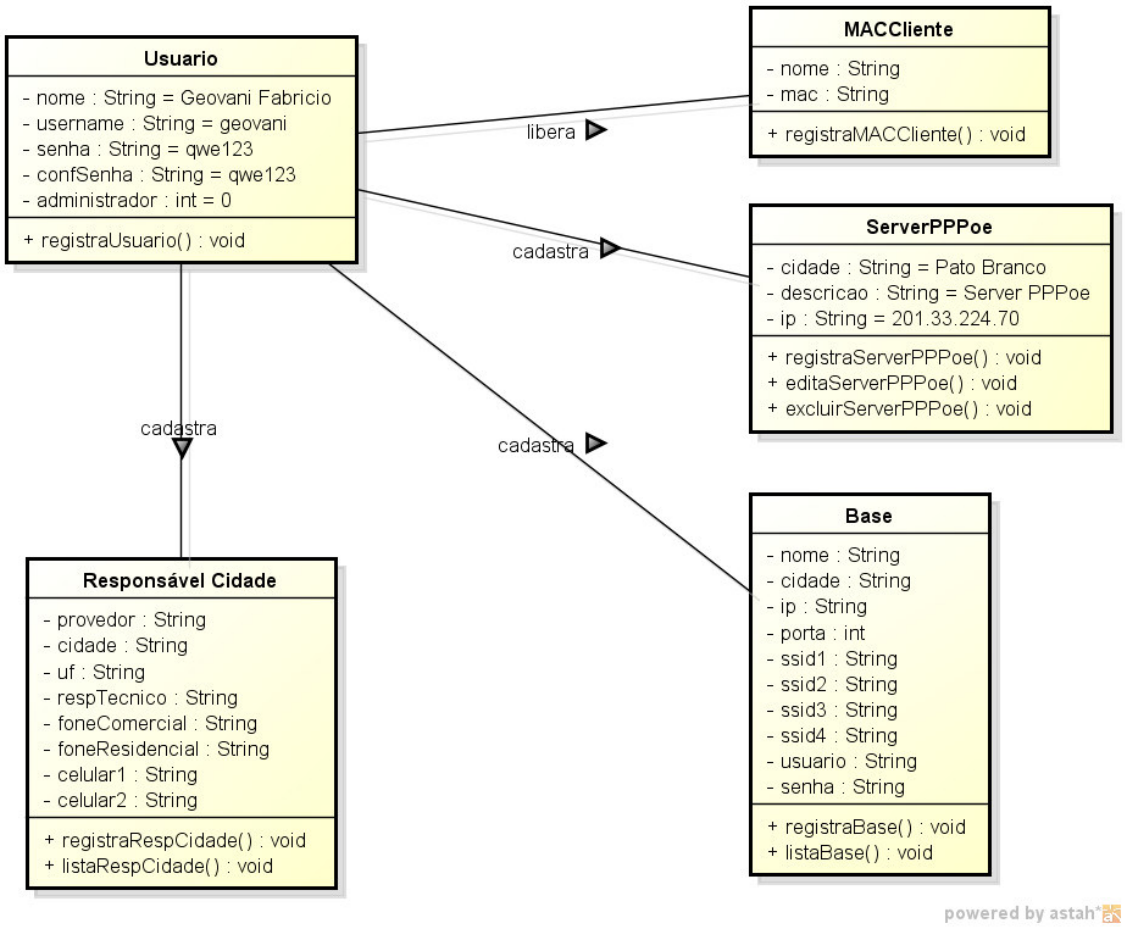

Figura 20 – Diagrama de classes

A Figura 21 apresenta o diagrama de atividades de inclusão de clientes. De acordo com a representação desse diagrama, verifica-se que o usuário do sistema informará os dados nos campos da interface do sistema, o sistema por sua vez, através de um mecanismo de validação, validará a MAC digitada pelo usuário informando válida ou invalida. Sendo a MAC validada, o sistema conecta a base selecionada pelo usuário e libera dentro da lista de acesso do equipamento o cliente informado. O sistema por sua vez pode conseguir a conexão ou não ao equipamento, esse deve fornecer um *feedback* do sucesso ou falha na conexão.

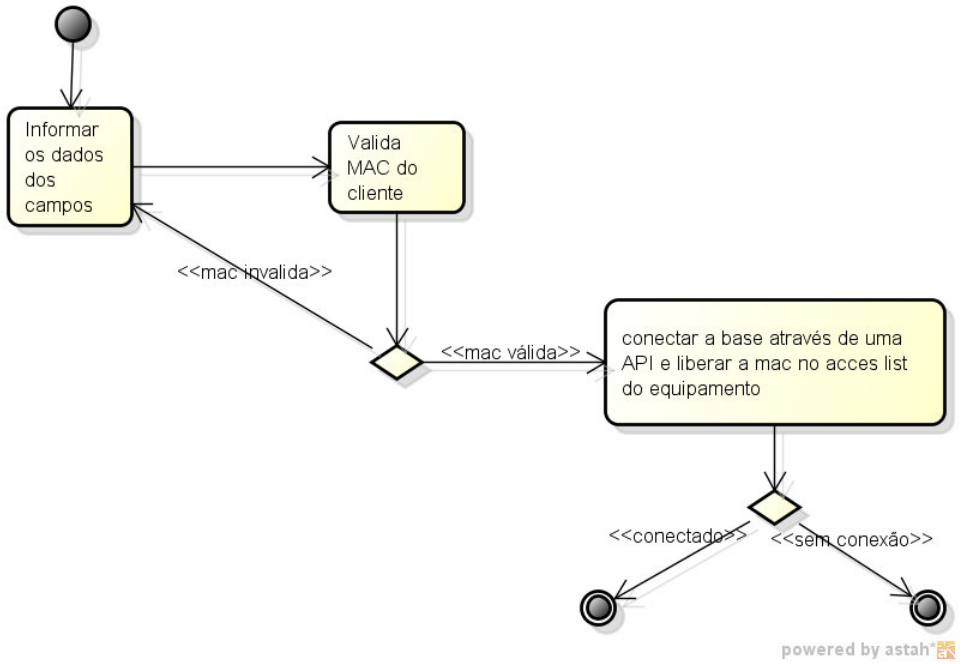

Figura 21 – Diagrama de atividade inclusão de cliente no equipamento

A Figura 22 representa a atividade de listar os clientes *online* no equipamento. O usuário seleciona a base a ser acessada. O sistema, por sua vez, conecta na base e lista na tela os clientes que estão *online* no momento.

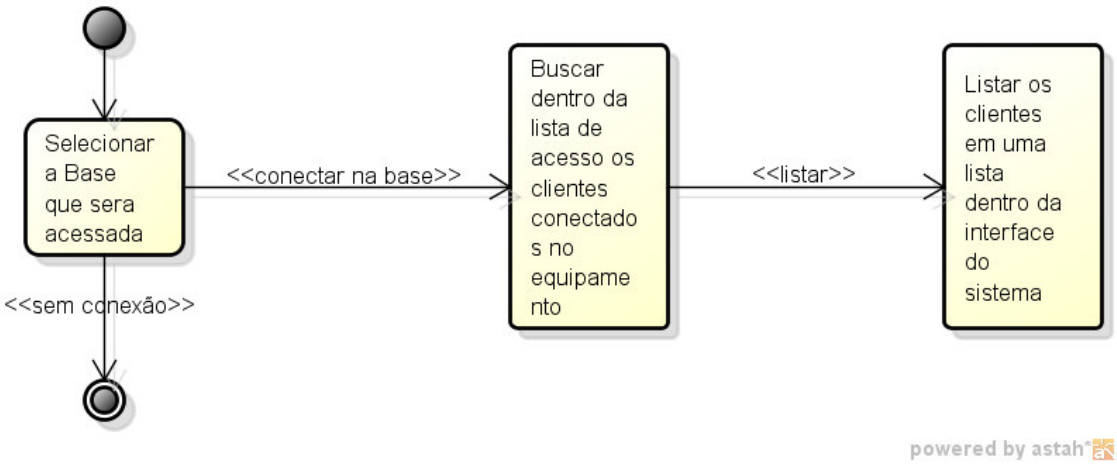

Figura 22 – Diagrama de atividade listar clientes online

A Figura 23 mostra a sequência das mensagens para adicionar um cliente na base escolhida pelo usuário na etapa onde o mesmo preencheu os campos.

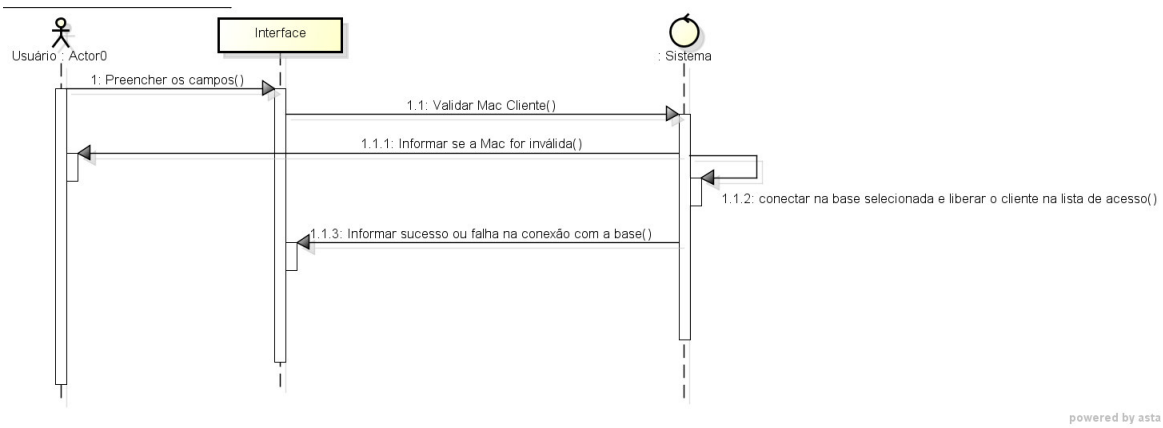

Figura 23 – Diagrama de sequência adicionar cliente

A Figura 24 representa a sequência para listar na tela os clientes que estão conectados em uma base escolhida pelo usuário.

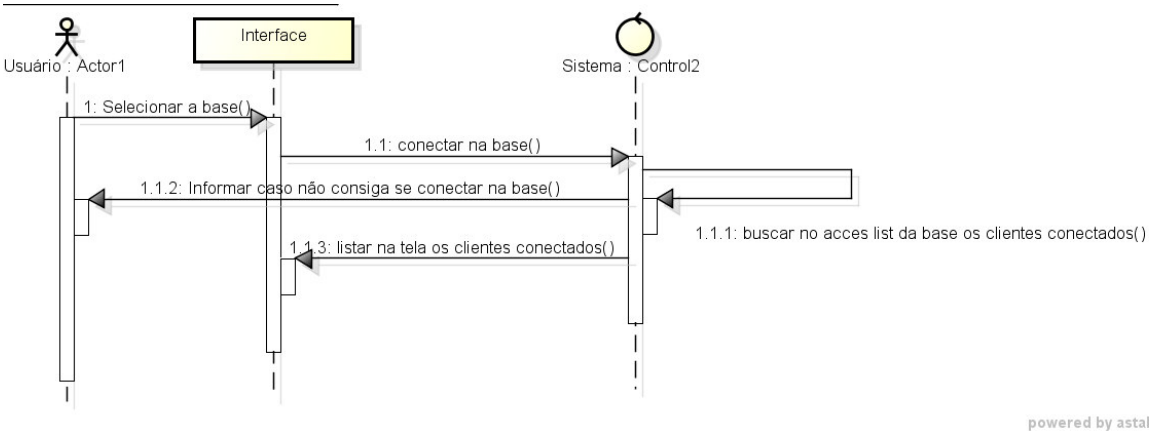

Figura 24 – Diagrama de sequência listar clientes online

4.3 DESCRIÇÃO DO SISTEMA

O acesso ao sistema ocorre por meio de validação de *login,* apenas o usuário administrador terá permissão dentro do sistema para cadastrar novos usuários. Na Figura 25 está a tela de *login* ao sistema. Se o usuário não informar os dados corretos de *login* (usuário e senha) uma mensagem será informada contendo o texto: “Usuário e senha inválidos, digite novamente”.

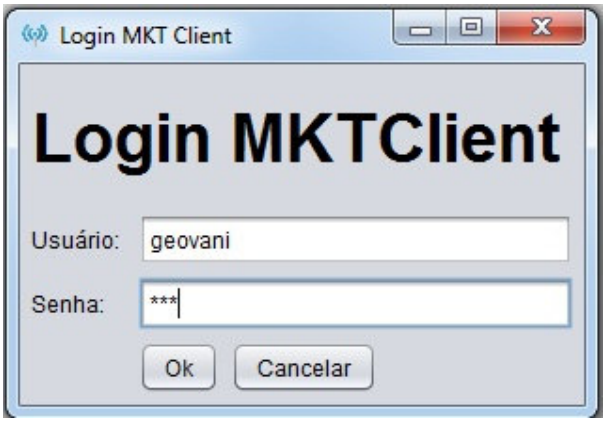

Figura 25 – Tela de login

Assim que é validado o usuário e a senha junto ao banco de dados o sistema salva em uma classe o usuário que fez o *login*. Essa informação é útil para alterar a senha do *login* conectado e verificar o tipo do usuário e consequentemente saber quais funções o usuário terá acesso.

Realizada a validação de acesso, a tela principal do sistema é apresentada. A Figura 26 representa a tela principal do sistema.

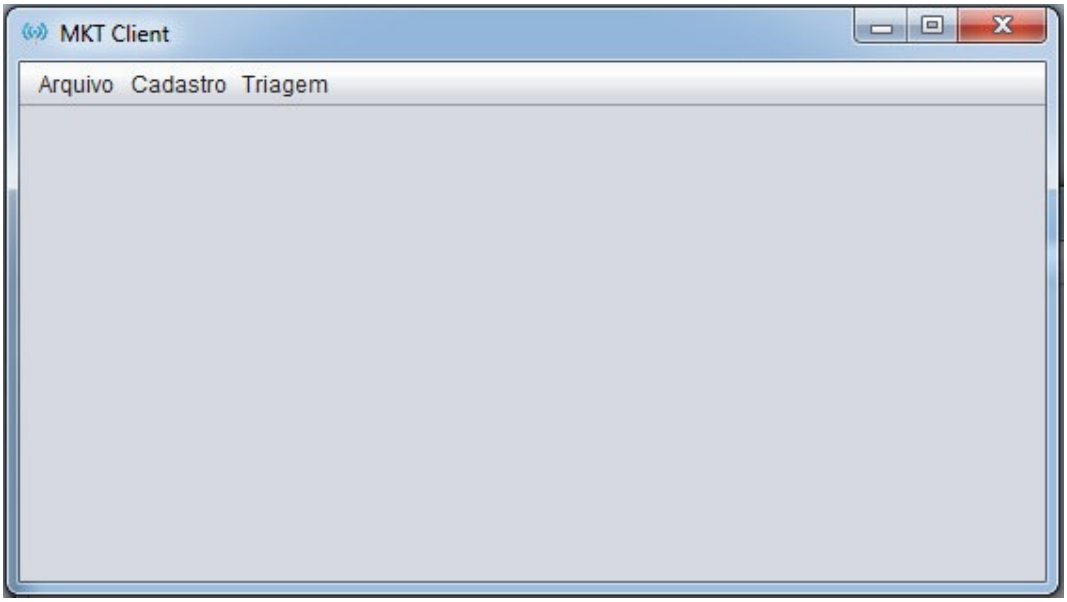

Figura 26 – Tela inicial do sistema

No topo da tela estão os menus de acesso aos cadastros e as funções do sistema. O primeiro menu “Arquivo” tem as funções de Cadastrar “Novo Usuário” e “Alterar Senha”. Apenas o usuário administrador terá a opção de cadastrar novo usuário e todos os usuários terão permissão para alterar senha. A Figura 27 apresenta a tela de “Novo Usuário” aberta.

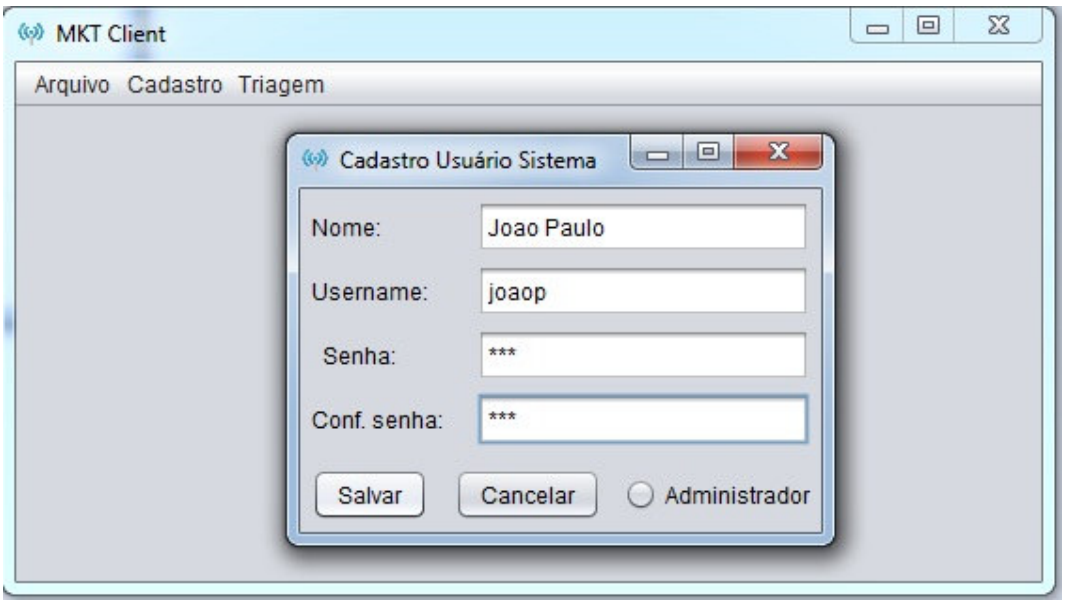

Figura 27 – Cadastro de novo usuário

Utilizando da Classe "SessaoUsuLogado" é recuperado o usuário logado, o que possibilita alterar a senha do funcionário que está utilizando o sistema. A Figura 28 representa a tela de "Alterar Senha".

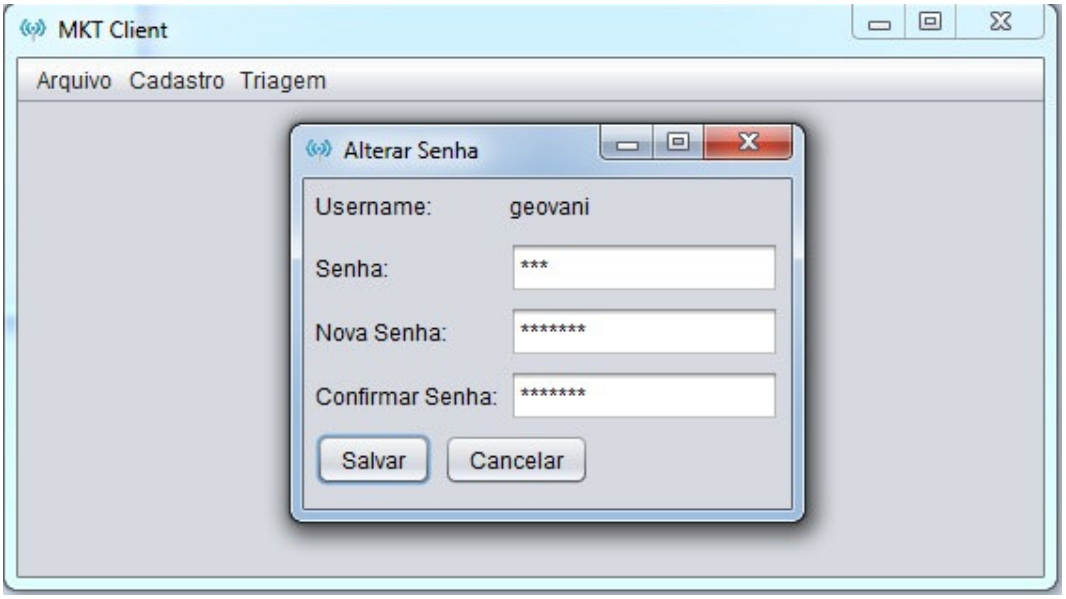

Figura 28 – Alterar senha de usuário logado

O segundo menu, "Cadastro", possui as funções de Cadastrar Responsável Cidade e Cadastrar Base.

A opção "Responsável Cidade" (Figura 29) abre a tela para cadastrar um novo

Responsável.

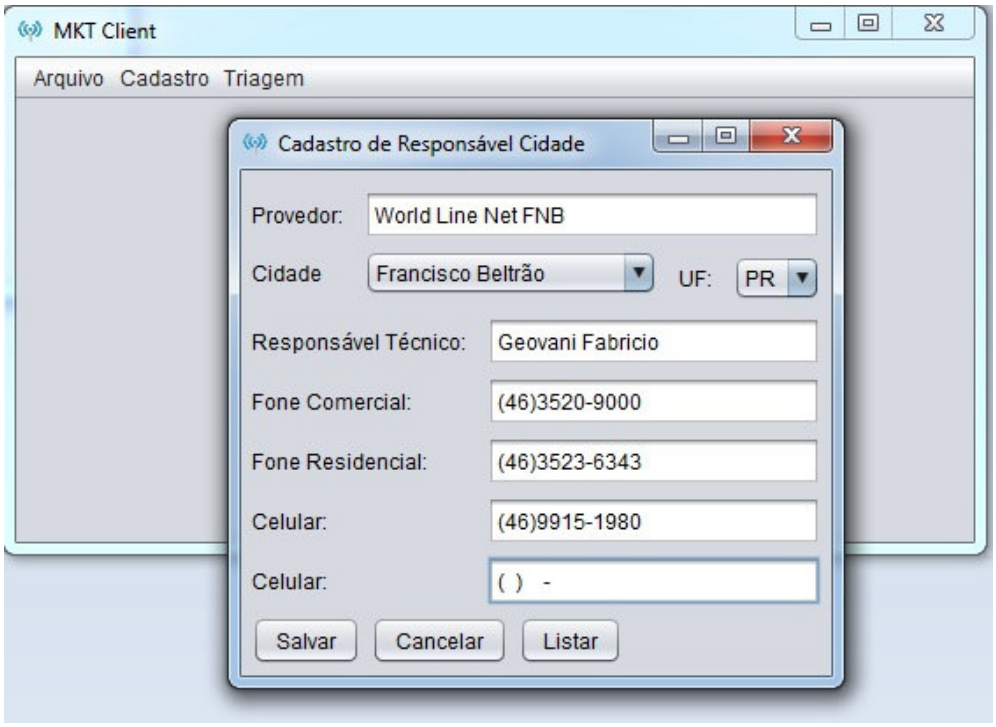

Figura 29 – Responsável cidade

Se selecionada a opção “Cancelar” a tela será finalizada. Caso a escolha seja “Salvar” o cadastro do responsável será salvo no banco de dados e uma mensagem na tela será apresentada “Inclusão efetuada com Sucesso” ou “Erro na Inclusão”. O botão “Listar” abre uma nova tela e lista os responsáveis que estão cadastrados conforme a apresenta a Figura 30.

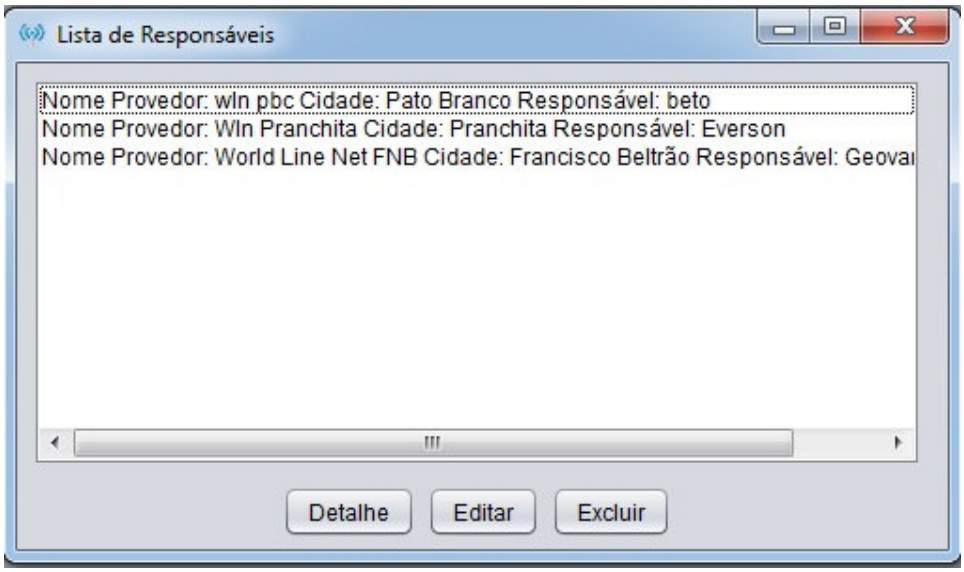

Figura 30 – Lista responsável cidade

Caso o usuário clique no botão “Detalhe”, todos os dados referentes ao responsável serão apresentados conforme apresenta a Figura 31.

Figura 31 – Detalhe responsável cidade

A Figura 32 mostra a tela que será aberta quando o usuário clicar em “Editar”. O usuário pode confirmar as modificações ou cancelá-las.

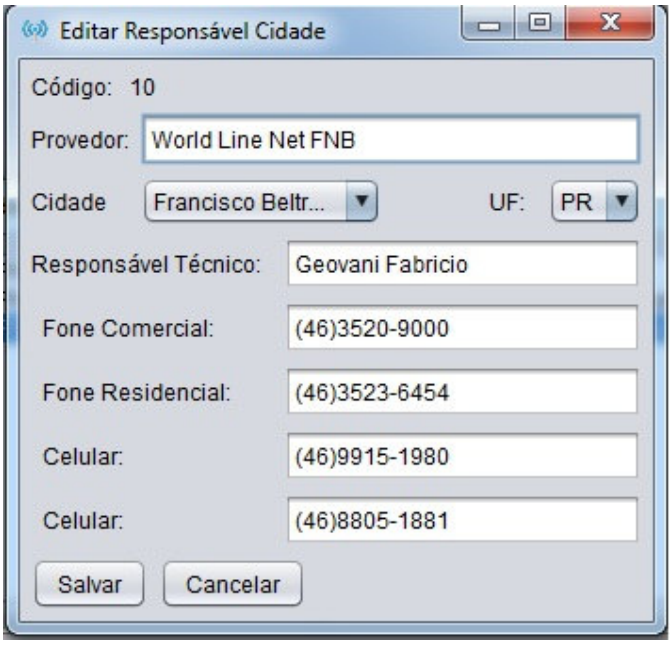

Figura 32 – Editar responsável cidade

Quando a opção for “Excluir” (Figura 33) uma confirmação de exclusão é solicitada ao usuário podendo assim o mesmo confirmar ou cancelar a operação, essa operação excluirá

do banco todas as informações do responsável selecionado na lista.

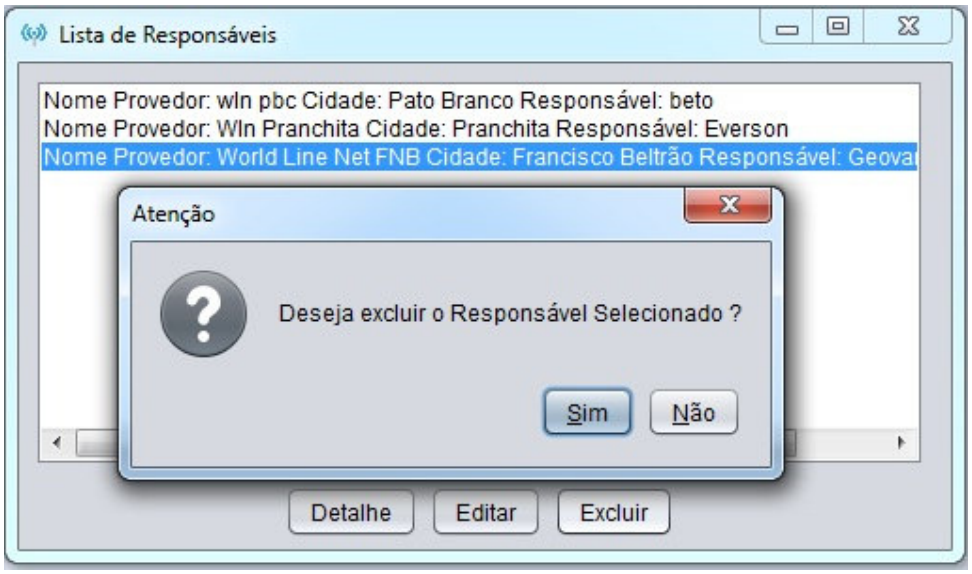

Figura 33 – Excluir responsável cidade

No mesmo menu “Cadastro” tem a função de “Base” (Figura 34) que permite o cadastro da base. A “Base” cadastrada no sistema nada mais é que um equipamento Routerboard executando um sistema Operacional RouterOS que está instalado em uma torre ou prédio e possui a função de gerar ondas de rádio (sinal wireless) para os clientes. Esse equipamento por sua vez possui uma lista dos clientes liberados (access-list) e uma lista dos clientes que estão conectados no momento.

O botão “Salvar” confirma o cadastro e apresenta uma mensagem de sucesso ou falha na operação. O botão “Cancelar” cancela a operação e fecha a tela. O botão “Listar” apresenta exatamente as mesmas funções do Listar Responsável Cidade.

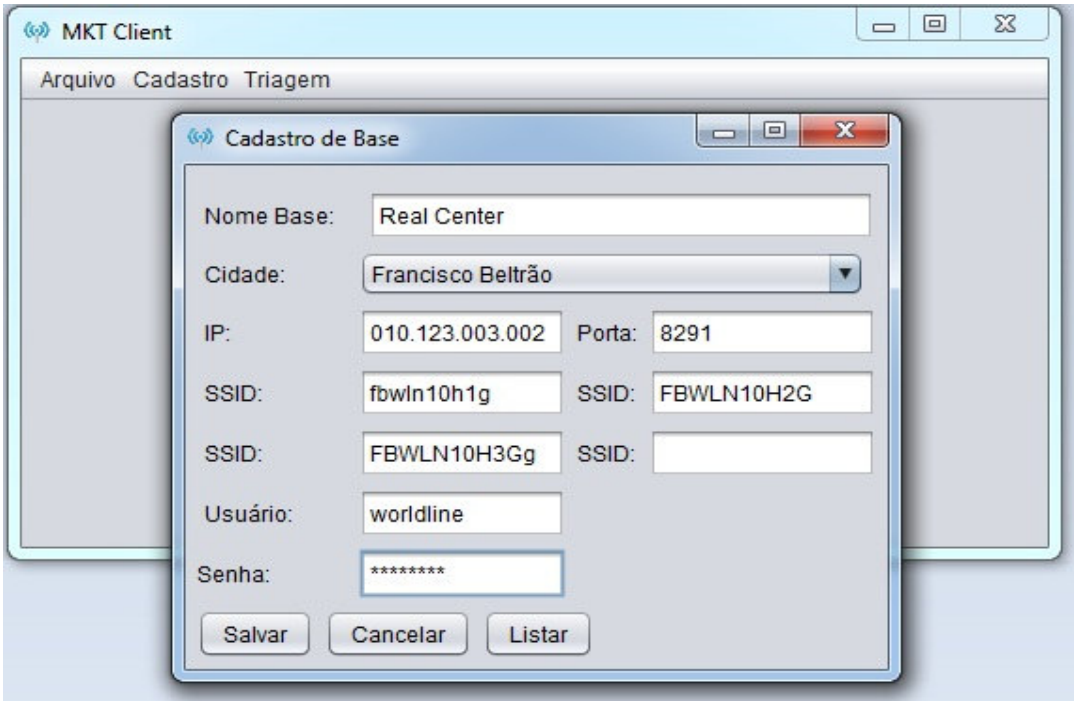

Figura 34 – Cadastro base

O botão "Listar" apresenta uma lista com as bases cadastradas. Sendo possível obter detalhes, editar e excluir bases. Conforme mostra a Figura 35.

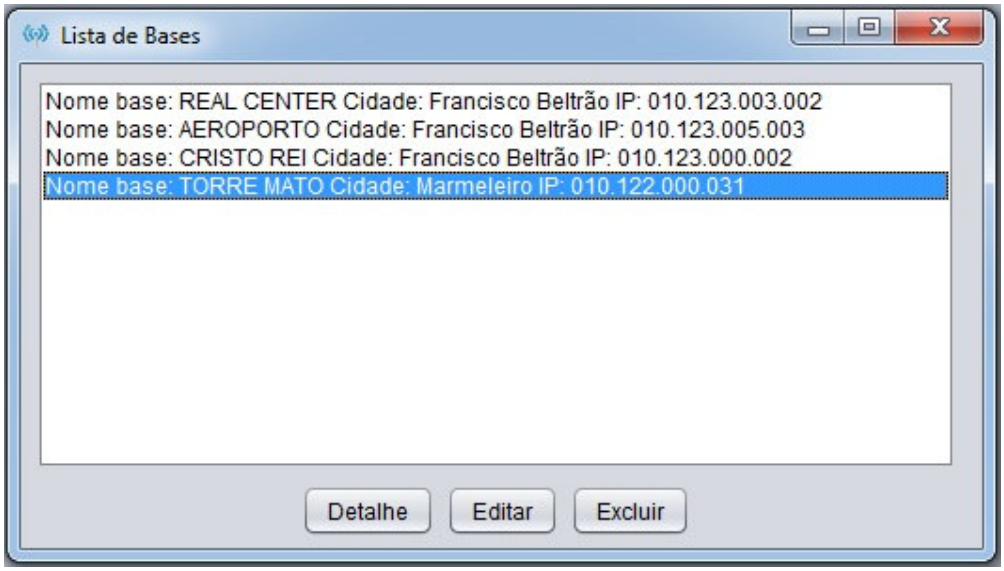

Figura 35 – Lista de bases cadastradas

A Figura 36 apresenta o botão "Detalhe" da lista de base, selecionando uma base na lista são disponibilizadas mais informações da base selecionada.

Figura 36 – Detalhes da base

O botão "Editar" da Lista de Base é representado pela Figura 37. É possível fazer alterações no cadastro da base utilizando esse recurso.

Figura 37 – Editar base

Também é possível excluir uma base desde que esta seja seleciona na lista e utilizando do botão "Excluir" representado na Figura 38.

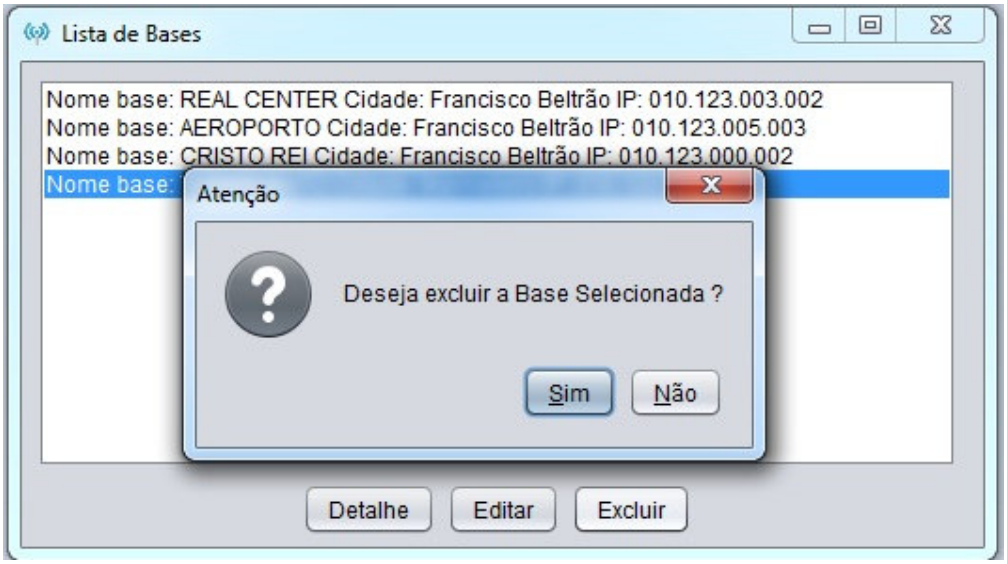

Figura 38 – Excluir base

O terceiro Menu da Tela Principal cujo nome é "Triagem" é destinada para as operações que serão realizadas dentro dos equipamentos "Base" que estão cadastrados no sistema.

As operações realizadas diretamente no equipamento são:

a) Adicionar Cliente

Essa função (Figura 39) permite cadastrar um cliente no *access-list* da base selecionada. Informações padrão são solicitadas como "nome do cliente", "Mac Address" esse campo está formatado como Hexadecimal o que inibe um possível endereço físico inválido. Ao selecionar o campo "Cidade" é selecionado, o sistema busca em seu banco de dados quais são as bases dessa cidade e lista no campo "Base". Logo lista quais são ou *ssid´s*, ou melhor, as interfaces *wireless* disponíveis na base selecionada. O campo "Signal Strength Range" permite informar qual será o mínimo de sinal que o cliente vai precisar para se conectar na base. O campo "Authentication" libera a autenticação do cliente ou não e por fim o campo "Forwarding" que serve para não deixar um cliente visualizar outro.

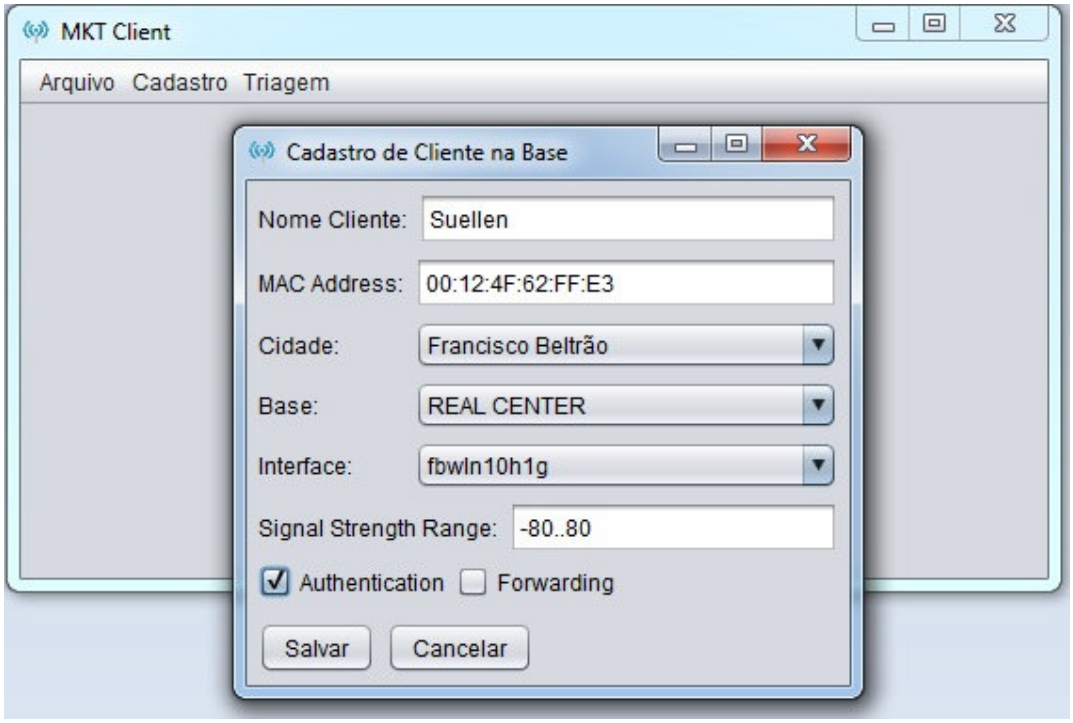

Figura 39 – Adicionar cliente base

b) Clientes Liberados

A tela "Clientes Liberados" representada pela Figura 40 permite verificar quais são os clientes que estão liberados dentro do *access-list* do equipamento. Assim que escolher a cidade, o sistema busca quais são as bases dessa cidade e as carrega no combo "Base". O botão "Listar" acessa o equipamento e lista quais são os clientes que estão liberados nessa Base. O botão "Disable" e "Enable" desativa e ativa, respectivamente, o cliente dentro do equipamento.

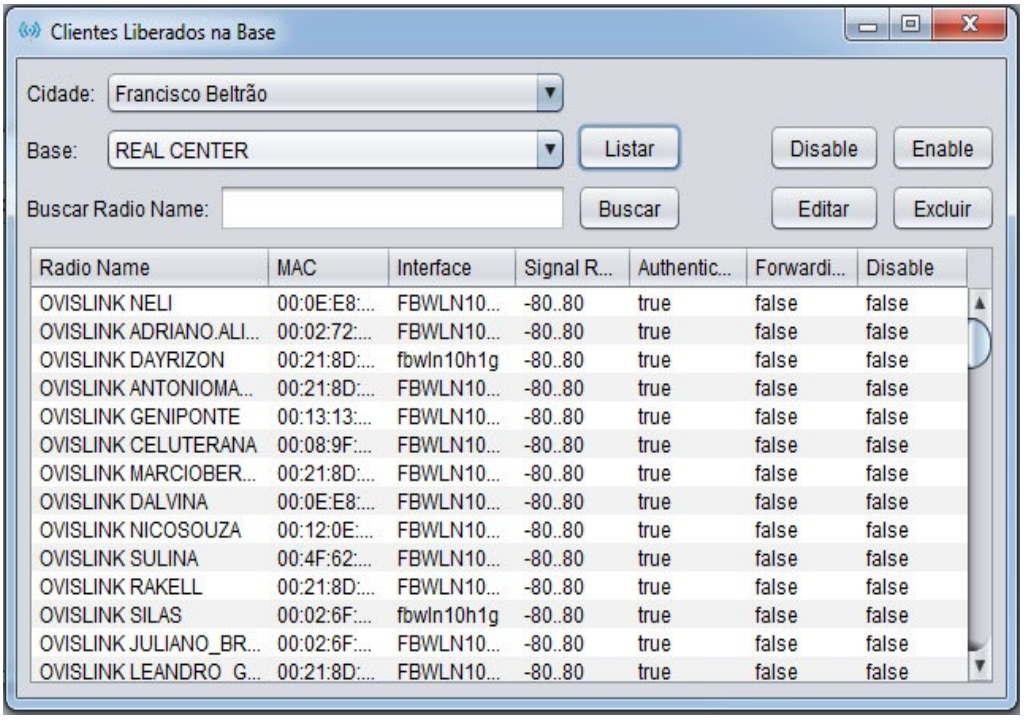

Figura 40 – Clientes liberados

O botão “Buscar” (Figura 41) nada mais é que um filtro na coluna Radio Name.

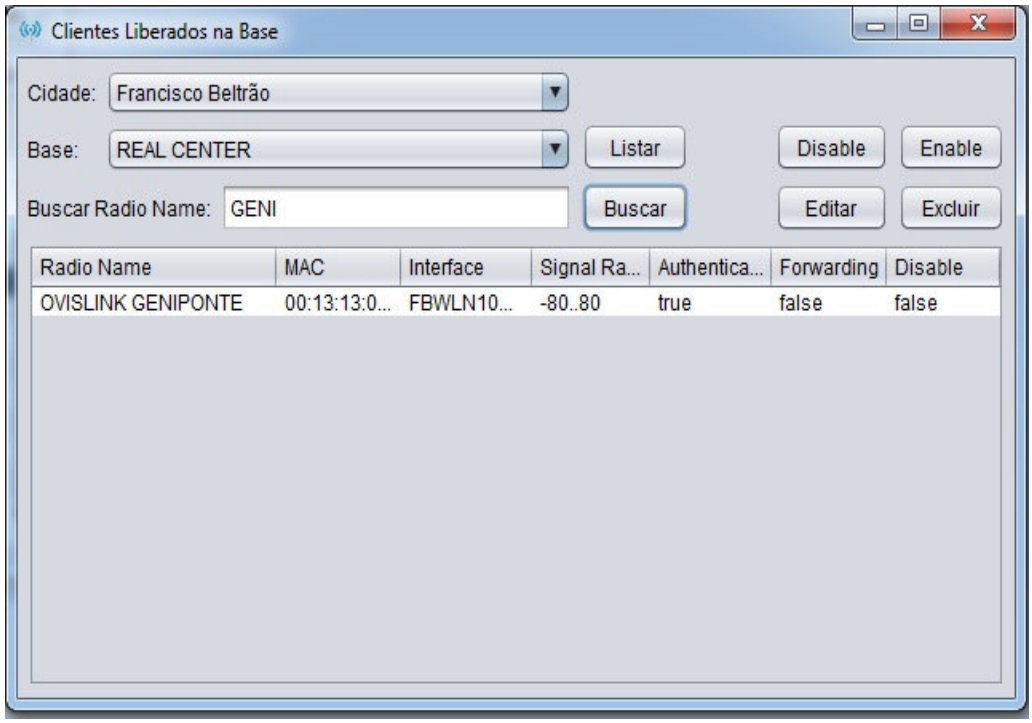

Figura 41 – Filtro buscar cliente

O botão “Excluir” (Figura 42) exclui o cliente selecionado do *access-list* da base, para isso solicita uma confirmação de exclusão.

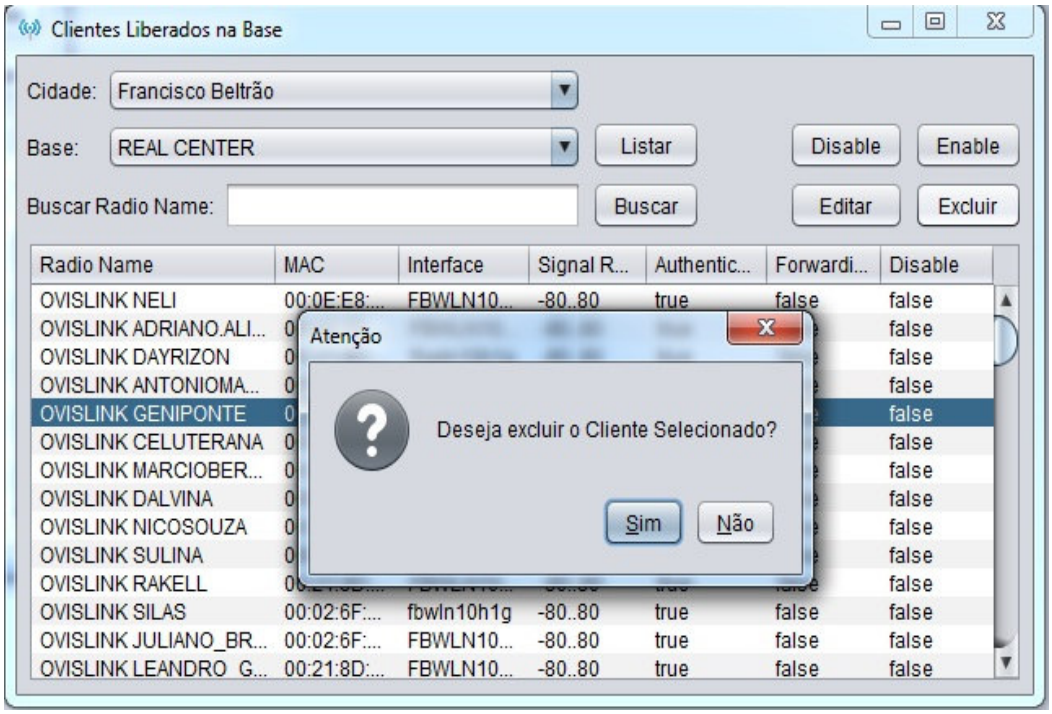

Figura 42 – Excluir cliente do access-list

O botão “Editar” abre uma nova Janela onde o usuário pode editar informações do cadastro do cliente. Essa janela é representada pela Figura 43. O botão “Salvar” confirma as alterações do cliente dentro do *access-list* da Base e o botão “Cancelar ” cancela a operação.

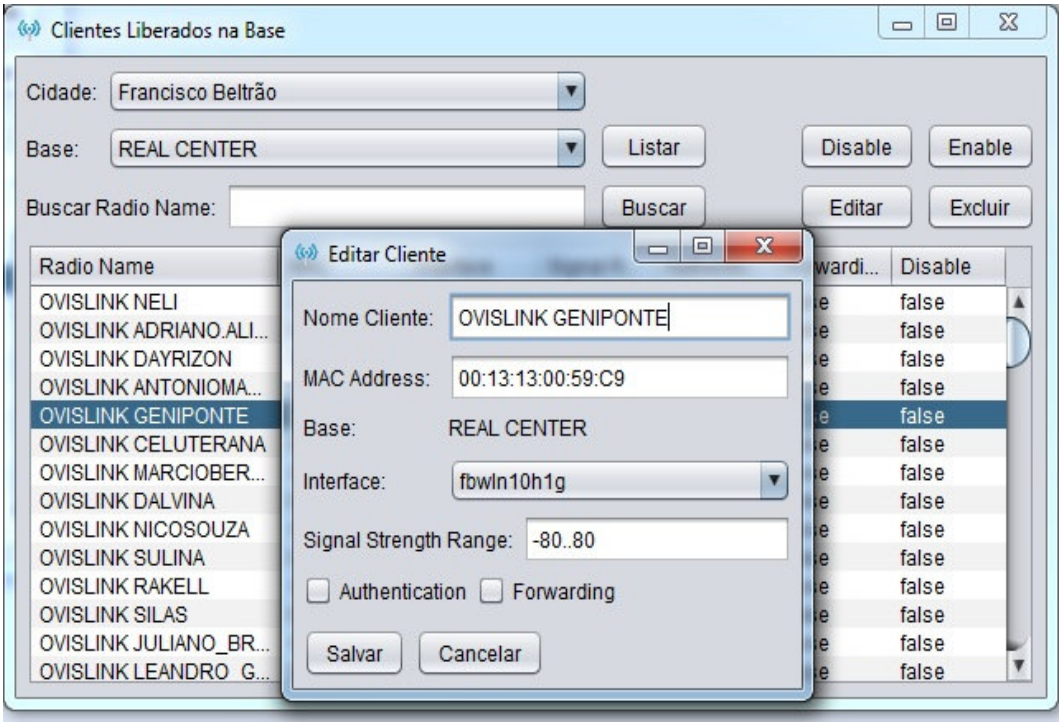

Figura 43 – Editar cliente no access-list

b) Clientes Online

Nessa tela (Figura 44) o usuário consegue listar os clientes que estão conectados na Base selecionada pelo usuário. O mecanismo de busca é um filtro na coluna "Radio Name" e o botão "Derrubar Cliente" finaliza a conexão do cliente selecionado na lista por alguns segundos. Várias informações de valor relevante para o Setor de Triagem são mostrados nesse *list*, como o sinal do cliente, há quanto tempo está conectado, em qual interface está conectado e o endereço físico do cliente.

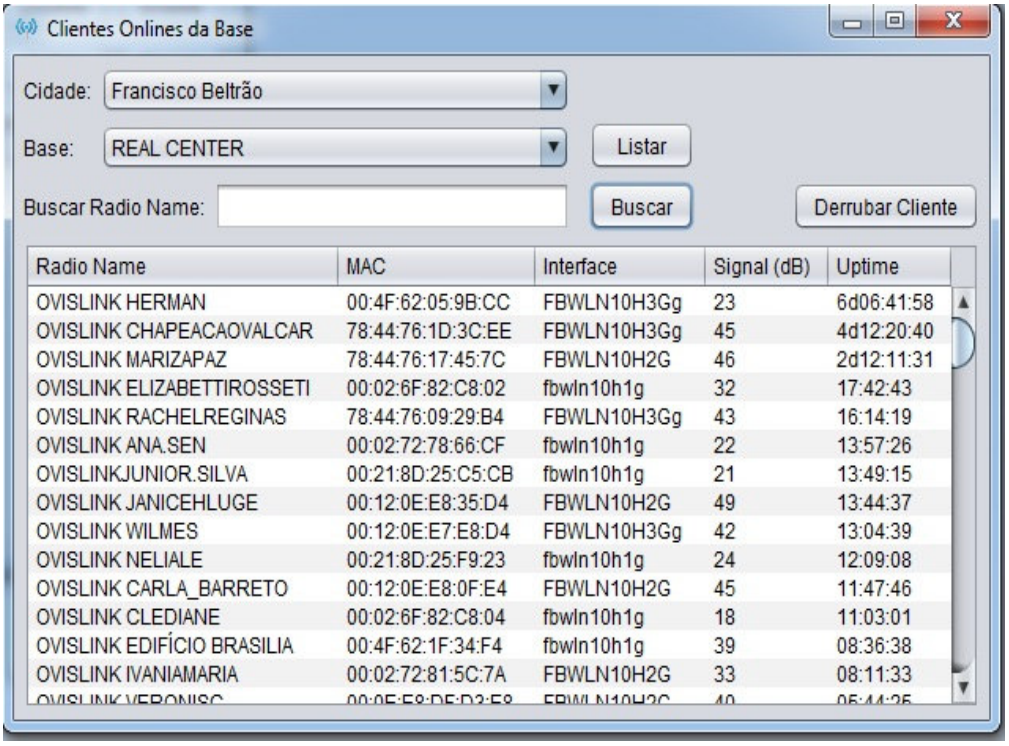

Figura 44 – Clientes online na base

d) PPPoe Online

Para o cliente conseguir o acesso à Internet precisa estar conectado em uma base. Após isso estabelecer uma conexão com um equipamento RouterBoard (Servidor PPPoe) que também possui um sistema Operacional RouterOS. Essa nova conexão se faz necessária para trocar as chaves de acesso, usuário e senha do cliente obtendo assim a conexão ou não.

A Figura 45 representa a tela na qual é possível cadastrar e excluir esses Servidores de PPPoe, também é possível verificar os clientes que estão conectados nesse servidor . O botão “Buscar” é um filtro na coluna “Username” da tabela em que estão listados os clientes conectados no Servidor.

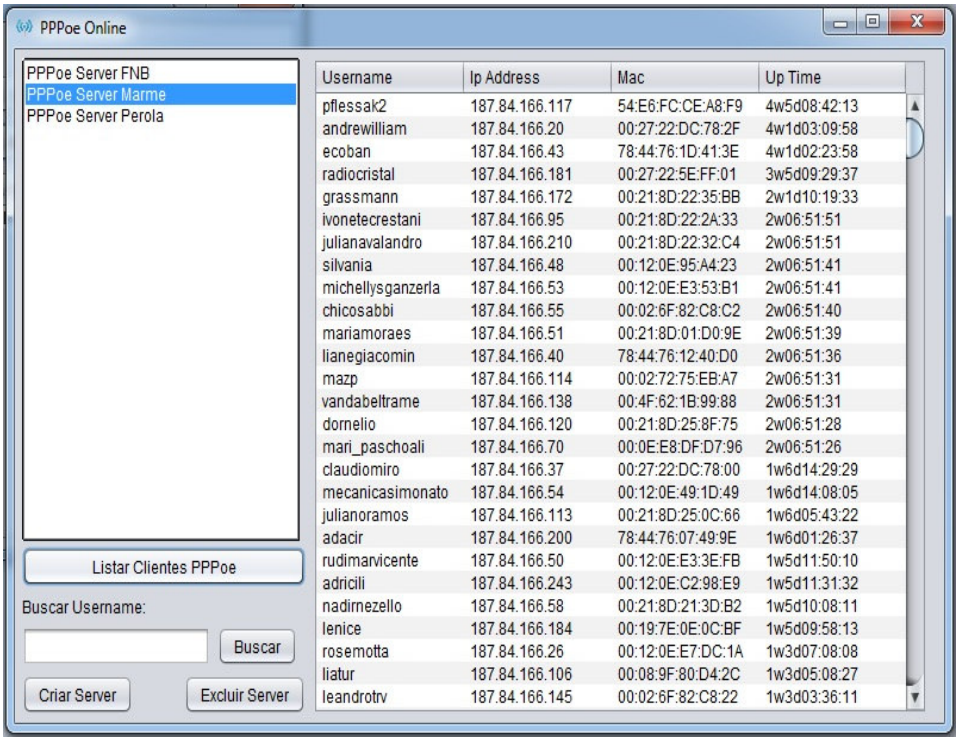

Figura 45 – PPPoe online

O botão “Criar Server” abre a tela que esta representada na Figura 46 na qual é feito o cadastro do Servidor de PPPoe. O campo “IP” possui uma função que verifica se o IP digitado é válido ou não. O botão “Salvar” confirma o cadastro e o “Cancelar” cancela a operação.

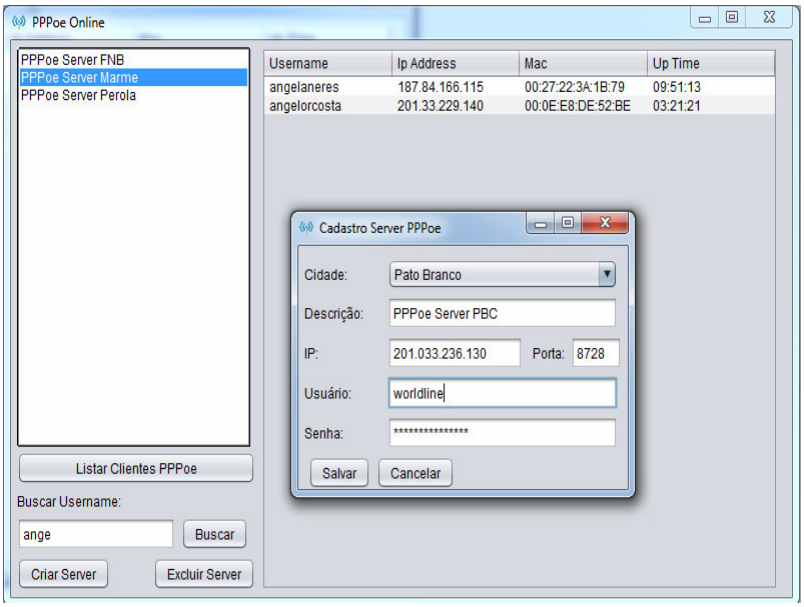

Figura 46 – Cadastro servidor PPPoe

A Figura 47 representa o filtro de busca por “username” na tabela.

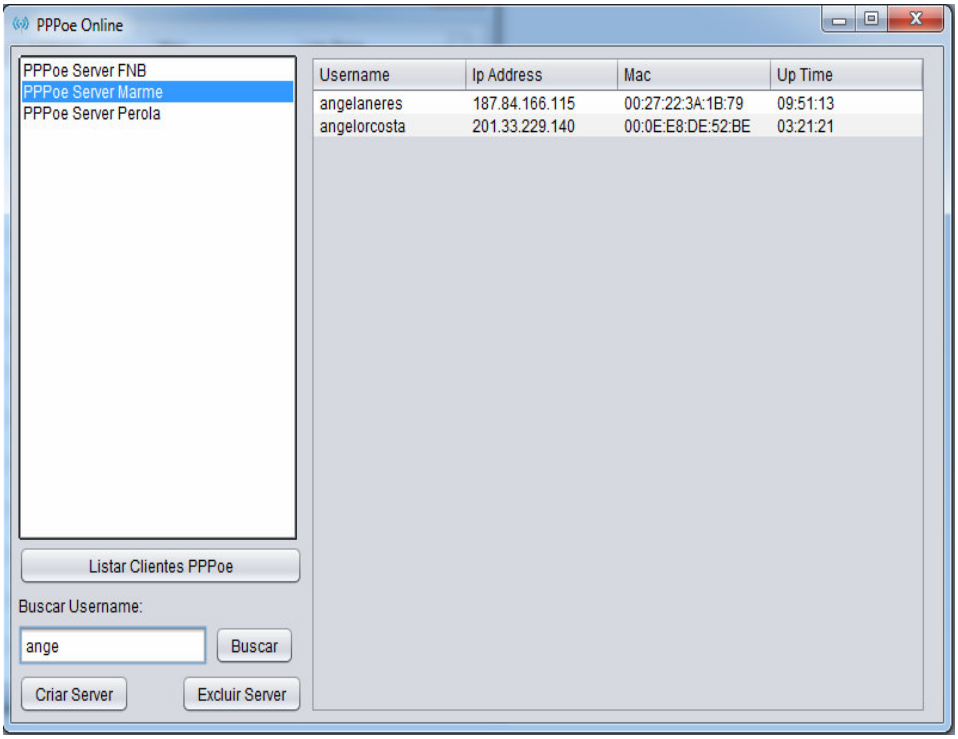

Figura 47 – Filtro de busca cliente PPPoe

4.4 IMPLEMENTAÇÃO DO SISTEMA

Pacotes podem facilmente definir as classes com funções parecidas que ficarão dentro do projeto, facilitam a localização das classes e criam uma estrutura hierárquica de diretórios. Empacotar classes significa criar uma estrutura que ajuda a manutenção e fomenta os princípios de encapsulamento e modularidade da orientação a objetos.

O sistema desenvolvido como resultado deste trabalho foi dividido em pacotes permitindo organizar as classes com finalidades comuns. A Figura 48 representa o projeto e seus pacotes.

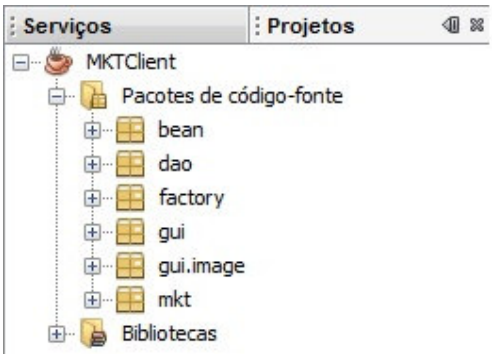

Figura 48 – Pacotes do projeto

O projeto está dividido nos pacotes:

a) “bean” – contendo as classes dos objetos do sistema;

b) “dao” – com as operações no banco, trata a parte de dados do sistema;

c) “factory” – contém a classe responsável por fazer a conexão com o banco de dados;

d) “gui” - nesse pacote estão as classes que farão a interface com o usuário;

e) “gui.image” – com as imagens e ícones necessários no sistema;

f) “mkt” – armazena as duas classes responsáveis pela conexão e troca de comando através de uma API com o equipamento *routerboard*;

Assim que o usuário tenta se logar no sistema através da interface de login o método “validaLogin()” (Listagem 1) é chamado. Os objetos das classes “Usuario” e “OperacoesDao” respectivamente “usu” e “dao” estão declarados e instanciados dentro da classe Principal. Esse método consiste em recuperar em duas *Strings* o usuário e senha digitados. O objeto “dao” irá fazer uma pesquisa no banco procurando pelo usuário digitado, se encontrar retorna um objeto do tipo “Usuario”. Em seguida faz a verificação do retorno com o usuário e senha digitados. Por fim salva na classe “SessaoUsuLogado” o usuário que está conectando, fecha a tela de *login* e abre a tela principal do sistema.

```
184     private void validaLogin() {
185
186         String senhaTela = String.valueOf(tfSenha.getPassword());
187         String usuTela = tfUsuario.getText();
188
189         try {
190             usu = dao.pesquisarUsu(tfUsuario.getText());
191         } catch (Exception e) {
192             JOptionPane.showMessageDialog(this, "Erro: " + e.getMessage());
193         }
194
195         if (usuTela.equals(usu.getUsuario()) && senhaTela.equals(usu.getSenha())) {
196             SessaoUsuLogado usuLogado = SessaoUsuLogado.getInstance();
197             usuLogado.setUsuario(usu); // salva na sessão o usuário logado
198
199             TelaPrincipal telaP = new TelaPrincipal();
200             telaP.setVisible(true);
201             this.dispose();
202
203         } else {
204             JOptionPane.showMessageDialog(this, "Usuário e senha inválidos, digite novamente.");
205             tfUsuario.setText("");
206             tfSenha.setText("");
207         }
208     }
209 }
```

Listagem 1 – ValidaLogin()

A interface “OperacoesBancoImpl” (Listagem 2) define que comportamentos uma classe que implementa essa interface deve ter. Essa interface mostra com clareza e facilidade quais são os métodos que irão ser implementados na classe “OperacoesDao”.

```
public interface OperacoesBancoImpl {

    public boolean incluirUsuario( Usuario usu ) throws Exception;
    public boolean editarUsuario( Usuario usu ) throws Exception;
    public Usuario pesquisarUsu( String usuario ) throws Exception;

    public boolean incluirServerPPPoe( ServerPPPoe pppoe) throws Exception;
    public boolean editarServerPPPoe( ServerPPPoe pppoe) throws Exception;
    public boolean excluirServerPPPoe( String descricao) throws Exception;
    public ArrayList listarServerPPPoe() throws Exception;
    public ServerPPPoe pesquisarServerPPPoePeloNome(String descricao) throws Exception;

    public boolean incluirRespCidade( RespCidade respcidade) throws Exception;
    public boolean editarRespCidade( RespCidade respcidade) throws Exception;
    public boolean excluirRespCidade( int cod) throws Exception;
    public ArrayList listarRespCidade() throws Exception;

    public boolean incluirBase( Base base) throws Exception;
    public boolean editarBase( Base base) throws Exception;
    public boolean excluirBase( int cod) throws Exception;
    public ArrayList listarBase() throws Exception;
    public ArrayList pesquisarBasePelaCidade(String cidade) throws Exception;
    public Base pesquisarBasePeloNome(String nome) throws Exception;
}
```

Listagem 2 – Interface OperacoesBancoImpl

Como o sistema terá que acessar o banco e salvar algumas informações foi necessária a instalação de um banco de dados, o escolhido foi o MySQL e por meio da ferramenta MySQL-Front foi criado o banco e as tabelas. Em seguida foi adicionado o *driver* JDBC, “mysql-connector-java-5.1.13”, que encapsula uma interface com os comandos padrões do banco, os métodos de inserção, pesquisa, *update* e *delete* são os mesmos para os bancos que utilizam dessa JDBC. O “Class.forName” carrega o *driver* JDBC na memória passando o caminho do arquivo. A classe “Connection” é responsável por armazenar a conexão com o banco. A classe “DriverManager” estabelece a conexão através do método “getConnection” passando por parâmetro o caminho do banco, usuário e senha para conexão.

A classe “BdConnection” que esta dentro do pacote “factory” é responsável por fazer a conexão com o banco de dados. A Listagem 3 apresenta essa classe.

```java
package factory;

import java.sql.Connection;
import java.sql.DriverManager;

public class BdConnection {

    public static Connection con;

    public static Connection getConexao() throws Exception {

        if ( con == null ) {
            //Carregar o Driver JDBC em memória
            Class.forName("com.mysql.jdbc.Driver");

            //Conectar no Banco (3 informações: url, user, pass
            String url = "jdbc:mysql://127.0.0.1:3306/mktclient";
            String user = "root";
            String pass = "admin";

            con = DriverManager.getConnection( url, user, pass );

        }

        return con;
    }
}
```

Listagem 3 – Classe BdConnection

O sistema vai permitir incluir, editar e excluir bases, responsáveis pelas cidades, servidores PPPoe e usuários do sistema. Em seguida será apresentado o incluir, o editar e o excluir bases. A lógica para incluir, editar e excluir responsáveis, servidores e usuários são praticamente a mesma por esse motivo não será representada nesse relatório.

Na interface do cadastro da base o botão “Salvar” chama o método inserirBase(). Ele é responsável por validar as informações digitadas e em seguida chamar o método “incluirBase(Base base)” da Classe “OperacoesDao” que é responsável por incluir a base no banco. A Listagem 4 contém o código dessa função.

```
319     private void inserirBase() {
320
321         boolean validaIP = IPehValido(tfIP.getText());
322
323         while (validaIP == false) {
324             String texto = JOptionPane.showInputDialog(this, "Ip inválido. Digite o novo endereço (***.***.***.***): ");
325             tfIP.setText(texto);
326             validaIP = IPehValido(texto);
327         }
328         Base base = new Base();
329         base.setNome(tfNomeBase.getText());
330         base.setCidade(tfcidade.getSelectedItem().toString());
331         base.setIp(tfIP.getText());
332         base.setPorta(Integer.parseInt(tfPorta.getText()));
333         base.setSsid1(tfSSID1.getText());
334         base.setSsid2(tfSSID2.getText());
335         base.setSsid3(tfSSID3.getText());
336         base.setSsid4(tfSSID4.getText());
337         base.setUsuario(tfUsuario.getText());
338         base.setSenha(tfSenha.getText());
339
340         try {
341             boolean ret = dao.incluirBase(base);
342
343             if (ret == true) {
344                 JOptionPane.showMessageDialog(this, "Inclusão Efetuada com Sucesso");
345             } else {
346                 JOptionPane.showMessageDialog(this, "Erro na Inclusão");
347             }
348         } catch (Exception e) {
349             JOptionPane.showMessageDialog(this, "Erro:"+ e.getMessage());
350         }
```

Listagem 4 – inserirBase()

A função da linha 321 na Listagem 5 é responsável por verificar se o IP digitado na tela é válido. Caso o IP seja inválido entra em um *loop*, saindo somente quando o IP for digitado corretamente. Na Figura 51 está a função “IPehValido(String ip)”. Essa função verifica se o tamanho da *string* é menor que sete e maior que 15, após quebrar a *string* verifica se esta foi quebrada em 4 pedaços e por último verifica se cada bloco esta entre 1 e 255 retornando verdadeiro ou falso.

```
354     public static boolean IPehValido(String ip) {
355         //Checa se o comprimento da string é menor que 7 e maior que 15
356         //Mínimo 1.1.1.1 Máximo 255.255.255.255
357         if (ip.length() < 7 || ip.length() > 15) {
358             return false;
359         }
360 
361         //Quebrando a string em array pelo símbolo . deve ser gerado um array com 4 itens
362         String bloco[] = ip.split("\\.");
363 
364         if (bloco.length != 4) {
365             return false;
366         }
367 
368         boolean retorno = true;
369         try {
370             for (String item : bloco) {
371                 if (Integer.parseInt(item) < 1 || Integer.parseInt(item) > 255) {
372                     retorno = false;
373                 }
374             }
375         } catch (NumberFormatException ex) {
376             retorno = false;
377         }
378 
379         return retorno;
380 
381     }
```

Listagem 5 – IPehValido(String ip)

Assim que o IP é validado, é criado e instanciado um objeto do tipo “Base” e são atribuídos valores a seus atributos através dos métodos “set”. A linha “boolean ret = dao.incluirBase(base)” executa o método da classe “OperacoesDao” responsável por incluir a base passada por parâmetro ao banco. O retorno desse método corresponde ao sucesso ou fracasso na operação. Primeiramente é estabelecido a conexão com o banco através do método “getConexao()” na classe “BdConnection”. Em seguida o “preparedStatement” prepara o SQL para fazer o *insert* no banco. O método “incluirBase(Base base)” (Listagem 6) é responsável por atribuir os valores ao “preparedStatement” que ao final é executado.

```
41          Connection con = BdConnection.getConexao();
75          pstmtInserirBase = con.prepareStatement(
76                  "INSERT INTO mktclient.base VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
            @Override
205         public boolean incluirBase(Base base) throws Exception {
206
207             pstmtInserirBase.setString(1, base.getNome());
208             pstmtInserirBase.setString(2, base.getCidade());
209             pstmtInserirBase.setString(3, base.getIp());
210             pstmtInserirBase.setInt(4, base.getPorta());
211             pstmtInserirBase.setString(5, base.getSsid1());
212             pstmtInserirBase.setString(6, base.getSsid2());
213             pstmtInserirBase.setString(7, base.getSsid3());
214             pstmtInserirBase.setString(8, base.getSsid4());
215             pstmtInserirBase.setString(9, base.getUsuario());
216             pstmtInserirBase.setString(10, base.getSenha());
217             pstmtInserirBase.setInt(11, 0);
218
219             pstmtInserirBase.execute();
220
221             return true;
222         }
```

Listagem 6 – incluirBase(Base base)

A interface do Cadastro de Base possui um botão “Listar” que abre uma nova janela (ListaBase) na qual são exibidas em uma lista todas as bases cadastradas no banco de dados. A partir dessa lista é possível ver detalhes das bases, editar e excluir as mesmas. A Listagem 7 mostra como as bases são listadas na tela. Foi declarado um objeto do tipo “OperacoesDao” e um ArrayList do tipo “Base”. A linha 44 executa o método “listarBase()” da Classe “OperacoesDao”. Esse método retorna uma *array* de bases. Em seguida é percorrido esse *array* e listado as informações em um *list* na tela.

```
23      * @author Geovani
24      */
25    public class ListaBase extends javax.swing.JFrame {
26
27        OperacoesDao dao;
28        Usuario usuLogado;
29        ArrayList<Base> lista = null;
30
31        /** Creates new form ListaBases */
32        public ListaBase() {
33            initComponents();
34            setTitle("Lista de Bases");
35            setIconImage(new ImageIcon(getClass().getResource("/gui/image/signal.png")).getImage());
36
37            try {
38                dao = new OperacoesDao();
39            } catch (Exception ex) {
                  JOptionPane.showMessageDialog(this, "Erro: " + ex.getMessage());
41            }
42
43            try {
44                lista = dao.listarBase();
45
46                for (int i = 0; i < lista.size(); i++) {
47
48                    Base base = lista.get(i);
49                    listaDeRegistros.add("Nome base: " + base.getNome() + " Cidade: " + base.getCidade()
50                            + " IP: " + base.getIp());
51
52                }
```

Listagem 7 – Listar bases

O botão “editar” da interface ListaBase chama uma nova tela e passa por parâmetro a base selecionada. Nessa nova interface os valores da base passada por parâmetro são atribuídos aos campos. O usuário pode fazer as alterações que achar necessário e em seguida salvar. O botão “Salvar” irá chamar a função (Listagem 7) que criará, instanciará e atribuirá os valores a base. A linha 327 executa o método “editarBase(Base base)” (Listagem 8) da Classe “OperacoesDao” passando a base por parâmetro.

```
309     private void editarBase() {
310
311         Base base = new Base();
312
313         base.setNome(tfNomeBase.getText());
314         base.setCidade(tfcidade.getSelectedItem().toString());
315         base.setIp(tfIP.getText());
316         base.setPorta(Integer.parseInt(tfPorta.getText()));
317         base.setSsid1(tfSSID1.getText());
318         base.setSsid2(tfSSID2.getText());
319         base.setSsid3(tfSSID3.getText());
320         base.setSsid4(tfSSID4.getText());
321         base.setUsuario(tfUsuario.getText());
322         base.setSenha(tfSenha.getText());
323         base.setId(Integer.parseInt(lbId.getText()));
324
325         try {
326
327             boolean ret = dao.editarBase( base );
328
329             if ( ret == true ) {
330                 JOptionPane.showMessageDialog( this, "Alteração Efetuada com Sucesso" );
331             } else {
332                 JOptionPane.showMessageDialog( this, "Erro na Alteração" );
333             }
334
335         } catch( Exception e ) {
336             JOptionPane.showMessageDialog( this, "Erro:" + e.getMessage() );
337         }
338     }
```

Listagem 8 – editarBase()

O prepareStatement (Listagem 9) prepara o SQL para editar o cadastro no banco, o método “editarBase(Base base)” recebe uma base por parâmetro e atribui os valores ao prepareStatement que ao final faz o “executeUpdate()”.

```
79          pstmtEditarBase = con.prepareStatement(
80                  "UPDATE mktclient.base SET nome = ?, cidade = ?, ip = ?, porta = ?, "
81                  + "ssid1 = ?, ssid2 = ?, ssid3 = ?, ssid4 = ?, usuario = ?, senha = ?  WHERE id = ?");
    @Override
347 public boolean editarBase(Base base) throws Exception {
348
349         pstmtEditarBase.setString(1, base.getNome());
350         pstmtEditarBase.setString(2, base.getCidade());
351         pstmtEditarBase.setString(3, base.getIp());
352         pstmtEditarBase.setInt(4, base.getPorta());
353         pstmtEditarBase.setString(5, base.getSsid1());
354         pstmtEditarBase.setString(6, base.getSsid2());
355         pstmtEditarBase.setString(7, base.getSsid3());
356         pstmtEditarBase.setString(8, base.getSsid4());
357         pstmtEditarBase.setString(9, base.getUsuario());
358         pstmtEditarBase.setString(10, base.getSenha());
359         pstmtEditarBase.setInt(11, base.getId());
360
361         int qtd = pstmtEditarBase.executeUpdate();
362
363         if (qtd == 0) {
364             return false;
365         } else {
366             return true;
367         }
368 }
```

Listagem 9 – editarBase(Base base)

O botão “Excluir” da interface Lista Base chama uma função (Listagem 10) responsável por identificar a base selecionada na lista e em seguida chamar o método “excluirBase(int cod) ” da Classe “OperacoesDao” passando o código da base a ser excluída por parâmetro.

```
210 private void excluirBase() {
211
212         int indice = listaDeRegistros.getSelectedIndex();
213         Base base = lista.get(indice);
214
215         int status = JOptionPane.showConfirmDialog(null, "Deseja excluir a Base Selecionada ?", "Atenção",
216                 JOptionPane.YES_NO_OPTION);
217
218         if (status == JOptionPane.YES_OPTION) {
219
220             boolean retorno = false;
221             try {
222                 retorno = dao.excluirBase(base.getId());
223             } catch (Exception ex) {
224                 JOptionPane.showMessageDialog(this, "Erro: " + ex.getMessage());
225             }
226
227             if (retorno == true) {
228                 JOptionPane.showMessageDialog(this, "Base excluida com sucesso!");
229                 this.dispose();
230             } else {
231                 JOptionPane.showMessageDialog(this, "Erro ao excluir base.");
232             }
233         }
234 }
```

Listagem 10 – excluirBase()

O prepareStatement (Listagem 11) prepara o SQL para excluir o cadastro no banco, o método “excluirBase(int cod)” recebe o código por parâmetro e ao final faz o “executeUpdate()”;

```
77          pstmtExcluirBase = con.prepareStatement(
78                  "DELETE FROM mktclient.base WHERE id= ?");
        @Override
334     public boolean excluirBase(int cod) throws Exception {
335         pstmtExcluirBase.setInt(1, cod);
336
337         int ret = pstmtExcluirBase.executeUpdate();
338
339         if (ret == 0) {
340             return false;
341         } else {
342             return true;
343         }
344     }
```

Listagem 11 – excluirBase(int cod)

A seguir é apresentado sobre a conexão do sistema com o equipamento de rede. Essa talvez seja a principal funcionalidade do software desenvolvido. Para essa etapa foi necessário a utilização de uma API(libAPI.jar) e das classes responsáveis pela conexão, envio de comandos e recebimento de informações. Tanto a API quanto as classes estão disponíveis na página do fabricante do equipamento (http://wiki.mikrotik.com/wiki/Manual:API).

A API permite aos desenvolvedores criar soluções de *software* para interagir com o *RouterOS,* que é o sistema operacional dos equipamentos *routerboard*. Para o funcionamento da API é necessário que esteja instalado no equipamento de rede uma versão RouterOS 3.x ou superior. A conexão é feita através da porta padrão reservada para esse serviço, a 8728.

A comunicação ocorre por meio de comandos que são enviados ao roteador e retornam com as mensagens de resposta. Os comandos utilizados na conexão através da API são semelhantes aos usados no terminal do RouterOS, porém alguns são particulares da API java.

Alguns exemplos de comandos API utilizados no sistema:

a) O comando a seguir acessa o *access-list* do equipamento e solicita todos os registros. O proplist fica encarregado de retornar apenas as propriedades solicitadas.

```
"/interface/wireless/access-list/getall\n"
+ "=.proplist=comment,mac-address,interface,signalrange,authentication,forwarding"
```

b) Esse faz uma consulta no *access-list* buscando apenas o atributo “id” em que o “mac-address” é igual ao passado por parâmetro.

```
"/interface/wireless/access-list/getall\n"
    + "=.proplist=.id\n"
    + "?mac-address=" + macAddress.toUpperCase();
```

c) Acessa o *access-list* e remove o registro no qual o “id” é igual ao passado por parâmetro.

"/interface/wireless/access-list/remove\n"

+ "=.id=" + id;

d) Desabilita o registro do *access-list* usando o "id"

"/interface/wireless/access-list/disable\n"

+ "=.id=" + id;

e) Habilita o registro do *access-list* usando o "id"

"/interface/wireless/access-list/enable\n"

+ "=.id=" + id;

Como mencionado foram utilizadas duas classes fornecidas pelo fabricante do equipamento *routerboard*. As classes são encontradas no wiki da API java no site do fabricante.

A classe "OperacoesMkt" que esta no pacote "mkt" possui alguns métodos para obter a conexão com o equipamento, enviar comandos e receber os retornos. Os métodos dessa classe são:

a) Connect: espera um IP e a porta de conexão, estabelece a conexão. Na Listagem 12 está o código desse método.

```
33      // Conexão com o Routerboard
34      public boolean connect(String ip, int porta) {
35          boolean ret = false;
36          try {
37              socket = new Socket(ip, porta);
38              in = new DataInputStream(socket.getInputStream());
39              out = new DataOutputStream(socket.getOutputStream());
40              ret = true; // se conectar retorna true
41          } catch (UnknownHostException ex) {
42              JOptionPane.showMessageDialog(null, "Erro: " + ex.getMessage());
43          } catch (IOException ex) {
44              JOptionPane.showMessageDialog(null, "Erro: " + ex.getMessage());
45          }
46          return ret; // se não conectar retorna false
47      }
```

Listagem 12 – Conectar RouterOS

b) Login: espera o *login* e a senha de acesso, também utiliza a classe "MessageDigestHelper" presente no mesmo pacote para estabelecer o acesso. A Listagem 13 representa esse método.

```
49      //Login
50      public boolean login(String login, String password) {
51          String[] readIn;
52          String transition = "";
53          boolean ret = false;
54          writeln("/login");
55          readIn = read();
56          for (int i = 0; i < readIn.length; i++) {
57              System.out.println(readIn[i]);
58          }
59          String[] tmp = readIn[2].split("=ret=");
60          transition = tmp[tmp.length - 1];
61          String chal = "";
62          chal = MessageDigestHelper.myHexStrToStr("00") + password + MessageDigestHelper.myHexStrToStr(transition);
63          chal = MessageDigestHelper.myMD5Helper(chal);
64          write("/login");
65          write("=name=" + login);
66          writeln("=response=00" + chal);
67          readIn = read();
68          for (int i = 0; i < readIn.length; i++) {
69              System.out.println(readIn[i]);
70          }
71          if (readIn[readIn.length - 1].contains("!done")) {
72              if (readIn[1].contains("!trap")) {
73                  return false;
74              } else {
75                  return true;
76              }
77          }
78          return ret;
79      }
```

Listagem 13 – Login com RouterOS

c) Writeln: é utilizado para enviar os comandos para o RouterOS;

d) Write: envia apenas uma linha de comando;

e) Read: faz a leitura do material retornado pela API RouterOS;

f) progTalk: esse método (Listagem 13), talvez o mais usado é utilizado para enviar os comandos e ao mesmo tempo retorna um *array* de *string* com o material de resposta da API RouterOS. Esse método faz uso dos métodos writeln(), write() e read().

```
137     //é utilizado para enviar comandos e receber os resultados
138     public String[] progTalk(String text) {
139         text = text.trim();
140         int counter = 0;
141         String tag = "";
142         if (text.contains("\n")) {
143             tag = text.substring(0, text.indexOf("\n"));
144         } else {
145             tag = text;
146         }
147         text = text.concat("\n.tag=" + tag);
148         String[] internalText = text.split("\n");
149         while (counter < internalText.length - 1) {
150             write(internalText[counter]);
151             counter++;
152         }
153         writeln(internalText[counter]);
154         return read();
155     }
156
```

Listagem 14 – Enviar comandos

g) Disconnect: esse método apenas desconecta o *socket*.

Todas esses métodos listados são utilizados pelo sistema para estabelecer a conexão e interagir com o RouterOS. A Listagem 15 exemplifica o uso desses métodos pelo sistema.

```
371     private void listarClientesBase() {
372
373         conectarMKT();
374
375         String[] progTalk = utilMkt.progTalk("/interface/wireless/registration-table/getall\n"
376                 + "=.proplist=comment,mac-address,interface,signal-to-noise,uptime");
377
378         int linhaTabela = 0;
379         for (int i = 1; i < progTalk.length; i++) {
380
381             if (progTalk[i].contains("=comment=")) {
382                 String comment = progTalk[i].replace("=comment=", ""); // recebe a string da posição i e remove o comment
383                 tbBase.getModel().setValueAt(comment, linhaTabela, 0); // imprime na linha e coluna 0
384             } else if (progTalk[i].contains("=mac-address=")) {
385                 String mac = progTalk[i].replace("=mac-address=", "");
386                 tbBase.getModel().setValueAt(mac, linhaTabela, 1);
387             } else if (progTalk[i].contains("=interface=")) {
388                 String ssid = progTalk[i].replace("=interface=", "");
389                 tbBase.getModel().setValueAt(ssid, linhaTabela, 2);
390             } else if (progTalk[i].contains("=signal-to-noise=")) {
391                 String signal = progTalk[i].replace("=signal-to-noise=", "");
392                 tbBase.getModel().setValueAt(signal, linhaTabela, 3);
393             } else if (progTalk[i].contains("=uptime=")) {
394                 String uptime = progTalk[i].replace("=uptime=", "");
395                 tbBase.getModel().setValueAt(uptime, linhaTabela, 4);
396                 linhaTabela++;
397             }
398         }
399         utilMkt.disconnect();
400     }
```

Listagem 15 – Exemplo do uso dos métodos

Essa função mostrada no exemplo da Listagem 16 está presente na interface “ClientesOnlineBase”. Consiste em conectar ao equipamento e listar os registros dos clientes que estão *online* na base no momento da conexão. A linha 373 chama uma função (Listagem 15) que fica responsável por conectar ao equipamento. A linha 375 possui uma variável do tipo *array* de *string* que espera o retorno do comando passado pela API. O comando busca todos os registros no “registration-table” do equipamento retornando apenas os atributos listados no *proplist*. Em seguida é percorrido esse *array* de retorno e atribuído às informações a uma tabela. Por fim é desconectado do equipamento.

```
451      private void conectarMKT() {
452          Base base2 = new Base();
453          try {
454
455              base2 = dao.pesquisarBasePeloNome(cbBase.getSelectedItem().toString()); //pesquisa no BD qual é a bas
456          } catch (Exception ex) {
457              JOptionPane.showMessageDialog(null, "Nenhuma base foi selecionada");
458          }
459
460          String ip = base2.getIp();
461          String user = base2.getUsuario();
462          String senha = base2.getSenha();
463          int porta = base2.getPorta();
464
465          boolean retorno;
466
467          retorno = utilMkt.connect(ip, porta); // retorna true ou false
468          if (retorno == false) {
469              JOptionPane.showMessageDialog(this, "O sistema não consegui se conectar ao equipamento, "
470                      + "verifique se a base esta online");
471          }
472
473          retorno = utilMkt.login(user, senha); // retorna true ou false
474          if (retorno == false) {
475              JOptionPane.showMessageDialog(this, "O usuário ou a senha da base esta errada no cadastro");
476          }
477
478      }
479  }
```

Listagem 16 – ConectarMKT()

Essa função (Listagem 16) verifica qual é a base selecionada pelo usuário, faz uma conexão com o bando de dados e retorna todas as informações dessa base necessárias para em seguida estabelecer a conexão (linha 467) e fazer a *login* (linha 473).

5 CONCLUSÃO

Com o desenvolvimento deste projeto foi possível colocar em prática o conhecimento adquirido nas disciplinas cursadas no curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas da UTFPR, Câmpus Pato Branco. Independentemente da linguagem, a lógica para implementar um sistema é a mesma. E todas as matérias e os professores contribuíram de certa forma para o desenvolvimento desse projeto.

Diversas dificuldades surgiram durante o desenvolvimento do sistema, tanto na modelagem quanto na implementação. É possível que isso tenha ocorrido pelo autor deste trabalho não trabalhar na área e assim não ter lógica tão apurada e nem fluência com codificação. Contudo, isso não foi utilizado como justificativa, seguiu-se em frente buscando busquei relembrar alguns conceitos em exemplos e vídeo-aulas guardadas do curso.

A conexão com o equipamento *routerboard* foi o ponto forte do projeto. Sem nunca ter desenvolvido nada parecido foi a oportunidade de testar o conhecimento adquirido no curso e buscar entender a API para então fazer a conexão com o equipamento e a troca de informações. Os comandos não seriam problema, pois a sintaxe é praticamente a mesma do terminal *RouterOS*, do qual já tinha conhecimento. Tendo em mãos o manual da API disponibilizado no site do fabricante o desenvolvimento deste projeto foi a oportunidade de verificar algumas particularidades da mesma e encontrar as soluções.

O grande facilitador no período de desenvolvimento foi a própria linguagem Java que por meio do seu JavaDoc auxiliou nas muitas dúvidas e na solução de problemas. A ferramenta de desenvolvimento NetBeans também foi de grande valia pela facilidade de sua interface, localização de erros e depurador.

Por fim, como autor deste trabalho, fica-se satisfeito com o produto final do trabalho de conclusão. E espera-se que esse seja apenas o início para a continuidade do projeto de um *software* para provedores de Internet.

Dentre os trabalhos futuros para melhoria e agregação de funcionalidades ao sistema desenvolvido podem ser citados: o armazenamento de *logs* para o controle do que foi alterado por cada um dos usuários e melhorias na interface.

REFERÊNCIAS

BEZERRA, Eduardo**. Principios de análise e projeto de sistemas com UML**. Campus, 2006.

BOOCH, Grady. **Object-oriented analysis and design**, 2ª edição, Addison-Wesley, 1998.

BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**, 2ª edição, Rio de Janeiro: Campus, 2000.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário**, 7ª edição**,** Rio de Janeiro: Campus, 2000,

FERRANTE, Agustin Juan, RODRIGUEZ, Vicente Rodrigues y. **Tecnologia de informação e gestão empresarial**. Rio de Janeiro: E-Papers, 2000

MACORATTI, José Carlos. **UML - Conceitos Básicos II**. Disponível em <http://www.macoratti.net/vb_uml2.htm>. Acesso em: 27 jul. 2012.

MYSQL-FRONT. **Mysql_Front**. Disponível em: <http://www.mysqlfront.de/>. Acesso em: 04 set. 2012.

OMG. **Unified Modeling Language (OMG UML), Superstructure**, Version 2.4.1, 2011.

PRESSMAN, Roger. **Engenharia de software**, 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

MARTIN, James. **Análise e projeto orientados a objeto.** São Paulo: Makron Books, 1995.

SCOTT, Kendall. **O processo unificado explicado**, Bookamn, 2003. Disponível em <http://booklens.com/kendall-scott/processo-unificado-explicado>. Acesso em 20 jun. 2012.

SILVA, Ricardo Pereira E. **Como modelar com a UML 2**. Florianópolis: Visual Books, 2009.

SILVA, Ricardo Pereira E. **UML 2 em modelagem orientada a objetos**. Florianópolis: Visual Books, 2007.