DESENVOLVIMENTO DE APLICATIVO GERENCIADOR DE CONTROLE E ACIONAMENTO PARA PLATAFORMA iOS[1]

Diego Baltieri[2]
Wladimir da Costa[3]

RESUMO

Ter uma ferramenta que gerencie os controles e acionamentos do setor privado por meio de um aplicativo móvel capaz de se comunicar com componentes eletrônicos e realizar acionamento de portões, lâmpadas, alarmes e câmeras traz comodidade e segurança aos profissionais do setor de vigilância privada. Torna-se difícil realizar essa tarefa quando o processo é feito de forma manual. Com o objetivo de solucionar esta situação, desenvolveu-se um aplicativo denominado *Connect*, para a plataforma iOS, que tem como principal função o gerenciamento dos controles e acionamentos. Foi utilizado neste trabalho o conceito de engenharia de s*oftCware*, padrões de projeto, banco de dados, UML e processo unificado da *Rational*.
Palavras-chave: Aplicativo, iOS, vigilância privada, banco de dados, controle, monitoramento, UML, RUP

ABSTRACT

Having a tool that manages the controls and drives the vigilant security services through a mobile application that is able to communicate with electronic components and perform drive gates, lights, alarms and cameras bring convenience and safety to all vigilant security professional. It is very difficult to accomplish this task when the process is done just manually. In order to solve the situation, we developed a prototype application called Connect to the iOS platform, which the main function is the management of controls and drives. We present in this work the concept of software engineering, design patterns, database, UML and unified process of Rational.
Keywords: *App, iOS, vigilant security service, database, controls, monitoring, UML, RUP*

1 INTRODUÇÃO

A necessidade de garantir segurança às pessoas e patrimônios é cada vez maior devido aos altos índices de criminalidade, atraindo para este mercado produtos e soluções mais modernas, eficazes e que atendam às demandas do setor de segurança privada. (MaxPress, 2014)

O conceito de segurança privada surge com o "Estatuto da Segurança Privada" que é a lei n. 7102/83, onde temos a definição desta modalidade de segurança:

> Artigo 10. São considerados como segurança privada as atividades desenvolvidas em prestação de serviços com a finalidade de:
> - Proceder à vigilância patrimonial das instituições financeiras e de outros estabelecimentos, públicos ou privados, bem como a segurança de pessoas físicas; e,
> - Realizar o transporte de valores ou garantir o transporte de qualquer outro tipo de carga. (Couto, 2012)

Portanto, tornase necessário automatizar o gerenciamento das atividades neste setor, a fim de assegurar dados e bens patrimoniais, agilidade e eficiência às tarefas realizadas pelos profissionais da segurança privada. A solução proposta no projeto é desenvolver um aplicativo móvel para o setor de segurança privada, que auxilie o gerenciamento de controle e acionamento, e melhore o sistema de monitoramento.

Este trabalho tem como objetivo desenvolver um aplicativo gerenciador de controle e acionamento para auxiliar profissionais do setor de segurança privada, denominado *Connect*.

Os objetivos específicos são:

- ❖ Fazer uma pesquisa do referencial teórico para o desenvolvimento do aplicativo;
- ❖ Admitir os requisitos funcionais e não funcionais do aplicativo;
- ❖ Projetar o banco de dados relacional; e,
- ❖ Desenvolver um protótipo do aplicativo.

O aplicativo será desenvolvido fazendo o uso de processos de engenharia de *software Rational*

[1] Artigo baseado em Trabalho e Conclusão de Curso (TCC) desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, depositado em 19 de dezembro de 2015
[2] Tecnólogo em Tecnologia em Análise e Desenvolvimento de Sistemas – Fatec Americana – Centro Estadual de Educação Tecnológica Paula Souza ; Contato: diegobaltieri@me.com
[3] Prof. Coordenador Fatec Americana – Mestre em Ciência da Computação; Contato: cwladi@hotmail.com

Unified Process (RUP), por se tratar de um processo complexo, será feito o uso de apenas parte do processo RUP, de forma customizada nesta aplicação.

O RUP apresenta algumas das melhores práticas para um bom desenvolvimento de como desenvolver o *software* de forma iterativa, gerando requisitos, usando arquiteturas baseadas em componentes, modelagem de *software* de forma visual, análise de qualidade da aplicação e controle de alterações do *software* (IBM, 2001).

A modelagem do *software* será realizada com a utilização das ferramentas BrModelo (versão 2.0, 2015) para que possam proporcionar um padrão para a arquitetura do sistema de *software*. As linguagens que serão utilizadas para a implementação serão *Swift* (versão 2.1, 2015) e PHP (5.1, 2015) e suas respectivas IDEs *Xcode* (versão 7, 2015), *NetBeans* (versão 8.8, 2015), foi selecionado um sistema gerenciador de banco de dados livre, o MySQL (versão 5.1, 2015).

O projeto está estruturado em quatro seções, cujo conteúdo é sucintamente apresentado a seguir:

Na seção 2 é feito o levantamento teórico para desenvolvimento do aplicativo gerenciador de controle e acionamento *Connect*: – são apresentados os conceitos envolvidos no desenvolvimento de *software*.

A seção 3 - ferramentas envolvidas no desenvolvimento: – apresenta quais as ferramentas selecionadas para auxiliar no desenvolvimento da aplicação.

Na seção 4 - o sistema proposto: – demonstra o processo de desenvolvimento de *software* composto

Finalmente, a seção 5 apresenta as conclusões deste trabalho a partir da análise dos resultados obtidos e trabalhos futuros.

2 ENGENHARIA DE *SOFTWARE*

Esta seção tem como objetivo apresentar os conceitos básicos para o desenvolvimento do aplicativo gerenciador de controle e acionamento - *CONNECT*

Será apresentada nesta seção a definição de engenharia de *software*, processo unificado da *Rational* e ciclo de vida de um *software*.

Definição

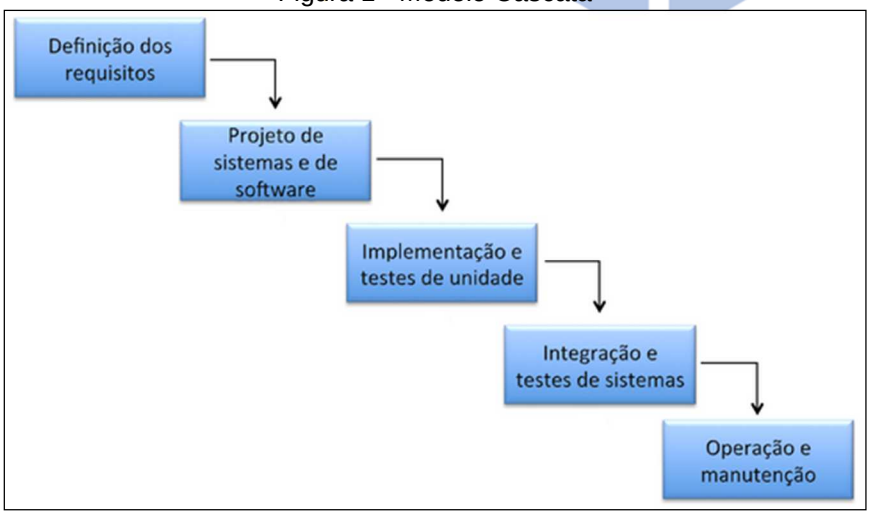
Engenharia de *software* é a área da computação que está relacionada com todos os aspectos de produção do *software*, desde a criação, construção, análise, desenvolvimento e manutenção. (SOMMERVILLE, 2007).

Ciclo de vida de desenvolvimento

Para compreender o ciclo de vida do *software*, é necessário enxergar que um *software* nunca será completamente finalizado, mesmo já estando pronto para ser utilizado, haverá sempre necessidade de realizar futuras implementações (REZENDE, 2005).

As etapas que o ciclo de vida do *software* abrange são: análise, projeto, implementação, testes e manutenção. Essas cinco etapas são denominadas de outra forma no modelo cascata, também conhecido como modelo de ciclo de vida, como mostra a figura 1 (SOMMERVILLE, 2007).

Figura 1 - Modelo Cascata



Fonte: Adaptada de Sommerville (2007)

Para uma melhor especificação do aplicativo, se faz necessário que todos os requisitos e finalidades do sistema sejam recolhidos na primeira fase: definição de requisitos. Na segunda, a arquitetura é definida, tanto em aspecto de *software*, com os requisitos funcionais, quanto em aspecto de *hardware*, com os requisitos não funcionais.

Na terceira fase, será feita a implementação e testes, o *software* começa a ser implementado, de modo a verificar se realmente está atendendo a sua finalidade. O *software* é testado de uma forma geral na fase de integração e testes. E na fase de operação e manutenção, após a entrega, dele pode ser requerida novas alterações, relacionadas a problemas não detectados nos testes (SOMMERVILLE, 2007).

A figura 2 representa o ciclo de vida do *software*. Como já mencionado, um *software* nunca será completamente finalizado, sempre estará em desenvolvimento, pois durante seu uso haverá sempre a necessidade de realizar modificações (BROOKSHEAR, 2003).

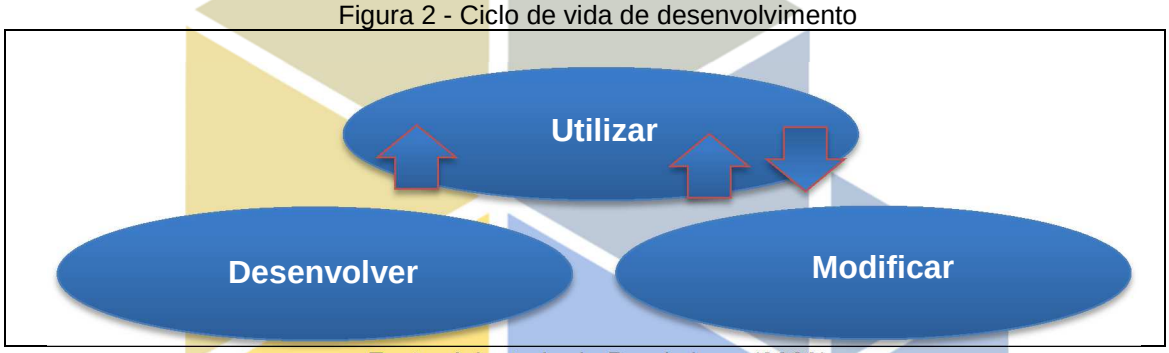
Toda fase de manutenção é concentrada em:

- Correções: alterações que estão associadas à correção de defeitos (manutenção corretiva);
- Adaptações: são exigidas ao decorrer da evolução do ambiente do sistema (manutenção adaptativa); e,
- Aprimoramentos funcionais produzidos por exigências variáveis do cliente (manutenção perfectiva).

A prevenção:

- Os aplicativos móveis se deterioram devido a modificações;
- Surge a manutenção preventiva, mais conhecida como reengenharia de *software*; e,
- Em essência, a manutenção preventiva faz modificações nos aplicativos, de modo que eles possam ser mais facilmente corrigidos, adaptados e melhorados.

Figura 2 - Ciclo de vida de desenvolvimento



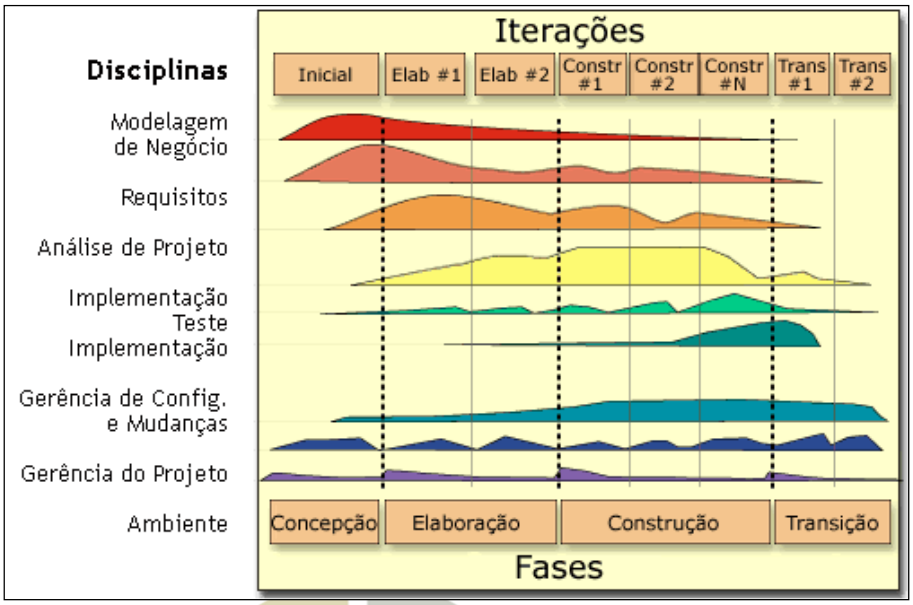
Fonte: Adaptada de Brookshear (2003)

Processo Unificado da Rational

O *Rational Unified Process* (RUP) que traduzido para o português significa processo unificado da *Rationa*l, é um processo de engenharia de *software* que aumenta a produtividade da equipe e oferece as melhores práticas relacionadas a *software* através de diretrizes, *templates* e orientações sobre ferramentas para todas as atividades críticas de desenvolvimento de *software*. Utiliza a linguagem UML no desenvolvimento dos casos de uso e a orientação a objetos (KRUCHTEN, 2003).

O RUP apresenta duas dimensões, a estática e a dinâmica, como é representado na figura 3 a seguir:

Figura 3 - As dimensões do RUP



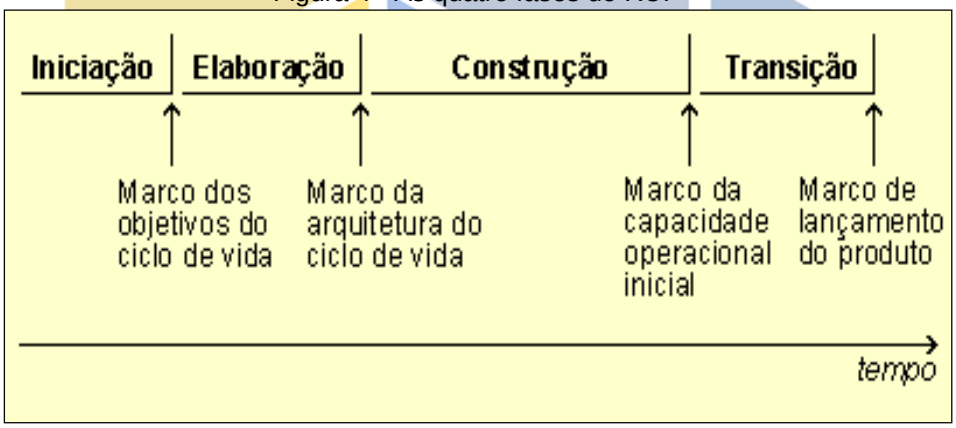
Fonte: Adaptada de Kruchten (2003)

O aspecto dinâmico do processo está representado na dimensão horizontal, o ciclo de vida. Essa dimensão faz com que o projeto do *software* seja elaborado com uma sequência de iterações incrementais. Já na dimensão vertical, representada no aspecto estático é descrito em termos de componentes do processo: atividades, disciplinas, artefatos e papéis do processo (KRUCHTEN, 2003).

Dimensão dinâmica

No aspecto dinâmico a arquitetura RUP apresenta quatro fases: concepção, elaboração, construção e transição, como é demonstrado na figura 4. (KRUCHTEN, 2003).

Figura 4 - As quatro fases do RUP



Fonte: Adaptada de Kruchten (2003)

A criação do escopo do projeto se dá na fase da concepção; nela é realizada uma estimativa dos recursos necessários, a avaliação de risco e o cronograma dos marcos temporais. Nesta fase também são identificados todos os atores envolvidos na iteração do sistema. São desenvolvidos, também, os seguintes artefatos:

- Caso de negócio inicial;
- Formulação do documento de visão geral dos requisitos do projeto;
- Relatório inicial de avaliação de risco;
- Estimativa de recursos;
- Cronograma inicial e;
- Protótipos iniciais.

A eliminação e especificação dos pontos que oferecem grandes riscos são feitos na fase de elaboração, onde alguns requerimentos são desenvolvidos:

- Modelo de casos de uso;

- Análise e projeto do sistema;
- Relatórios de riscos;
- Plano de gerenciamento;
- Alocação da equipe;
- Planejamento das fases, mostrando suas iterações e conteúdos; e,
- Realiza-se a análise do domínio do problema e projeta uma arquitetura para o sistema.

A construção é o início do desenvolvimento da aplicação, ao final de cada ciclo pode surgir uma versão utilizável do processo. Nesta fase são desenvolvidos alguns requerimentos:

- Sistema de *software* funcionando e documentação associada pronta para ser liberada aos usuários;
- Casos de teste baseados em cenários, e resultados de testes; e,
- Por último a transição: implantação do *software* e transferência do *software* para o usuário do sistema (KRUCHTEN, 2003).

Dimensão estática

Segundo Kruchten (2003), a dimensão estática possui quatro conceitos chaves que definem quem faz o quê, como e quando?

- *Workers* (trabalhadores) – quem;
- Atividades – como;
- Artefatos – o quê; e,
- Fluxos de trabalho (*workflows*) – quando.

O RUP possui também, algumas disciplinas, que são:

- **Modelagem do negócio** - disciplina que visa o entendimento da estrutura e a dinâmica da organização; o entendimento dos problemas e; a identificação das melhorias;
- **Requisitos** - disciplina que estabelece uma concordância entre os envolvidos no projeto sobre os requisitos do sistema, bem como todos os limites do sistema, estima o tempo e custo de desenvolvimento;
- **Análise e Projeto** - disciplina que visa transformar os requisitos em projeto; construir uma arquitetura para o sistema e; adaptar o projeto ao ambiente;
- **Implementação** – disciplina que visa estruturar o código, bem como implementar classes e objetos, testar e integrar as unidades;
- **Testes** - disciplina que testa o sistema para a identificação de erros, todos os erros são identificados e documentados; é possível validar o sistema de acordo com o que foi especificado e validar se o sistema foi desenvolvido de acordo com o projetado;
- **Implantação** - disciplina que visa a certificação da entrega do sistema ao usuário final;
- **Gerência de configuração e mudança** - disciplina que visa controlar os artefatos produzidos no desenvolvimento do projeto;
- **Gerenciamento e planejamento** - disciplina que controla o gerenciamento de riscos e disponibiliza guias para planejar, executar, acompanhar e monitorar o projeto; e,
- **Ambiente** - visa focar em atividades relacionadas à adaptação do processo organizacional do projeto.

Melhores práticas

No RUP, as melhores práticas apresentadas, descrevem como implantar um bom desenvolvimento de *software* para equipes desenvolvedoras. São elas:

Desenvolver aplicativo iterativamente

O RUP sugere uma abordagem iterativa do desenvolvedor, pois nela é necessária uma maior compreensão do problema através de melhoras sucessivas, para gerar de uma forma gradativa solução eficaz a cada iteração. A abordagem iterativa no desenvolvimento verifica os itens de maior risco em cada etapa do ciclo de vida, reduzindo significativamente o perfil de um projeto de risco, pois cada iteração termina com uma versão executável, o desenvolvedor permanece focado em produzir resultados e a checagem de status frequentes ajudam a garantir que o projeto permaneça dentro do cronograma. Uma abordagem iterativa também torna mais fáceis as mudanças táticas nos requisitos, nas funcionalidades e/ou no cronograma (IBMb, 1998).

Gerenciamento de requisitos

O RUP sugere ao desenvolvedor métodos de capturar os requisitos funcionais. Essa prática apresenta como obter, organizar o documento de funcionalidade e restrições; como acompanhar e documentar compromissos e decisões; e facilmente como capturar os requisitos de negócio (IBMb, 1998).

Uso de arquitetura baseada em componentes

O processo concentra-se no desenvolvimento inicial de uma arquitetura robusta executável. Ele descreve como projetar uma arquitetura flexível. O RUP apóia o desenvolvimento de *software* baseado em componentes (IBMb, 1998).

Modelagem visual de *software*

As abstrações visuais ajudam o desenvolvedor a se comunicar com os diferentes aspectos *software*; a ver como os elementos do sistema se encaixam; manter a coerência entre um projeto e sua execução; e promover a comunicação inequívoca (IBMb, 1998).

Verificar qualidade de *software*

Atualmente o fraco desempenho de aplicativos e a pouca confiabilidade são fatores comuns que impedem drasticamente a aceitação dos pedidos de *software*. Assim, a qualidade deve ser focada no que diz respeito aos requisitos baseados em confiabilidade, funcionalidade, desempenho da aplicação e desempenho do sistema. Esta prática auxilia no planejamento do projeto, na implementação, na execução e na avaliação dos testes. (IBMb, 1998).

Controle de alterações no *software*

Em todo *software* há mudanças, em alguns casos, estas mudanças são inevitáveis, por isso a capacidade de gerir a mudança e de ser capaz de acompanhar as mudanças é essencial. Esta prática descreve como controlar, acompanhar e monitorar as mudanças para permitir o desenvolvimento iterativo sucesso (IBMb, 1998).

A linguagem UML

A *Unified Modeling Language* TM (UML)® é uma especificação da *Object Management Group* (OMG)® (OMG, 1997 - 2011). É uma linguagem gráfica de modelagem para visualizar, especificar, construir e documentar os artefatos de sistemas de objetos distribuídos (UML, 2011).

A UML possui treze modelos gráficos que estão divididos em duas categorias, os diagramas de aplicações estáticas que representam a estrutura e os diagramas de comportamentos, no entanto dentro desta última categoria, existe uma subcategoria que compõe os diagramas de interação (SILVA, 2007).

A categoria de diagramas de estrutura inclui diagrama de classe, diagrama de objeto, diagrama de componentes, diagrama de estrutura composta, diagrama de pacote e diagrama de utilização (SILVA, 2007).

Os diagramas de comportamento são: diagrama de caso de uso, diagrama de máquina de estados e diagrama de atividades. Em sua subcategoria interação estão inclusos os diagramas de sequência, comunicação, visão geral de interação e por último, porém não menos importante o de temporização (SILVA, 2007).

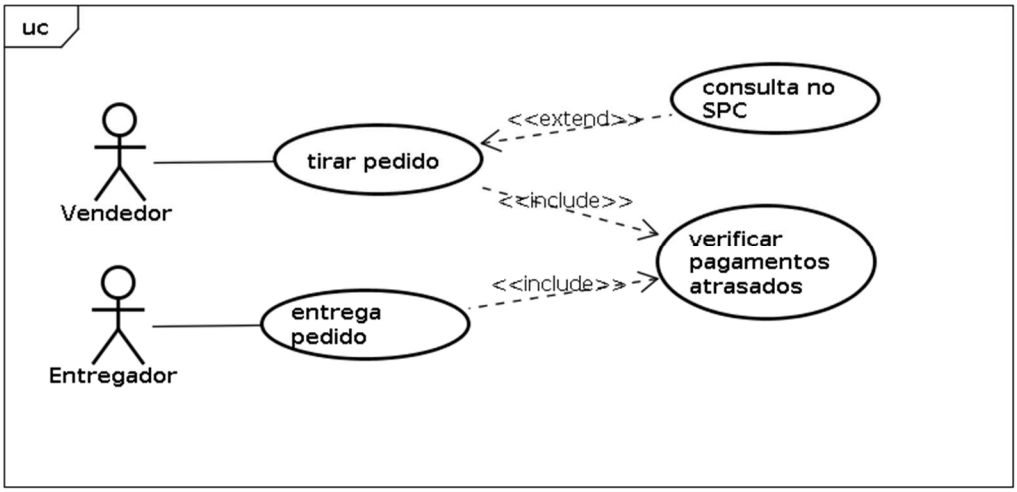
Principais diagramas UML

Os diagramas UML abordados neste projeto são os de: caso de uso, sequência, atividade, classe, estado e utilização.

Diagrama de caso de uso

O diagrama de caso de uso está relacionado à modelagem dinâmica do sistema. Ele é composto por elementos sintáticos denominados “atores” e relações que envolvem esses elementos (SILVA, 2007). A figura 5 apresenta um exemplo de um diagrama de caso de uso.

Figura 5 - Exemplo de diagrama de caso de uso

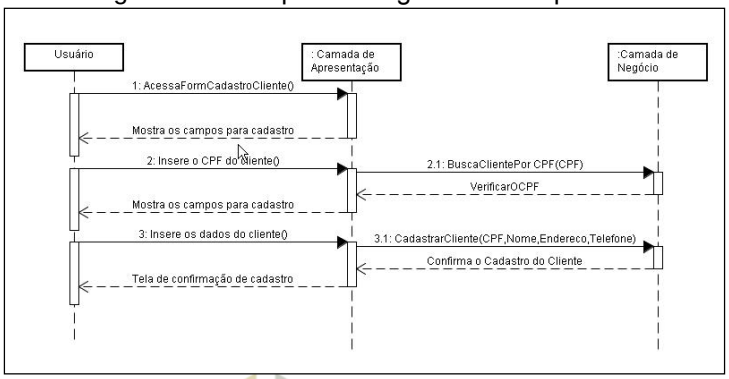


Fonte: Adaptada de Silva (2007)

Diagrama de sequência

O diagrama de sequência também está relacionado à modelagem dinâmica do sistema. E representa a iteração entre os objetos na troca de mensagens na ordem temporal em que elas acontecem. A leitura das mensagens que são enviadas de objetos para outros objetos é feita de cima para baixo (SILVA, 2007). A figura 6 apresenta um exemplo de um diagrama de sequência.

Figura 6 - Exemplo de diagrama de sequência

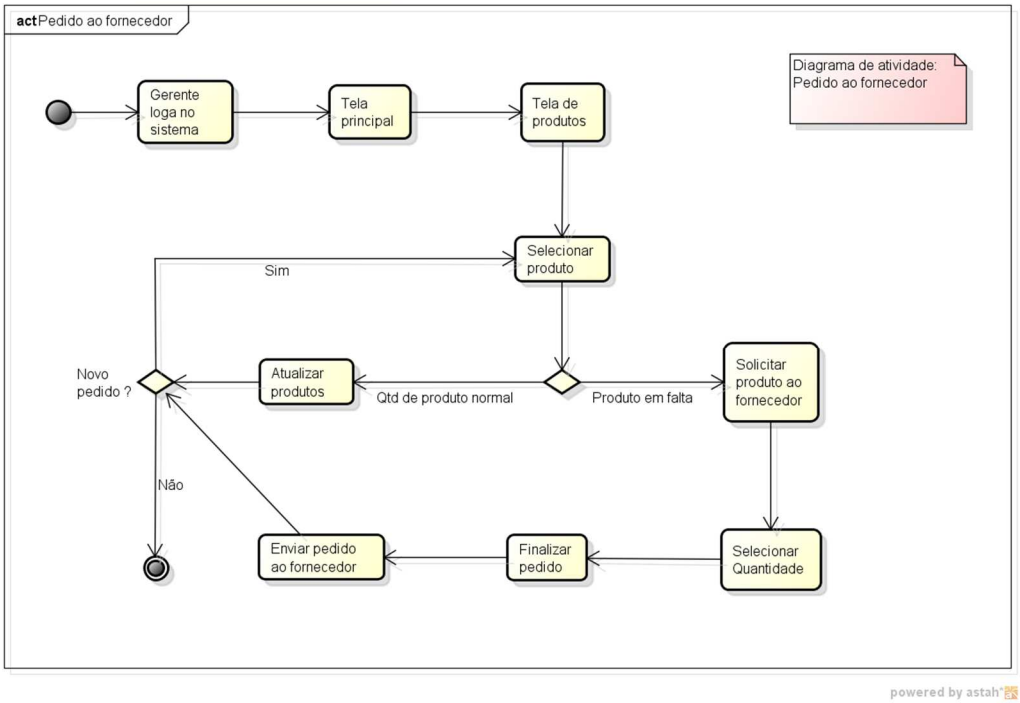


Fonte: Adaptada de Silva (2007)

Diagrama de atividade

O diagrama de atividades é composto por atividades, vista como um conjunto de atividades ou de ações. Este diagrama representa uma atividade correspondente ao sistema. Conforme mostrado na figura 7.

Figura 7 - Exemplo de diagrama de atividade

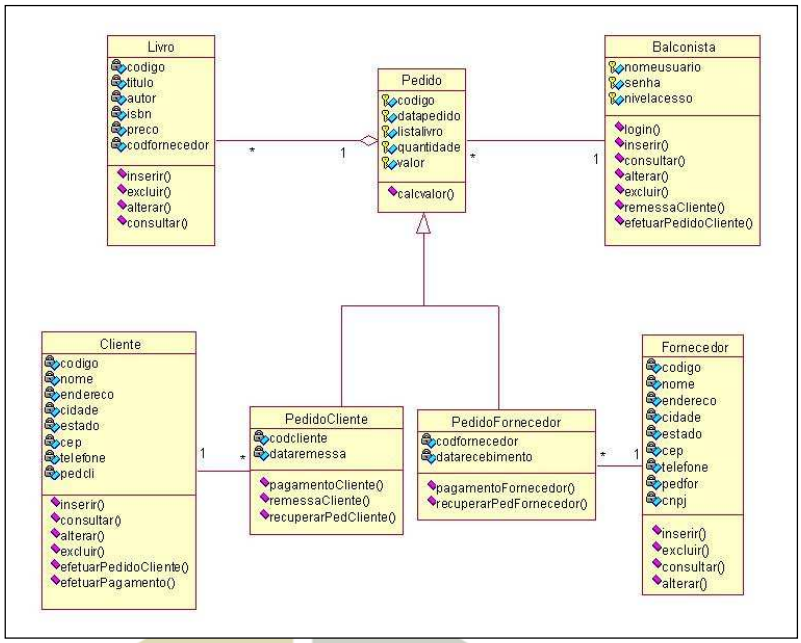


Fonte: Adaptada de Silva (2007)

Diagrama de classe

O diagrama de classes representa a estrutura e os relacionamentos das classes. No entanto, as classes e os relacionamentos cedidos a elas são os elementos sintáticos básicos do diagrama de classes (SILVA, 2007). A figura 8 apresenta um exemplo de um diagrama de classe.

Figura 8 - Exemplo de diagrama de Classe

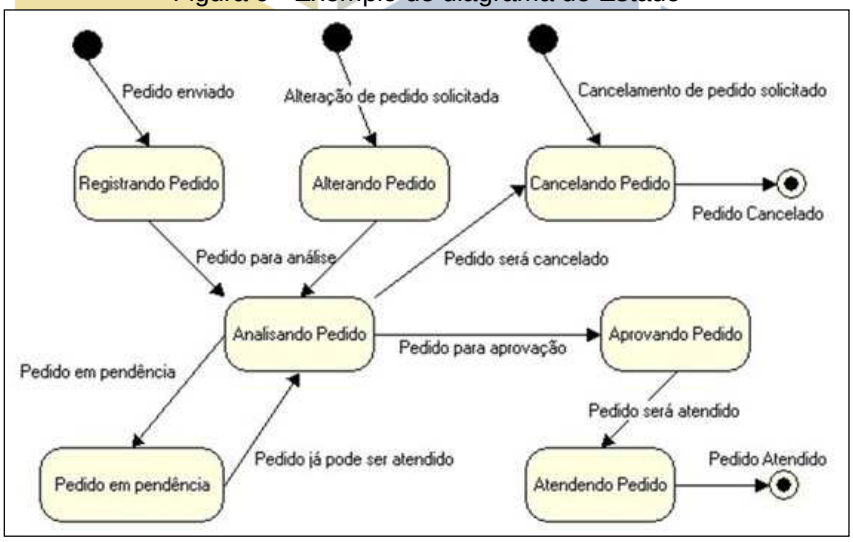


Fonte: Adaptada de Silva (2007)

Diagrama de estado

O diagrama de estado representa o estado em que um objeto se encontra no sistema. Com isso, os elementos principais desse diagrama são os estados e as transições. Um objeto muda de estado com o auxílio de uma transição (SILVA, 2007). A figura 9 apresenta um exemplo de um diagrama de estado.

Figura 9 - Exemplo de diagrama de Estado

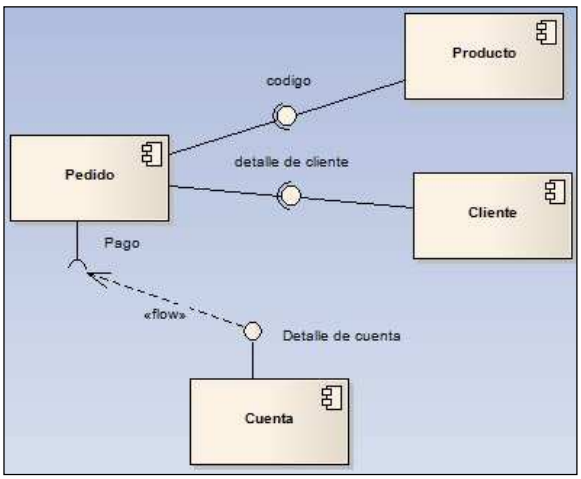


Fonte: Adaptada de Silva (2007)

Diagrama de componente

Este diagrama demonstra os componentes do sistema e a relação entre esses componentes. Assim o diagrama representa a classe organizada, especificando a qual classe pertence cada um dos componentes. Conforme a figura 10.

Figura 10 - Exemplo de diagrama de Componente



Fonte: Adaptada de Silva (2007)

Banco de dados

Banco de dados é um conjunto de dados persistentes, com o intuito de armazenar informações de uma determinada organização. Esses dados são mantidos por um *software* chamado de Sistema Gerenciador de Banco de Dados (SGBD), onde os usuários desse sistema podem realizar busca, exclusão, inserção e alteração nesses arquivos de banco de dados (DATE, 2003).

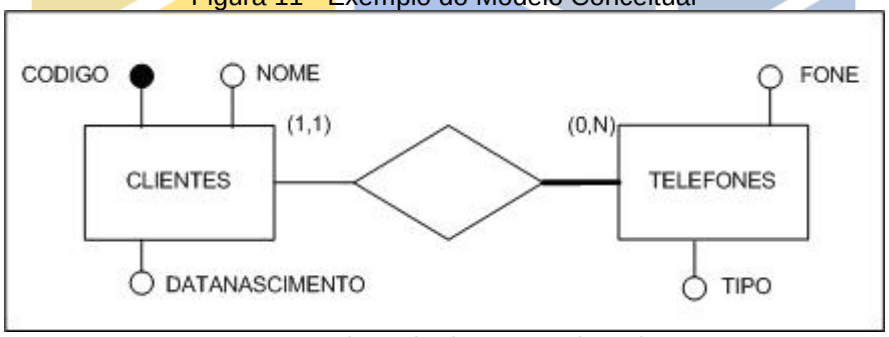
Alguns autores definem que dados e informações tem o mesmo significado, por outro lado, outros definem dados como os valores fisicamente armazenados no banco de dados e informações como o significado gerado a partir de um determinado dado (DATE, 2003).

Projeto de banco de dados

Na maioria dos projetos de banco de dados são utilizados dois modelos de banco de dados em duas fases: o modelo conceitual e o modelo lógico, porém existe mais um modelo: o físico.

Na primeira fase é abstraído o modelo conceitual. Este modelo descreve o banco de dados de forma independente, sem necessitar do SGBD. O projeto de banco de dados é representado em um Diagrama de Entidade Relacionamento (DER), como mostra a figura a seguir (HEUSER, 1998).

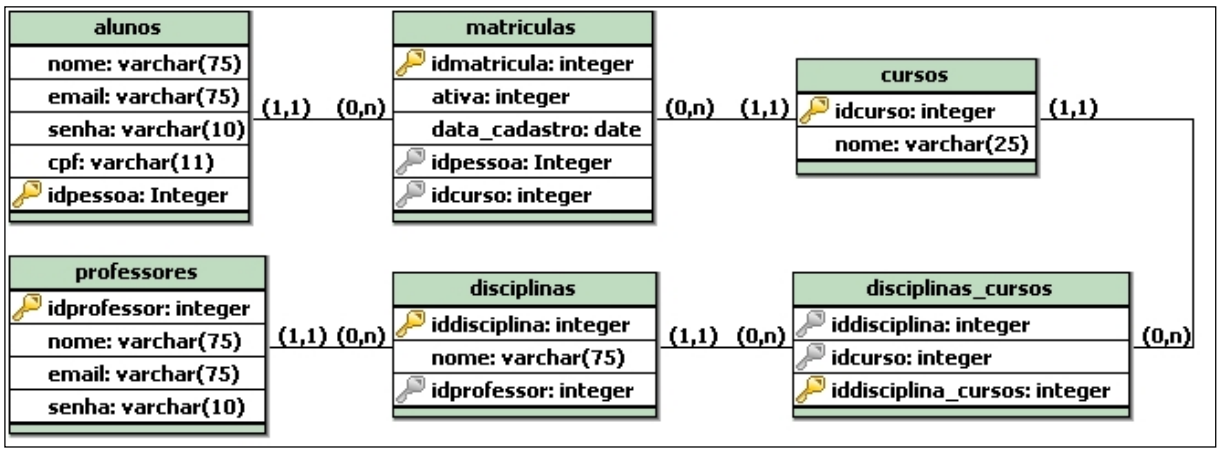
Figura 11 - Exemplo do Modelo Conceitual



Fonte: Adaptada de Heuser (1998)

E na segunda fase é abstraído o modelo lógico. Modelo que representa o banco de dados na maneira como é visto pelo usuário do SGBD. Ao contrário do conceitual, depende do SGBD utilizado no projeto. E é gerado a partir do modelo conceitual desenvolvido na primeira fase do projeto de banco de dados (HEUSER, 1998).

Figura 12 - Exemplo do Modelo Lógico



Fonte: Adaptada de Heuser (1998)

A linguagem SQL

A Structured Query Language (SQL) que traduzido para o português significa linguagem de consulta estruturada, é a linguagem padrão para definição e manipulação no banco de dados relacional (IBMc, 2011). É uma linguagem simples e de fácil uso (DAMAS, 2007).

A linguagem SQL se divide em três principais grupos: DDL, DML e DCL. A Data Definition Language – linguagem de definição de dados (DDL) trabalha com os objetos e tem os seguintes comandos: ALTER – altera um objeto do banco de dados (uma tabela, por exemplo), CREATE – cria um objeto na base de dados e DROP – apaga um objeto da base de dados; Os comandos ALTER e CREATE, podem ser usados para index (indices) e view (visões).

Outro grupo é a *Data Manipulation Language* – linguagem de manipulação de dados (DML), ela trabalha com as tuplas (linhas), seus comandos são SELECT – consulta os dados armazenados em uma tabela, INSERT – insere uma linha na tabela, DELETE– deleta e UPDATE – permite alterar quantas linhas de dados for preciso em uma tabela.

E por último, porém não menos importante, a Data Control Language – linguagem de controle de dados (DCL) trabalha com os utilizadores, controla o acesso aos dados, seus principais comandos são: GRANT – seta os privilégios, permite o acesso aos dados ao usuário e REVOKE – remove os privilégios dado ao usuário.

A linguagem SQL faz parte de uma das cinco gerações de linguagens, a quarta geração. Ela atende a quase todas as necessidades para o desenvolvimento de um banco de dados, porém para completar as necessidades que a SQL não atende, em algumas ocasiões o desenvolvedor concilia a linguagem SQL com alguma outra linguagem de programação (DAMAS, 2007).

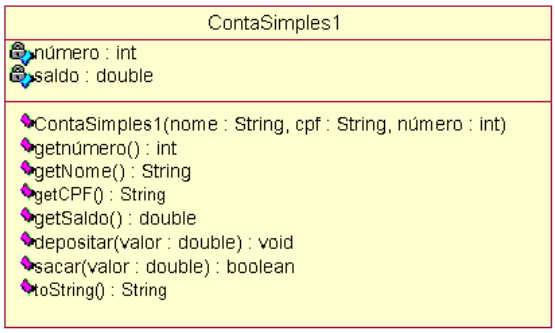
Orientação a objetos

A programação orientada a objetos é implementada pelo envio de mensagens a objetos. Ela reúne alguns conceitos que precisam ser entendidos antes iniciar uma programação orientada a objetos, como classe, objetos, herança e polimorfismo.

Classes

Uma classe é um modelo ou protótipo onde os objetos serão criados e definidos. Assim, a classe define o estado e o comportamento de um objeto físico do mundo real, como está representado na figura 13 (RICARTE, 2001).

Figura 13 - Exemplo de uma Classe



Fonte: Adaptada de Ricarte (2001)

Objetos

Objeto é uma instância de uma classe, sendo assim ele é um objeto físico do mundo real. São justamente os objetos que caracterizam a programação orientada a objetos. O objeto tem seus devidos atributos e métodos que o manipulam (RICARTE, 2001).

Polimorfismo

Polimorfismo consiste em implementar um código considerando classes abstratas ou interfaces, ao invés de classes concretas (SIERRA, 2010).

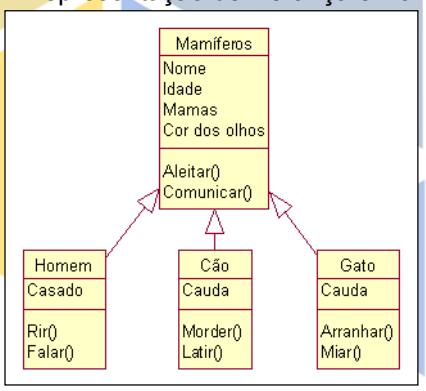
Herança

A herança organiza e estruturar o *software*. As classes herdam o estado e o comportamento de suas superclasses. Com a herança, classes podem herdar características da classe pai, como por exemplo, atributos e métodos. Podendo assim a subclasse, especificar ou estender a superclasse (RICARTE, 2001).

A figura 14 apresenta um exemplo de herança e polimorfismo. Foram criadas três classes de tipos diferentes de canetas, porem todas são derivadas de outra classe. As

Classes CanetaFlor, CanetaPalhaço e CanetaRelógio herdam os atributos e métodos (mesma assinatura) da classe Caneta. Porém as ações dos métodos são diferentes para cada uma das classes que estão herdando (MARIANI, 2010).

Figura 14 - Representação de Herança e Polimorfismo

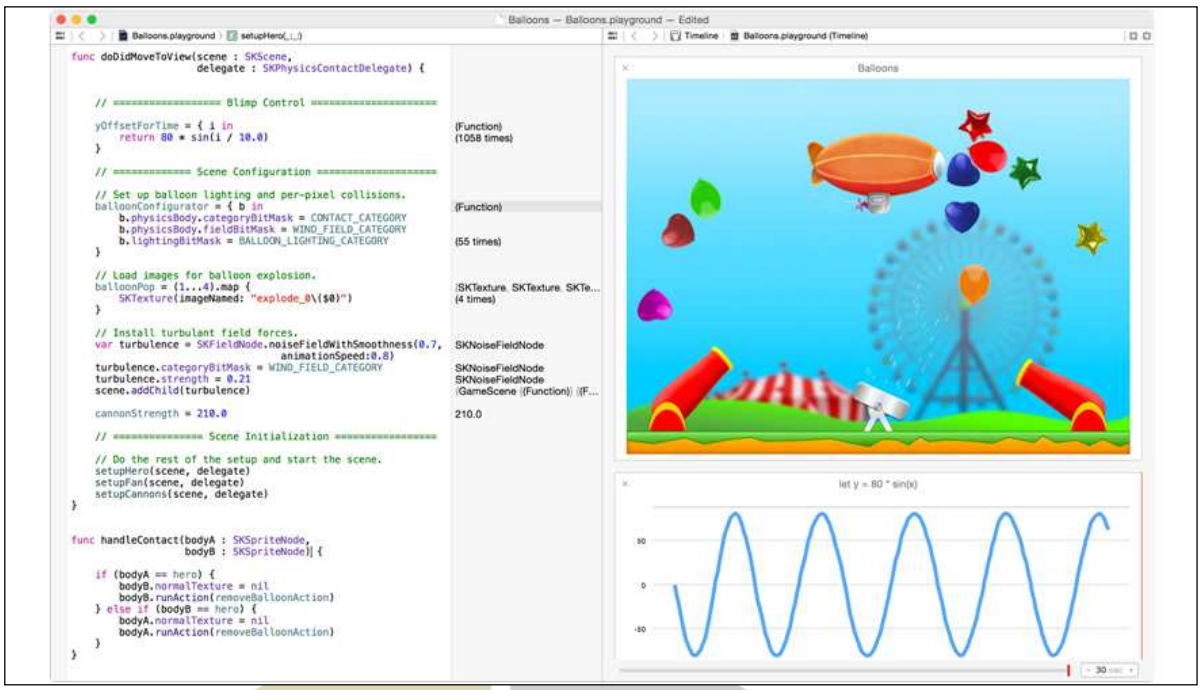


Fonte: Adaptada de Mariani (2010)

A linguagem Swift

Começou a ser desenvolvida no início de 2010 por Chris Lattner, *Swift* passou a ser a nova linguagem de programação da *Apple* para criação de aplicativos para os sistemas operacionais dos dispositivos móveis, computadores e relógios inteligentes. *Swift* é uma linguagem *open source*, ou seja possui o código fonte aberto, é compatível com a antiga linguagem Objective-C, além de possuir uma nova ferramenta conhecida como *PlayGround,* que permite uma execução com respostas instantânea da linha de código digitada. (Apple, 2015)

Figura 15 - Representação do playground em Swift



Fonte: Adaptada da Apple (2015)

A linguagem PHP

O PHP (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) é uma linguagem de *script open source*, ou seja, de código fonte aberto, muito utilizada e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML (PHP, 2015).

Nesta aplicação a linguagem php será utilizada apenas para intermediar a comunicação com o banco de dados MySql.

Padrões de projetos

Em uma definição mais recente, padrões de projeto não são projetos, como listas encadeadas e tabelas de acesso aleatório, que podem ser codificadas em classes e ser reutilizadas tais como então. Tampouco são projetos complexos, de domínio especifico, para uma aplicação inteira ou subsistemas [...] Padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular (GAMMA, 2000).

Apesar de existirem diversos padrões de projeto, padrões têm sido descritos em diferentes formatos.

***Padrão Model View Controller* (MVC)**

O Padrão MVC é um dos mais usados atualmente. Ele é dividido em três camadas:

a. Model (Modelo): representa os dados da organização e as regras de negócios que governam o acesso e atualizações de dados.

b. View (Visão): acessa os dados da organização através do modelo e especifica como os dados devem ser apresentados. Quando o modelo é alterado, é a visão é quem fica responsável por manter a consistência na sua apresentação.

c. Controller (Controlador) - traduz as interações com a visão em ações a serem realizadas pelo modelo. Com base na interação do usuário e os resultados das ações do modelo, o controlador responde ao selecionar uma visão apropriada.

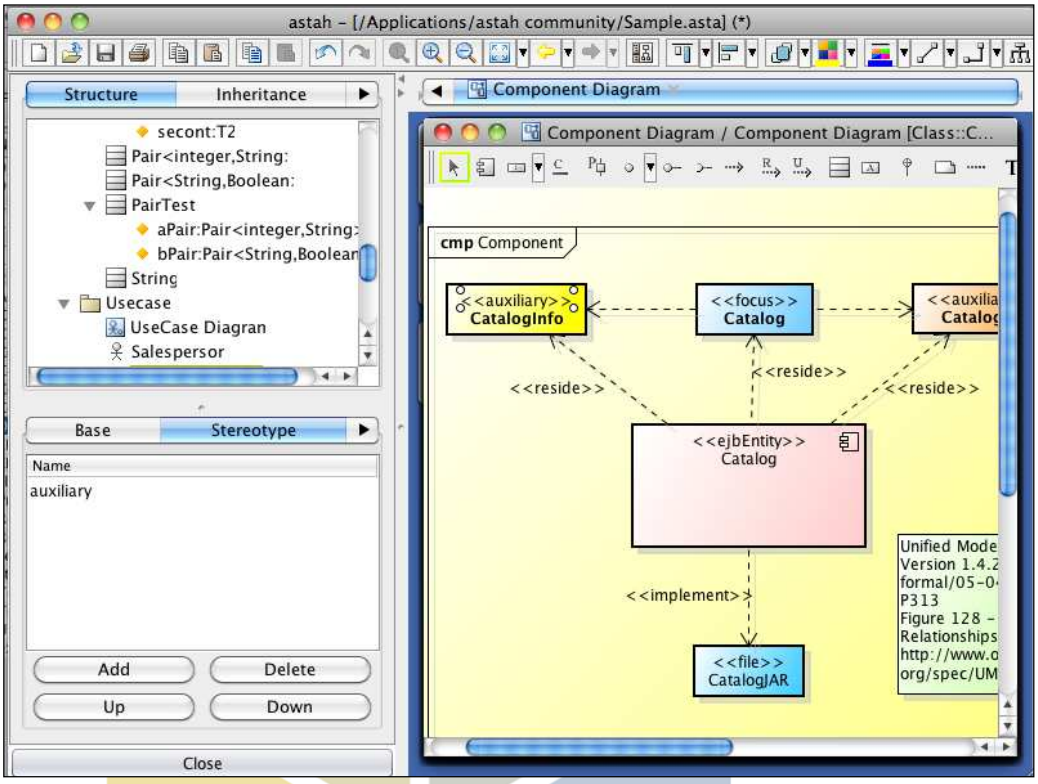
3 FERRAMENTAS

Foram selecionadas algumas ferramentas de desenvolvimento que serão usadas neste projeto com o intuito de melhor desempenho do *software*.

Astah Community

A ferramenta *Astah Community* é *open source* e utilizada para o desenvolvimento da modelagem de *software*. É flexível e extensível e contém vários recursos. Nela é possível desenvolver vários diagramas: diagrama de casos de uso, diagrama de classe, diagrama de sequência, diagrama de estados, diagrama de atividades, diagrama de componentes, diagrama de implantação, diagrama de estrutura composta, diagrama de comunicação e diagrama de pacote, como mostra a figura:

Figura 16 - Visão geral do *Astah Community*

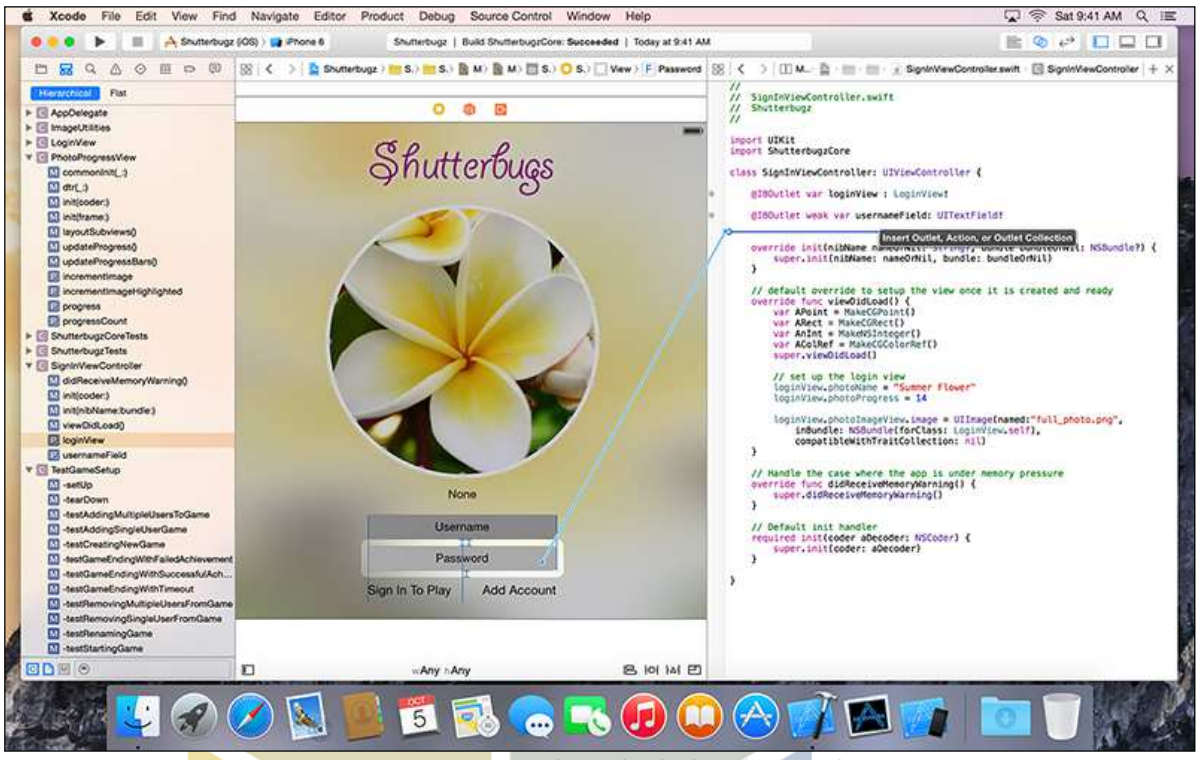


Fonte: Adaptada de (Astah, 2015)

A Ide Xcode

O xcode é a IDE oficial de desenvolvimento no ambiente *Apple*, nela é possível simular execuções do aplicativo até mesmo sem o dispositivo físico.

Figura 17 - Visão geral da ferramenta Xcode

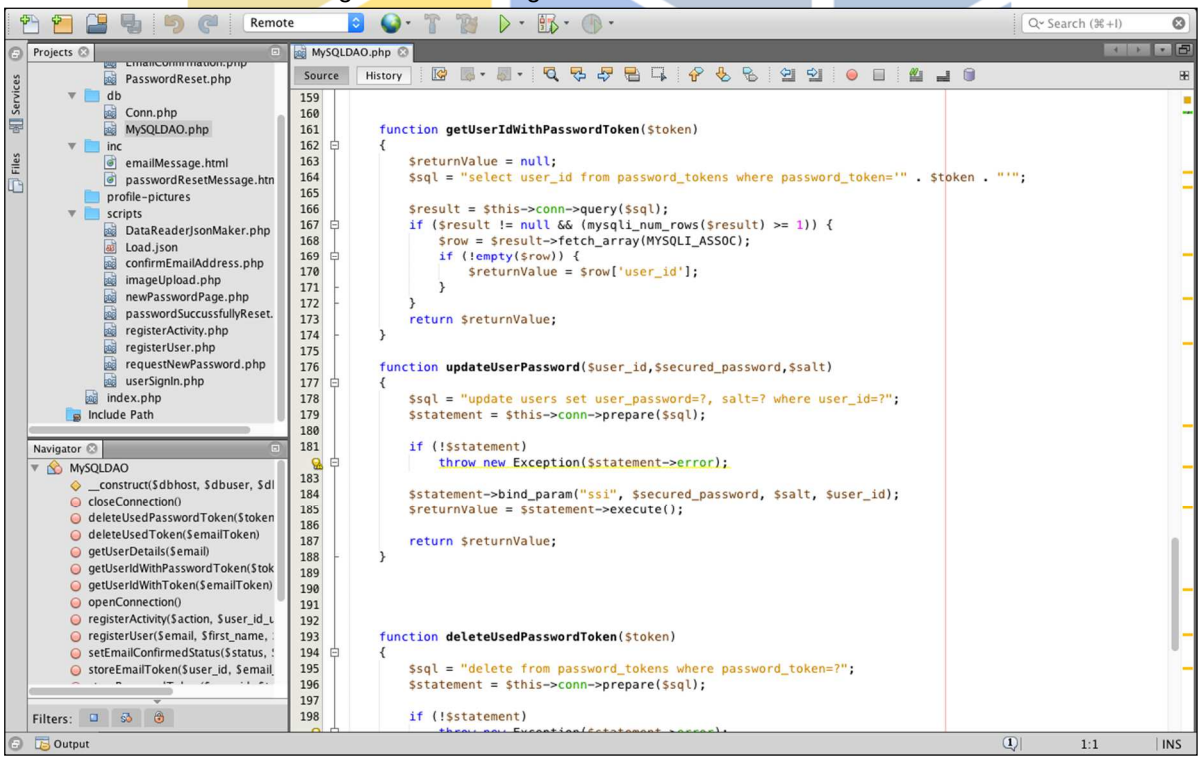


Fonte: Adaptada de (Apple, 2015)

A *Ide NetBeans*

A *IDE NetBeans* permite o desenvolvimento de aplicações utilizando linguagens como Java, PHP, C/C++. Oferece ainda opções de gerenciamento de projeto. O *NetBeans* é compatível com os principais sistemas operacionais, Windows, Linux e Mac OS.

Figura 18 - Visão geral da ferramenta *NetBeans*



Fonte: Adaptada de (netbeans, 2015)

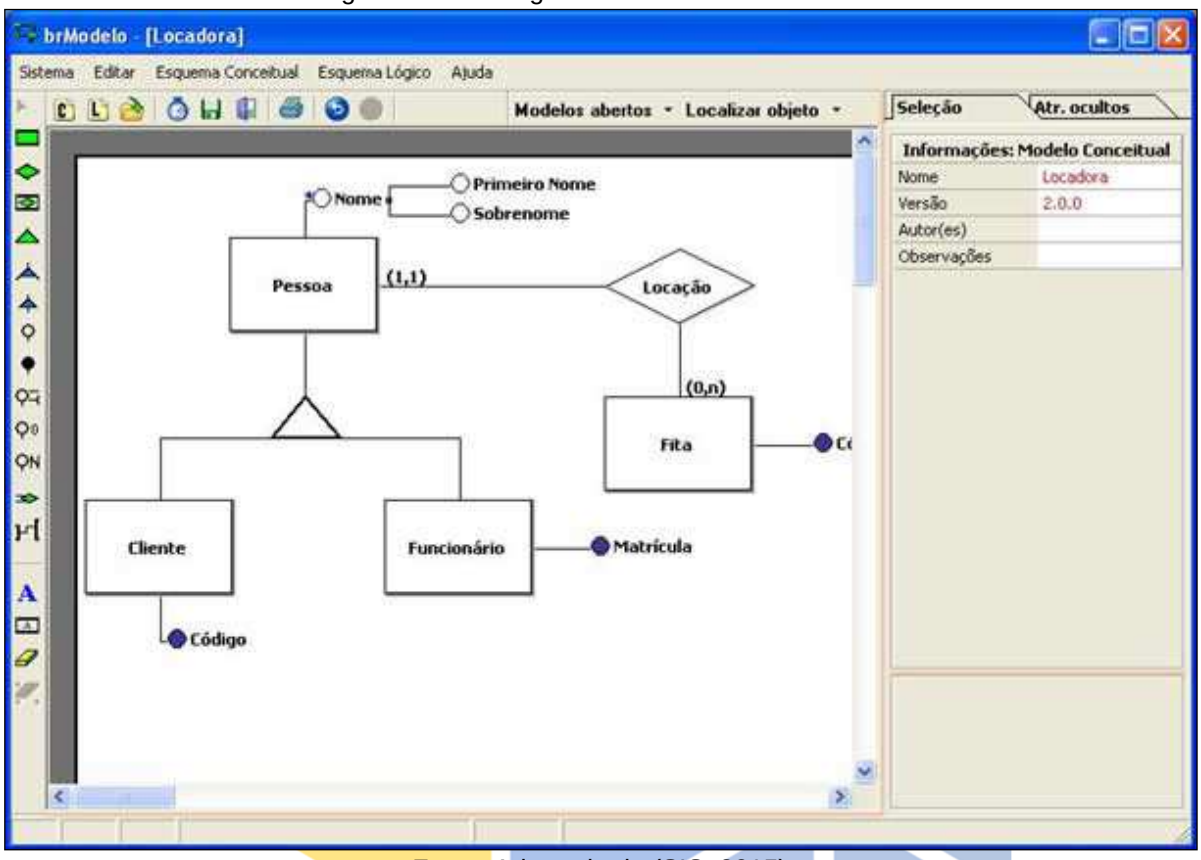
MySQL

O MySQL foi desenvolvido na Suécia em 1996, é um dos sistemas de gerenciamento de banco de dados (SGBD) mais populares. Possui alta compatibilidade com praticamente qualquer sistema operacional, como Linux, FreeBSD, Windows e Mac OS (MYSQL, 2015)

BrModelo

O brModelo é uma ferramenta gratuita desenvolvida especialmente para o ensino de modelagem em banco de dados relacional com base na metodologia do desenvolvedor Carlos A. Heuser. As principais funcionalidades dessa ferramenta são: construção do modelo de entidade e relacionamento, como é apresentado na figura 18, e o mapeamento para o modelo relacional de banco de dados (SIS, 2007)

Figura 19 - Visão geral da ferramenta brModelo



Fonte: Adaptada de (SIS, 2015)

4 ESTUDO DE CASO

Para o processo de desenvolvimento do *Connect* utilizou-se o RUP de forma customizada. O RUP é descrito na seção 1. Este capítulo descreve as algumas das fases do processo de desenvolvimento do sistema, são elas: concepção, elaboração e construção.

Concepção

Nesta fase realizou-se o levantamento de dados e análise dos requisitos.

Levantamento de dados e análise de requisitos

O levantamento dos dados e análise de requisitos foram feitos de forma individual, apenas observando a dificuldade que os vigilantes enfrentam ao executar essa atividade.

Necessidade do negócio

Algumas empresas do setor de segurança privada enfrentam dificuldades no sistema de gerenciamento e controle de acesso, para ter um bom acompanhamento e controle desses acionamentos, como por exemplo, o acionamento dos portões para entrada e saída de carga, quem foi o vigilante que abriu, e quando exatamente foi acionado. Com isso, essas empresas sentem falta de um sistema automatizado que proporcione, à administração e aos funcionários, a comodidade, segurança e confiabilidade.

Visão geral do sistema proposto

Serão apresentadas nesta subseção algumas informações relacionadas ao sistema.

Objetivos do sistema

Desenvolver um sistema que seja capaz de:

- Acesso através de autenticação com usuário e senha;
- Armazenar dados cadastrais dos usuários;
- Arquivar atividades realizadas;
- Fazer upload de imagem; e,
- Pesquisar por usuário ou atividade.

Descrição do escopo do projeto

Um *software* personalizado para o sistema de controle e acionamento precisa ser desenvolvido para que o setor de segurança privada possa controlar dados cadastrais de funcionários, bem como as atividades realizadas por eles, emitir relatórios, e com isso ter um melhor controle no sistema de monitoramento.

Impactos esperados pelo projeto

Melhor acompanhamento das atividades realizadas pelos profissionais da segurança privada, maior confiabilidade, agilidade e segurança.

Requisitos do sistema

Nesta seção apresentam-se os requisitos funcionais e não funcionais do sistema.

Requisitos funcionais

A tabela apresenta os requisitos funcionais do sistema *CONNECT*.

Tabela 1 – Requisitos Funcionais

Código	Requisito
RF01	Possuir dois tipos de usuário: administrador e outro comum
RF02	Permitir a liberação do sistema após a autenticação do usuário
RF03	Para usuário administrador permitir o gerenciamento de usuários
RF04	Gerar relatório de atividades
RF05	Fazer upload de imagem de perfil
RF06	Controlar acionamentos

Requisitos não funcionais

A tabela 2 apresenta os requisitos não funcionais do sistema *CONNECT*.

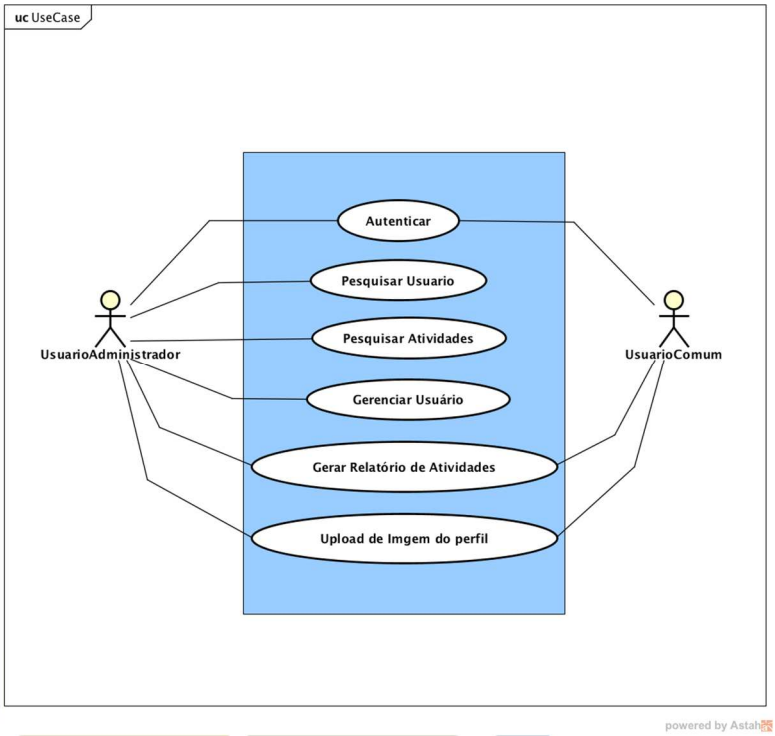
Tabela 2 – Requisitos Não Funcionais

Código	Requisito
RNF01	Compatível apenas com sistema operacional iOS
RNF02	MySQL – SGBD utilizado
RNF03	Níveis de acesso
RNF04	Desenvolvido com linguagem *open source swift*
RNF05	É necessário ter acesso a internet para utilizar a aplicação

Relatório de caso de uso

A figura 20 apresenta o diagrama de caso de uso do sistema, contendo todos os casos de uso do sistema e os usuários.

Figura 20 - Diagrama de Caso de Uso do Sistema



Fonte: ASTAH, 2015

Tabela 3 – Autenticar

Nome	**Autenticar**
Descrição	*A tela de autenticação aparecerá para o usuário e ele deverá informar nos campos indicados, o seu nome e senha.*

Tabela 4 – Pesquisar Usuário

Nome	**Pesquisar Usuário**
Descrição	*Após passar pela autenticação, o administrador poderá realizar pesquisas sobre usuários. Como por exemplo, pesquisar para efetuar algumas alterações.*

Tabela 5 – Pesquisar Atividades

Nome	**Pesquisar Atividades**
Descrição	*Após passar pela autenticação, o usuário administrador poderá realizar pesquisas sobre atividades realizadas. Como por exemplo, saber quem acionou os portões de entrada de carga e descarga, e quando exatamente esses portões foram acionados.*

Tabela 6 – Gerenciar Usuário

Nome	**Gerenciar Usuário**
Descrição	*O administrador do sistema devidamente autenticado poderá cadastrar e excluir usuários.*

Tabela 7 – Gerar Relatório

Nome	**Gerar Relatório**
Descrição	*O usuário do sistema devidamente autenticado poderá gerar relatorios de acordo com o tipo selecionado.*

Tabela 8 – Upload de Imagem de Perfil

Nome	**Upload de Publicação**
Descrição	*O usuário poderá realizar o upload da imagem de perfil.*

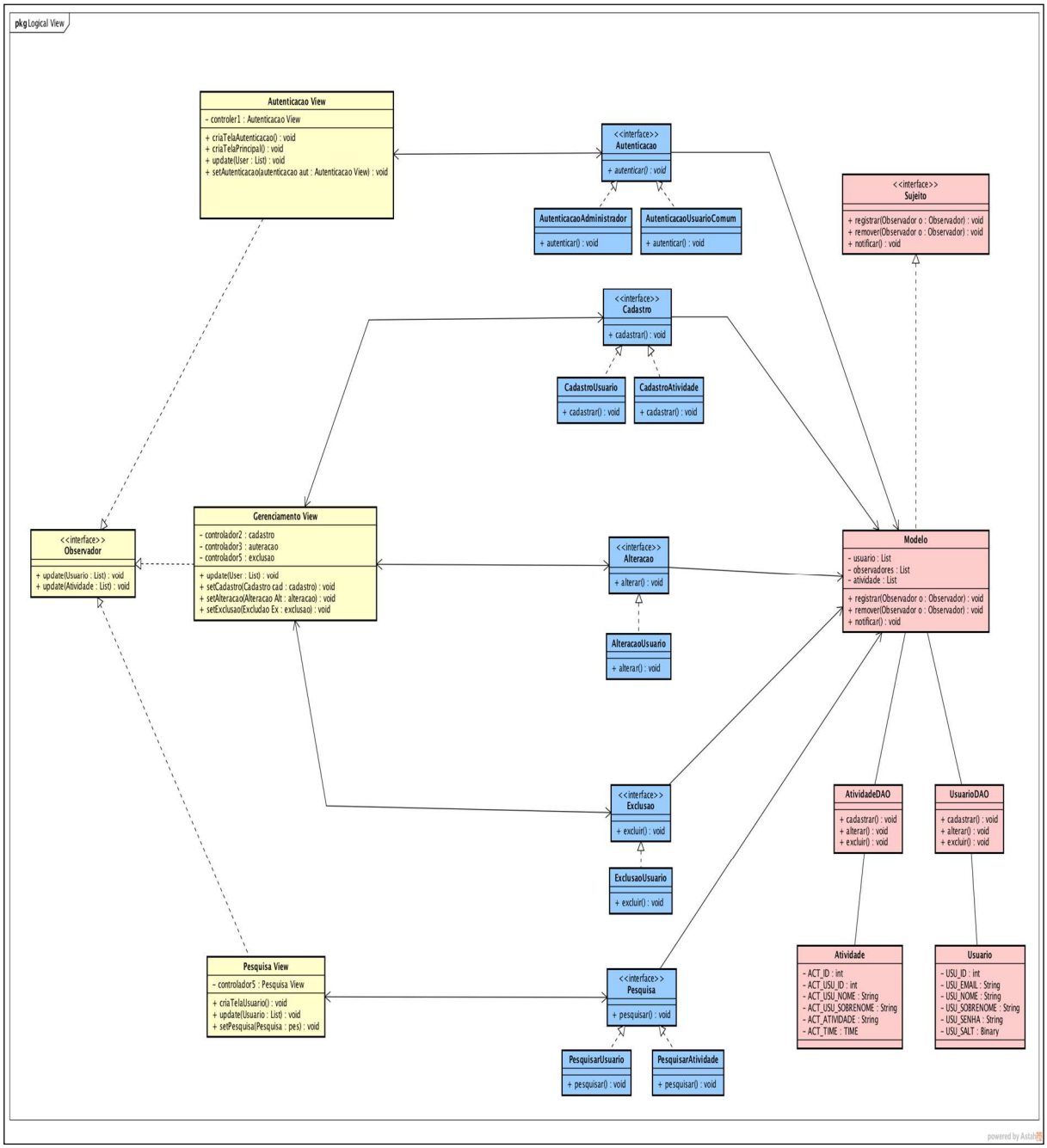
Elaboração

Esta seção descreve a segunda fase do processo de desenvolvimento do *CONNECT*. Nesta fase foram realizados os diagramas de classe, de sequência, de atividade, de estado, de componente e de utilização.

Diagrama de classes

A figura 21 apresenta o diagrama de classes do sistema. Neste diagrama utilizou- se o padrão de projeto: o Padrão MVC – *Model View Controller*. A parte amarela representa a *View*, a parte azul representa o *Controller* e a parte rosa representa o *Model*.

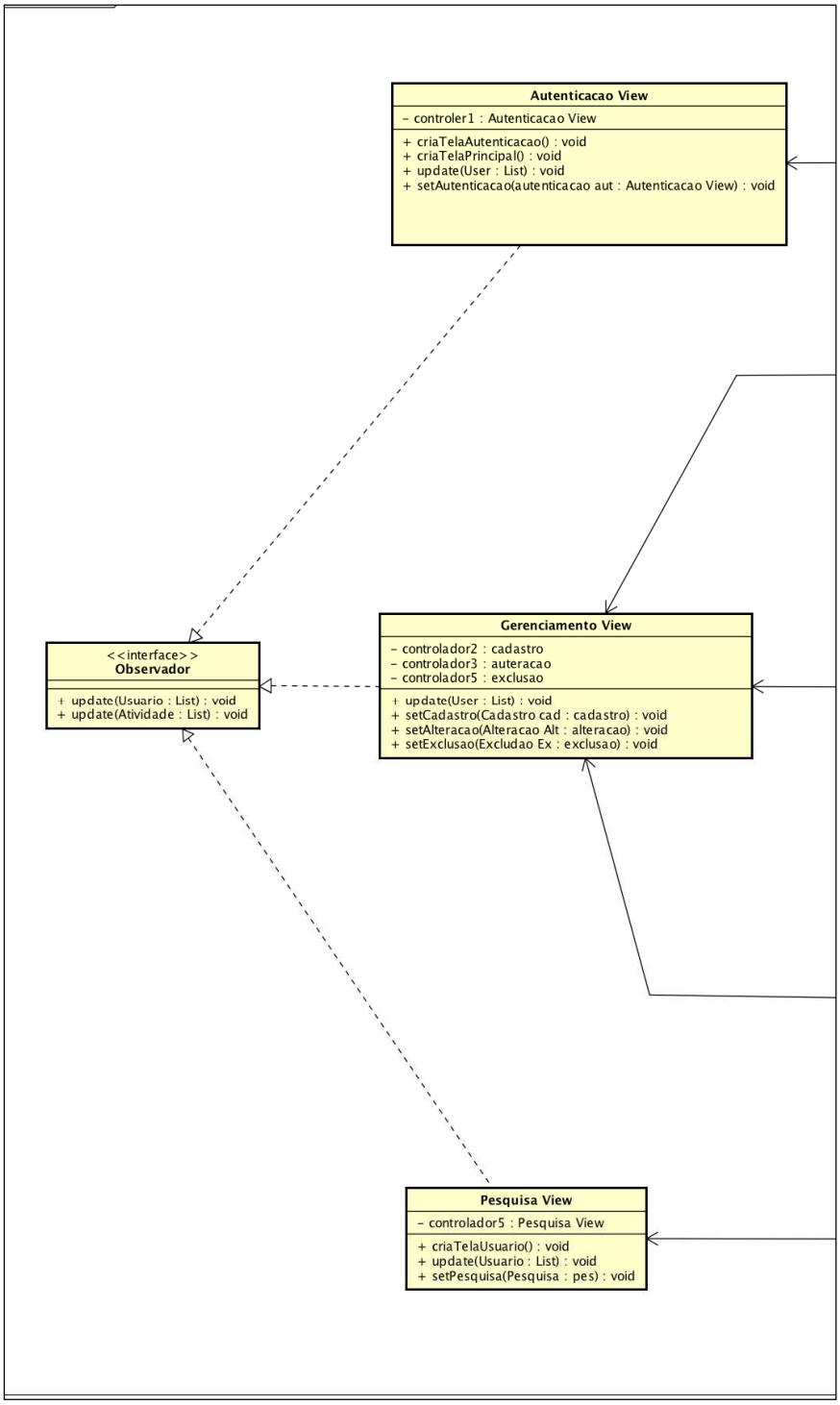
Figura 21 - Diagrama de Classes do Sistema – MVC.



Fonte: ASTAH, 2015

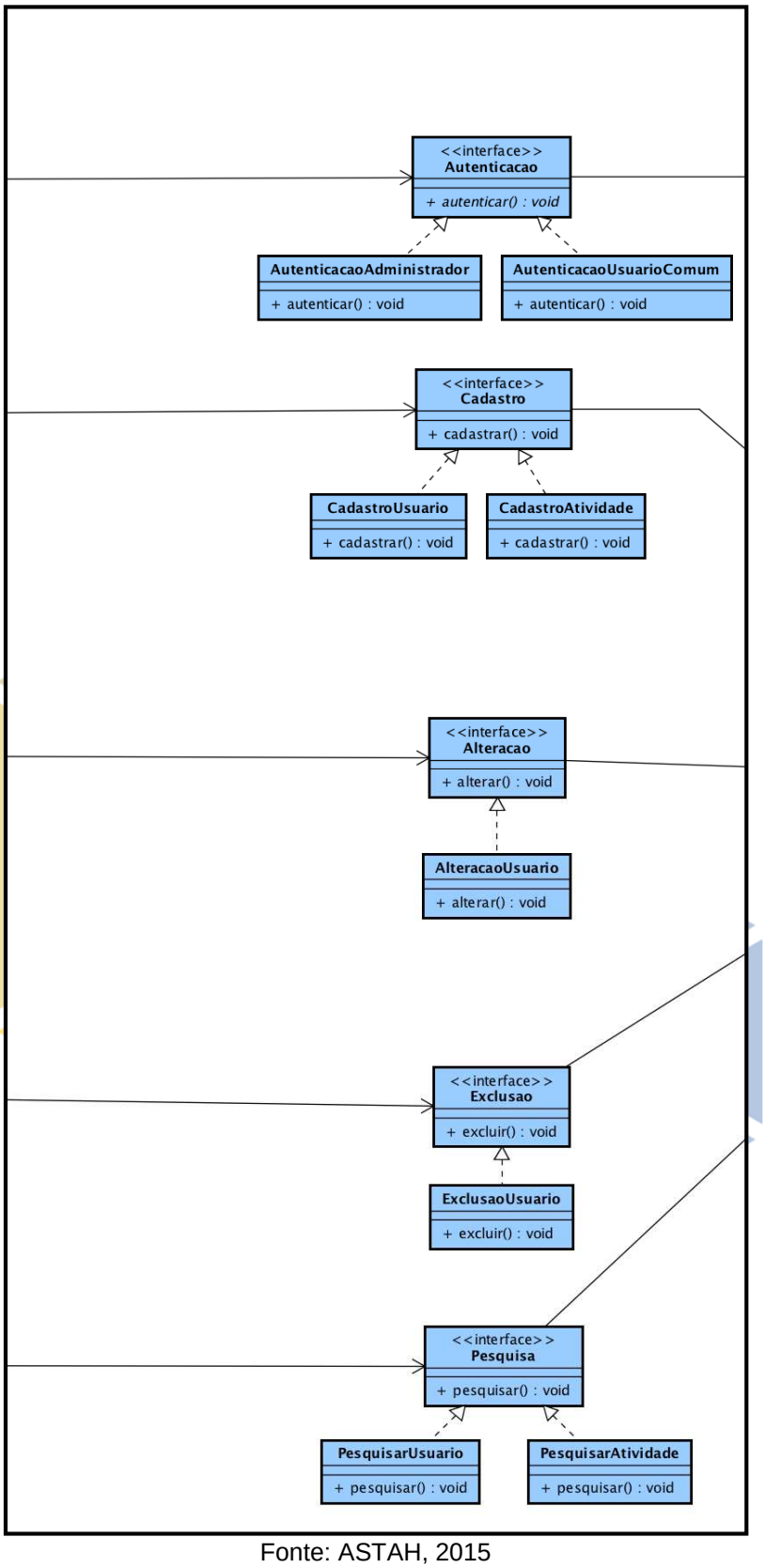
O diagrama foi dividido em três partes para melhor visualização do diagrama, como mostra as figuras 22 – representando a *view*, 23 – representando os *controllers* e 24 – representando o *model*.

Figura 22 - Diagrama – *View.*



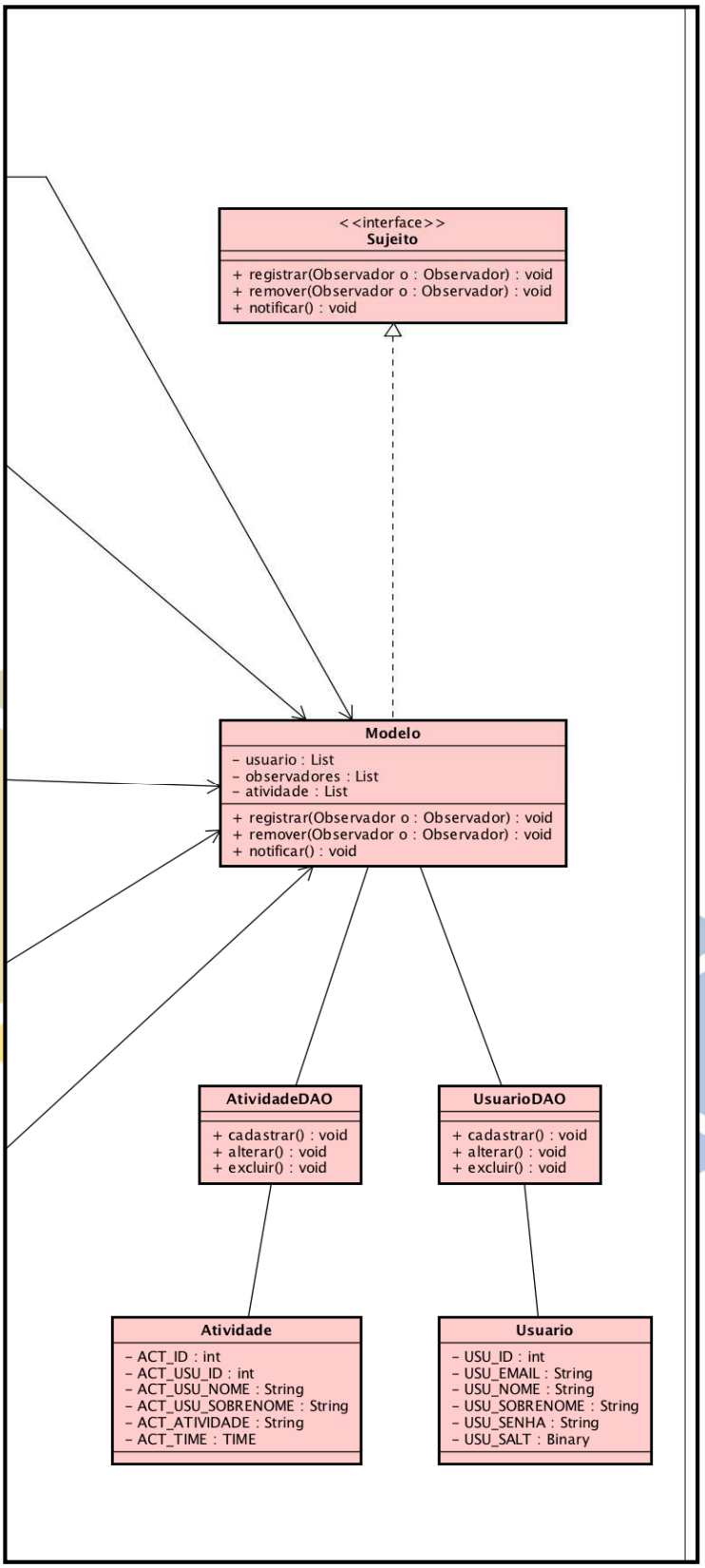
Fonte: ASTAH, 2015

Figura 23 - Diagrama – *Controllers.*



Fonte: ASTAH, 2015

Figura 24 - Diagrama – *Model.*



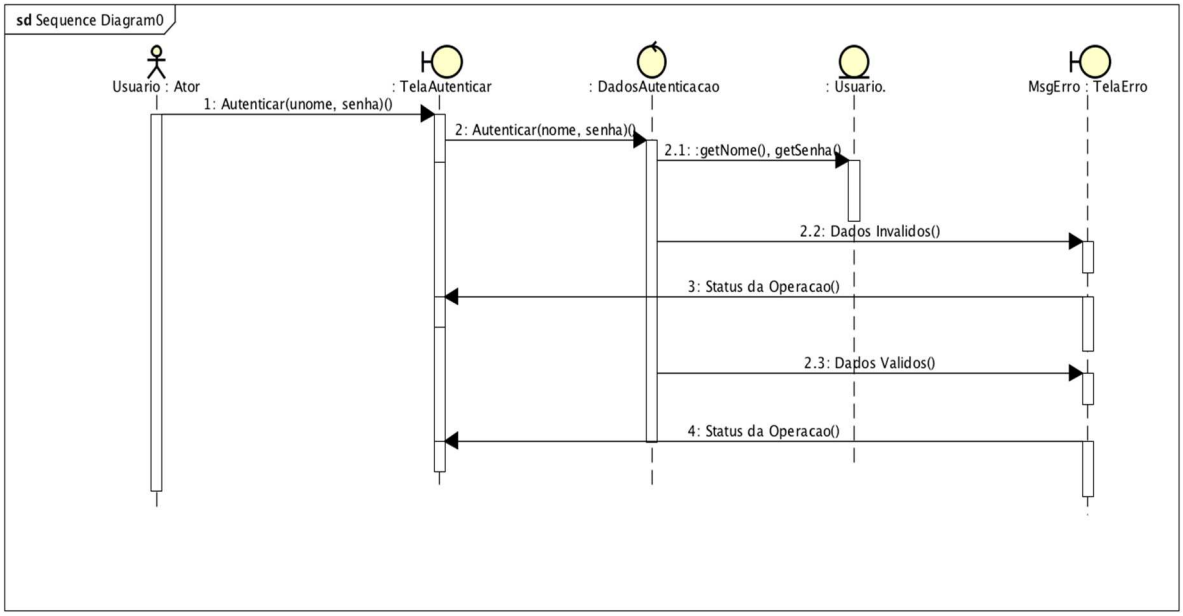
Fonte: ASTAH, 2015

Diagrama de sequência

A figura 25 representa a autenticação de um usuário no sistema. O usuário informa o devido nome e senha na tela de autenticação e em seguida os dados serão conferidos, se os dados estiverem incorretos, aparecerá uma mensagem informando o erro, caso contrário, o usuário terá acesso ao aplicativo, pois ele

estará logado.

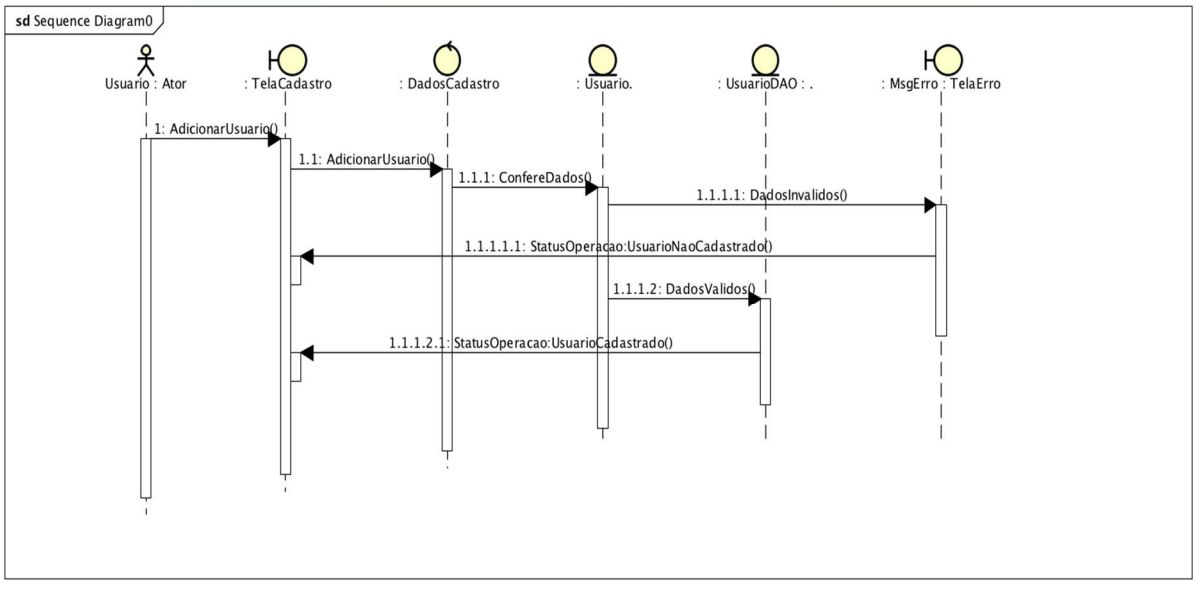
Figura 25 - Diagrama de Sequência - Autenticação.



Fonte: ASTAH, 2015

A figura 26 representa o cadastro de um usuário feito pelo administrador. O usuário informa os dados na tela de cadastro, esses dados serão enviados e conferidos na classe usuário, se estes dados estiverem incorretos, será informada a mensagem avisando o erro, caso contrário, será chamado o método da classe UsuarioDao que irá adicionar o aluno no banco de dados do sistema.

Figura 26 - Diagrama de Sequência - Cadastro de Usuário.

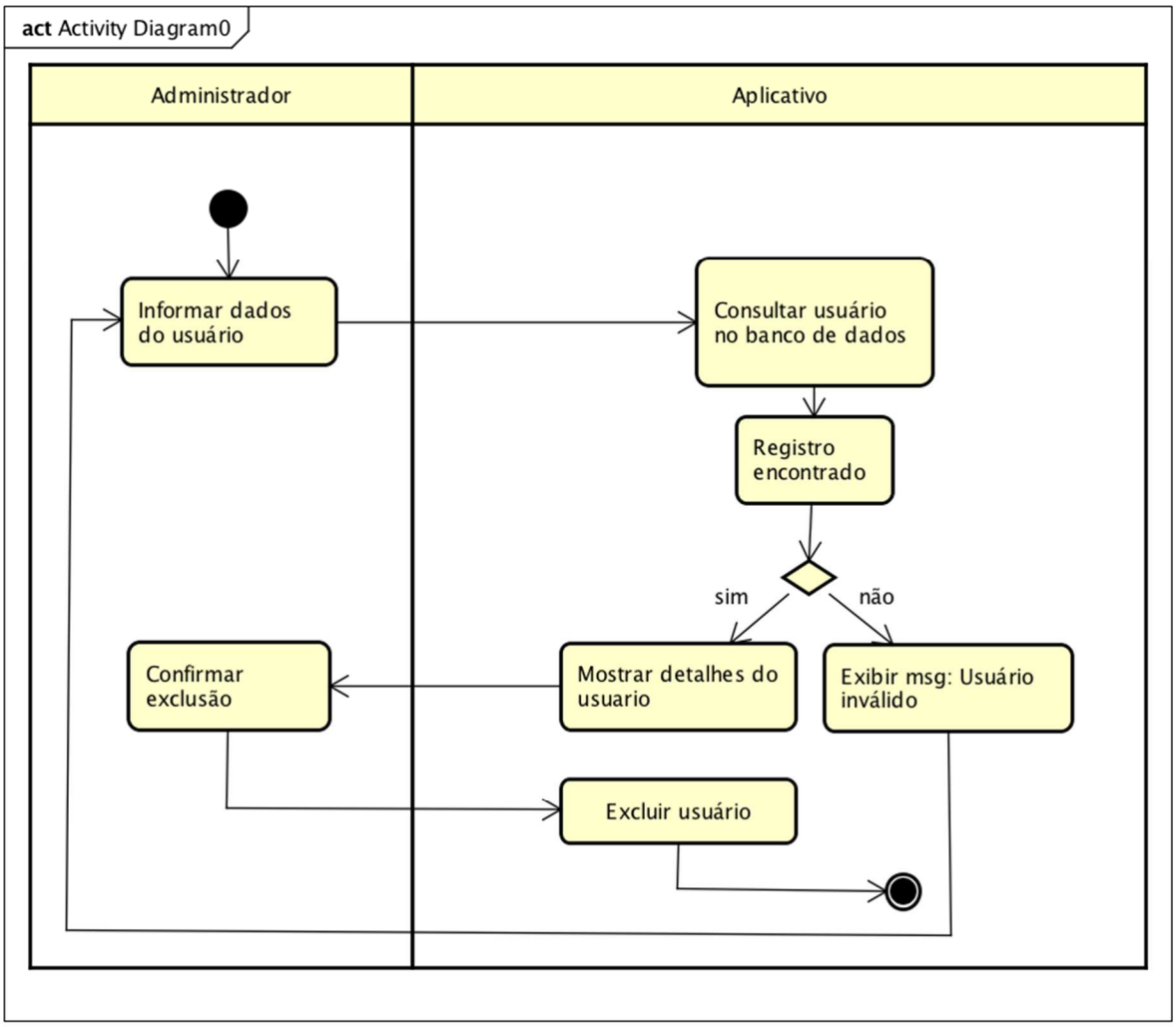


Fonte: ASTAH, 2015

Diagrama de atividades

A figura 27 descreve a atividade realizada pelo usuário administrador ao efetuar uma exclusão relacionada ao usuário. O administrador informa os dados do usuário, em seguida o sistema consulta esse usuário. Se o registro for encontrado o administrador visualiza os dados que o sistema irá recuperar confirmando assim a exclusão, caso o registro não exista, o sistema informará uma mensagem.

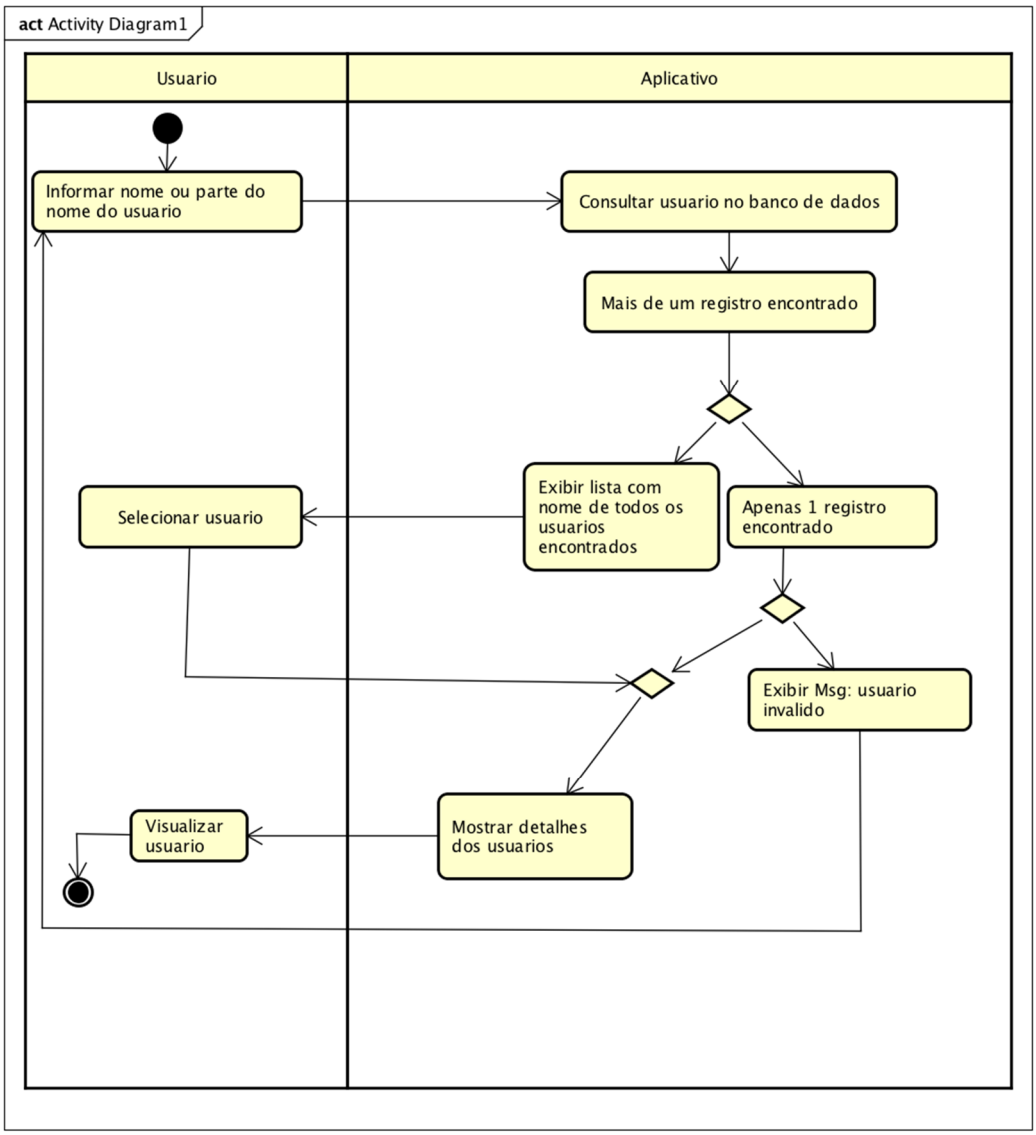
Figura 27 - Diagrama de Atividade de Exclusão de Usuário.



Fonte: ASTAH, 2015

A figura 28 descreve a atividade realizada pelo administrador ao efetuar uma pesquisa relacionada ao usuário. O administrador informa o nome do usuário a ser consultado na tela de pesquisa e em seguida o sistema busca este usuário no banco de dados. Se retornar mais de um registro, o sistema informará uma lista contendo estes registros, caso contrário, se não for encontrado nenhum registro, o sistema informará uma mensagem dizendo que o usuário que estar sendo pesquisado é invalido no sistema, ou seja, nenhum cadastro no banco de dados com o nome informado.

Figura 28 - Diagrama de Atividade de Pesquisa de Usuário.

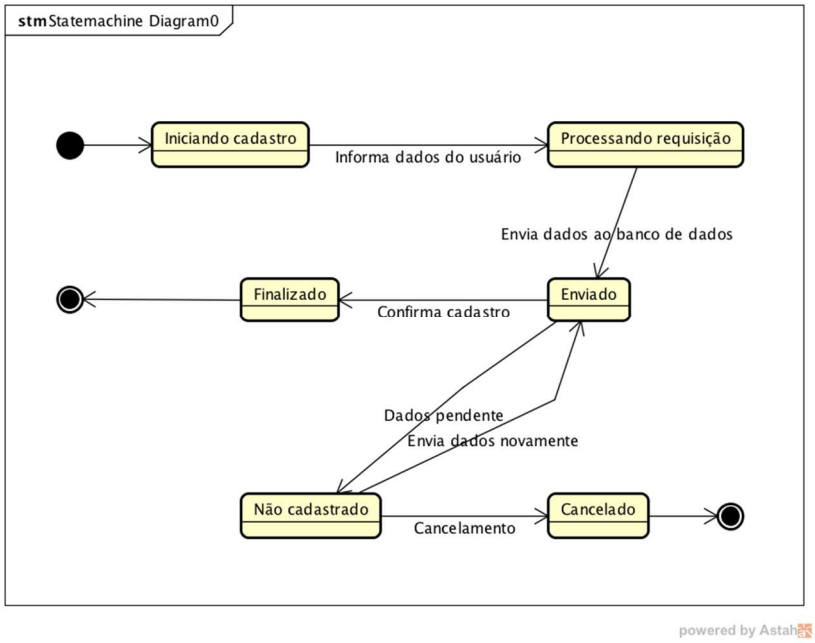


Fonte: ASTAH, 2015

Diagrama de estado

A figura 29 representa o diagrama de estado do cadastro de usuário. O cadastro será iniciado a partir da informação dos dados do usuário a ser cadastro, o objeto passará para o estado de processando requisição e em seguida estes dados serão enviados ao banco de dados, se os dados estiverem corretos, o objeto passa para o estado de finalizado, caso contrário, ele passa para estado de não cadastrado, assim ele pode voltar ao estado enviado, enviando os dados novamente, ou passar para o estado cancelado.

Figura 29 - Diagrama de Estado de Cadastro de Usuário.

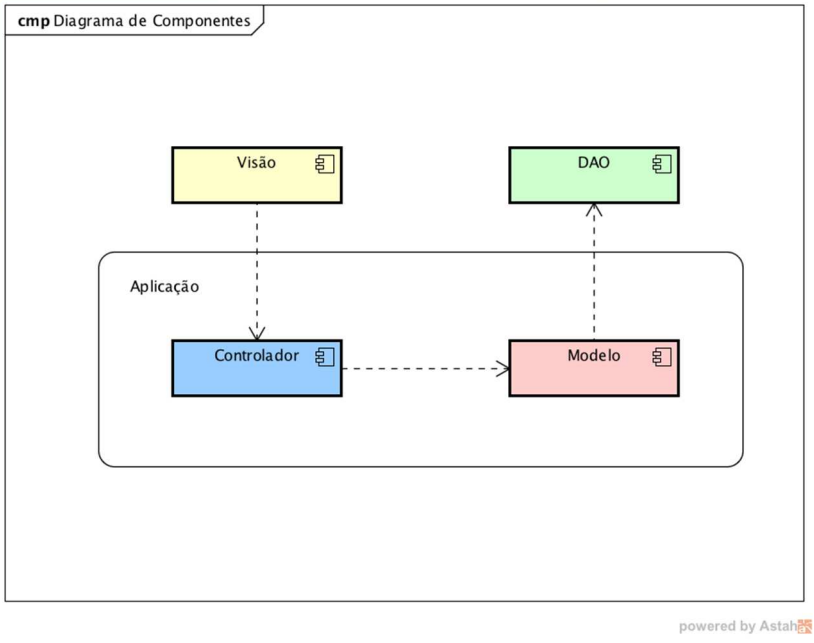


Fonte: ASTAH, 2015

Diagrama de componente

O diagrama de componentes deste projeto é composto pelas três partes do MVC – Modelo, Visão e Controladores; e o DAO, conforme mostrado na figura 30.

Figura 30 - Diagrama de Componente Geral.



Fonte: ASTAH, 2015

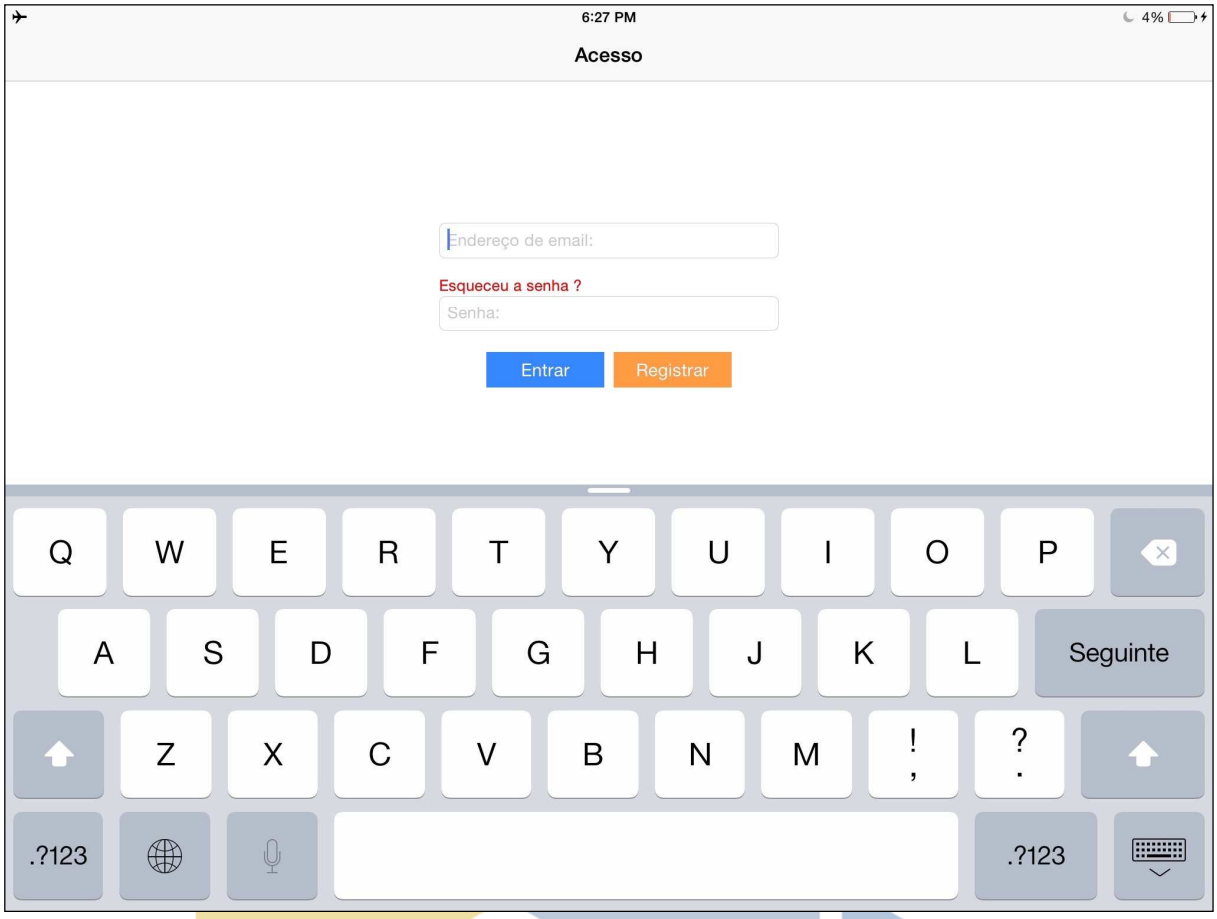
Construção

Será apresentada nesta seção a fase de construção. Nesta fase foram desenvolvidos os protótipos de telas e o projeto de banco de dados.

Protótipo de tela

A figura a seguir representa uma das principais interfaces do *software* com o usuário.

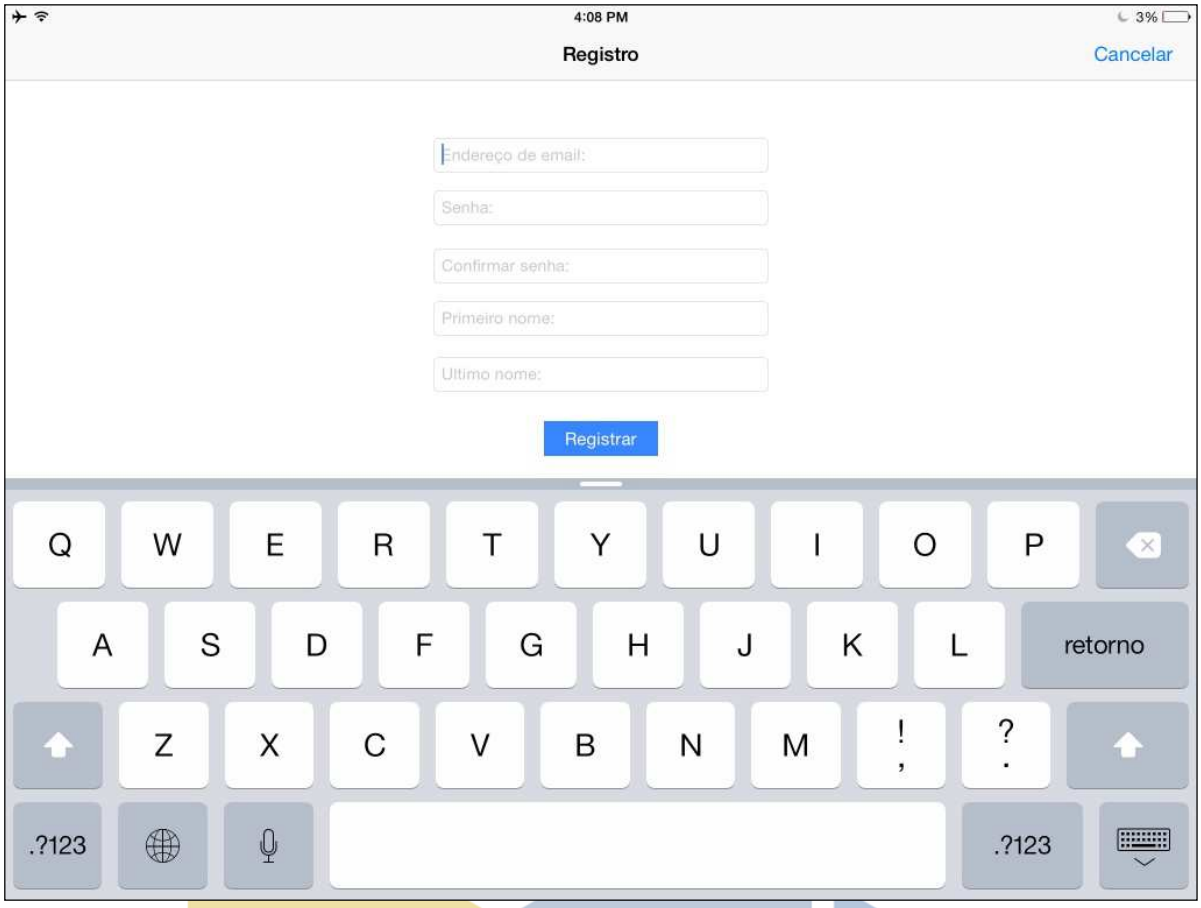
Figura 31 - Tela de Autenticação.



Fonte: Próprio autor

A tela de autenticação possui dois campos onde o usuário preenche com o seu login e senha respectivamente, também se encontra na tela de autenticação as opções de realizar um novo registro, e recuperar a senha, após os dados serem validados o usuário é redirecionado para a tela principal do sistema.

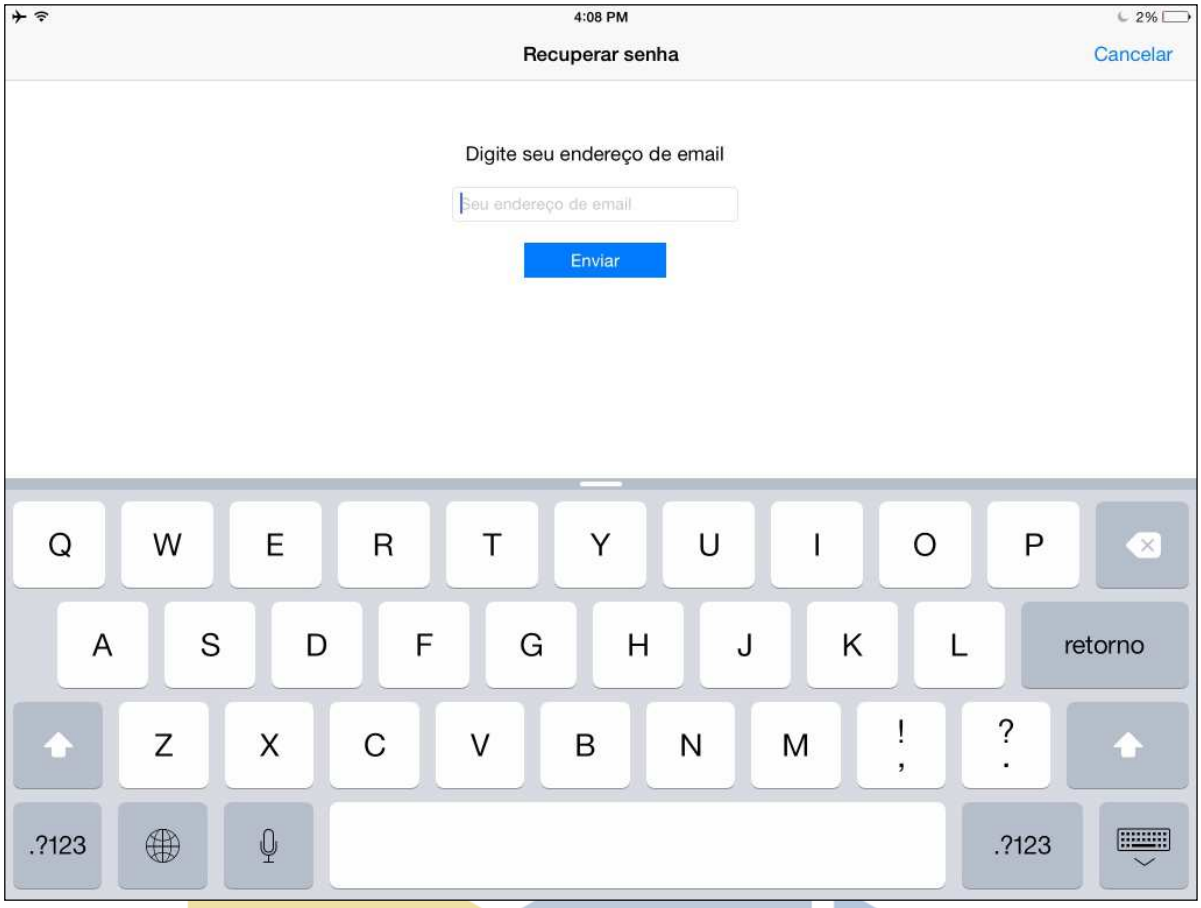
Figura 32 - Tela de Registro.



Fonte: Próprio autor

A tela de registro possui cinco campos, sendo eles respectivamente, endereço de e-mail, senha, confirmação de senha, primeiro nome, último nome e um botão registrar, esse primeiro cadastro só pode ser realizado por um usuário administrador por medidas de segurança.

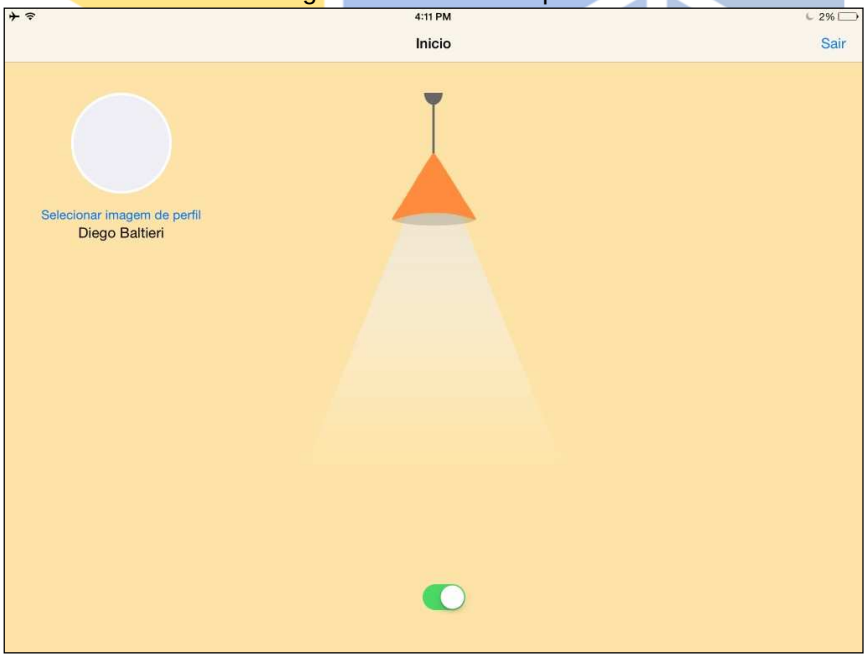
Figura 33 - Tela de Recuperação de Senha.



Fonte: Próprio autor

A tela de recuperação de senha possui apenas um campo, onde o usuário preenche com seu *e-mail* de cadastro, e um botão, para que as informações de recuperação sejam enviadas para o endereço cadastrado, caso o endereço de *e-mail* não seja válido, uma alerta aparecerá na tela.

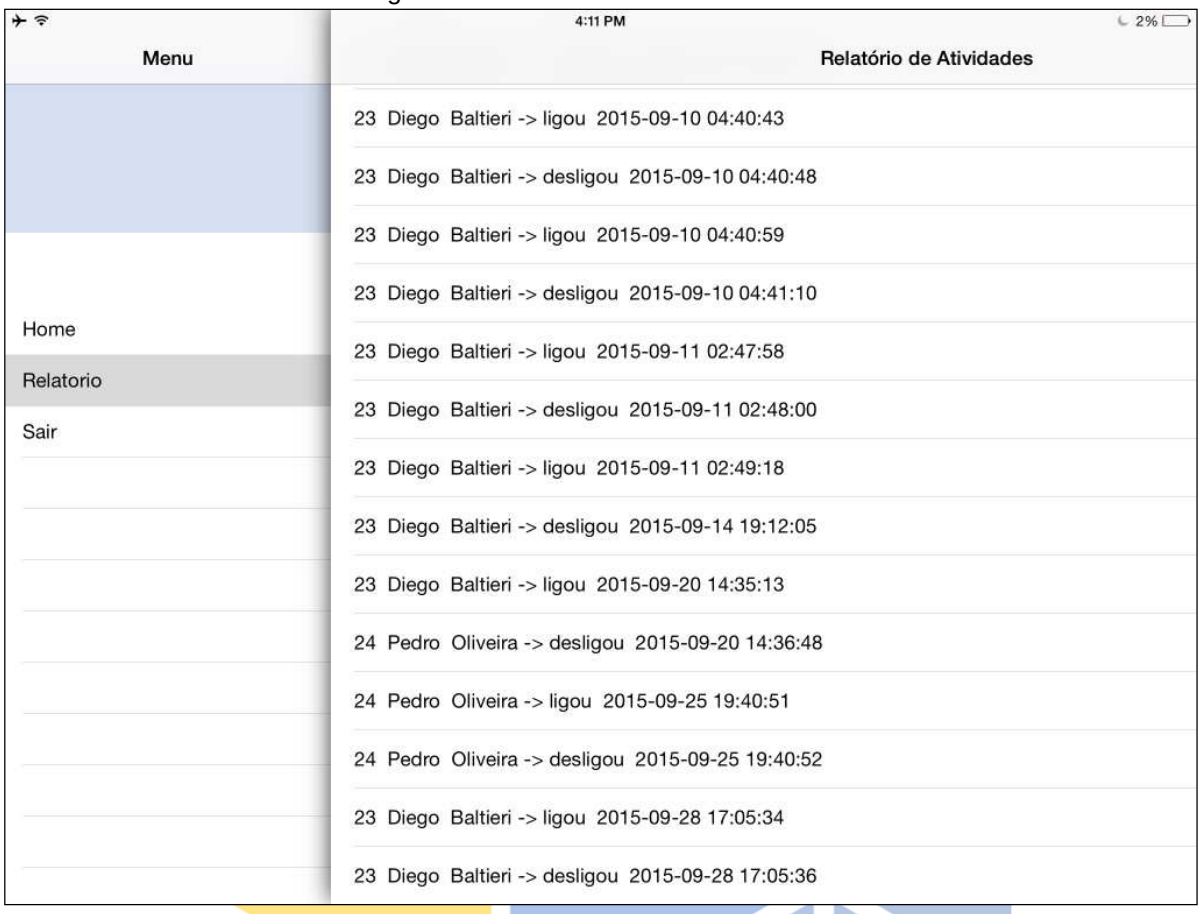
Figura 34 - Tela Principal.



Fonte: Próprio autor

A tela principal possui o nome do funcionário junto com a imagem de perfil que pode ser alterada. Também estão na tela principal todos os botões de controle e acionamento da empresa ou apenas do setor responsável e um botão para finalizar a sessão, no exemplo da figura 34 uma lâmpada foi acionada pelo usuário cujo nome aparece na tela principal, para obter mais detalhes dos acionamentos, basta o usuário arrastar a tela para a direita e selecionar a opção relatório.

Figura 35 - Tela Relatório de Atividades



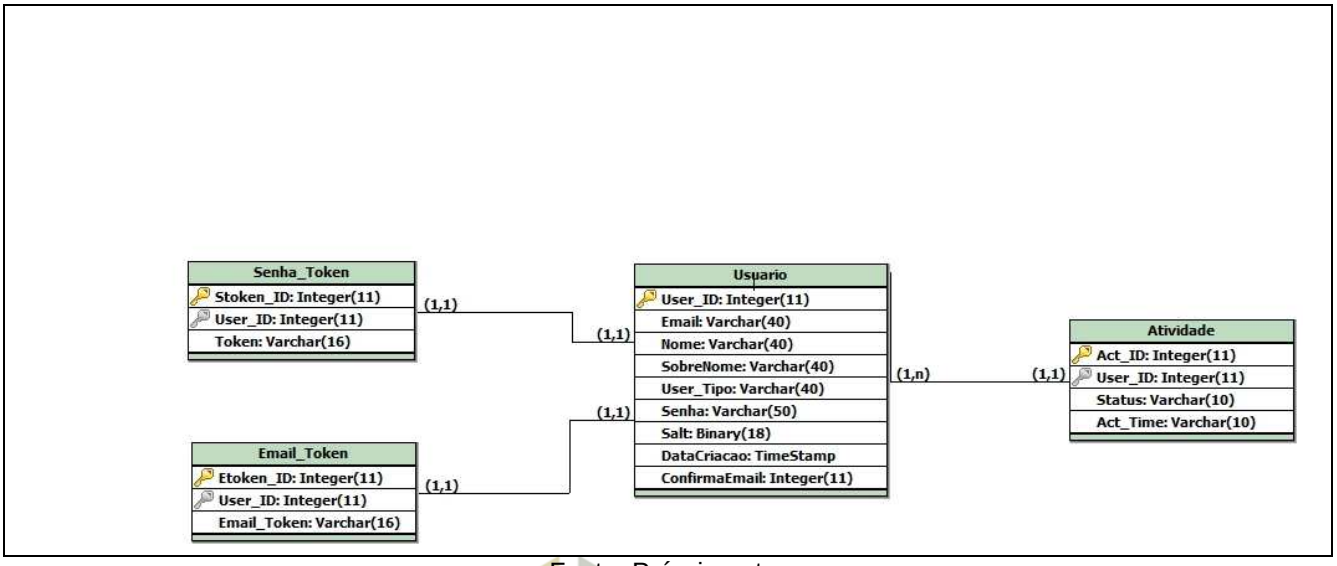
Fonte: Próprio autor

Na tela relatório de atividades o usuário administrador pode realizar consultas de todas as atividades realizadas e por quem elas foram executadas, de forma remotamente, ou seja essa pesquisa pode ser feita até mesmo fora da empresa e em outro dispositivo com a aplicação *CONNECT* instalada.

Projeto de banco de dados - diagramas

Serão apresentadas as fases de modelagem do banco de dados, conforme as figuras 33 e 34. A figura 33 demonstra o modelo conceitual do banco de dados, a sua estrutura; e a figura 34 apresenta o modelo lógico, onde são demonstrados os detalhes de implementação de cada campo das tabelas a partir do modelo conceitual.

Figura 36 - Modelo Lógico do Aplicativo.



Fonte: Próprio autor

Dicionário de dados

Tabela 9 – Tabela Usuário

Nome	**Usuário**		
Descrição	**Tabela que armazena os dados dos usuários**		
Campo	*Constraint*	Tipo	Descrição
User_ID	*PRIMARY KEY*	*INTEGER(11)*	Identificação
Email	*NOT NULL*	*VARCHAR(40)*	Email do usuário
Nome	*NOT NULL*	*VARCHAR(40)*	Primeiro nome
SobreNome	*NOT NULL*	*VARCHAR(40)*	Sobrenome
UserTipo	*NOT NULL*	*VARCHAR(40)*	Tipo de usuário, pode ser comum ou administrador
Senha	*NOT NULL*	*VARCHAR(50)*	Senha
Salt	*NOT NULL*	*BINARY(18)*	Valor binário gerado para aumentar a segurança dos dados
DataCriacao	*NOT NULL*	*TIMESTAMP*	Data da criação
ConfirmacaoEmail	*NOT NULL*	*INTEGER(11)*	Confirma o cadastro do usuário

Fonte: Próprio autor

Tabela 10 – Tabela Atividade

Nome	**Atividade**		
Descrição	**Tabela que armazena as atividades realizadas**		
Campo	*Constraint*	Tipo	Descrição
Act_ID	*PRIMARY KEY*	*INTEGER(11)*	Identificação da atividade
User_ID	*FOREIGNKEY*	*INTEGER(11)*	Identificação do usuário
Status	*NOT NULL*	*VARCHAR(10)*	Status da atividade
Act_time	*NOT NULL*	*VARCHAR(10)*	Momento exato da realização da atividade

Fonte: Próprio autor

Tabela 11 – Tabela Email Token

Nome		***Email Token***	
Descrição		**Tabela que armazena a confirmação do cadastro de usuário**	
Campo	*Constraint*	Tipo	Descrição
Etoken_ID	*PRIMARY KEY*	*INTEGER(11)*	Identificação do token
User_ID	*FOREIGNKEY*	*INTEGER(11)*	Identificação do usuário
Email_Token	*NOT NULL*	*VARCHAR(10)*	Token do email

Fonte: Próprio autor

Tabela 12 – Senha Token

Nome	**Senha Token**		
Descrição	**Tabela que armazena a conversão hexadecimal da senha**		
Campo	Constraint	Tipo	Descrição
Stoken_ID	*PRIMARY KEY*	*INTEGER(11)*	Identificação da token
User_ID	*FOREIGNKEY*	*INTEGER(11)*	Identificação do usuário
Token	*NOT NULL*	*VARCHAR(10)*	Token da senha

Fonte: Próprio autor

Desenvolvimento do sistema

Nesta fase foi realizada a implementação do *CONNECT*. Com detalhes da estrutura do padrão utilizado, o *Model View Controller (*MVC).

Classes da aplicação

As principais classe do projeto são: Classe Principal ou *MainPageViewController*, Classe Usuario ou Registro*ViewController*, Classe Atividades e a Classe *MySQLDAO.* Porém apenas as que se relacionam de forma direta com a classe Usuario serão apresentadas.

Classe Principal

A figura 37 ilustra parte da classe principal

Figura 37 Parte da classe principal

```
//
//  MainPageViewController.swift
//  Connect
//
//  Created by Diego Baltieri on 8/24/15.
//  Copyright (c) 2015 Arcalus Publicidade. All rights reserved.
//

import UIKit

class MainPageViewController: UIViewController, UIImagePickerControllerDelegate,
    UINavigationControllerDelegate {
    @IBOutlet weak var Luz: UIImageView!
    @IBOutlet weak var Interruptor: UISwitch!
    @IBOutlet weak var userFullNameLabel: UILabel!
    @IBOutlet weak var profilePhotoImageView: UIImageView!
    //@IBOutlet weak var userEmailLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()

        // Save da state of Switch

        var padrao = NSUserDefaults.standardUserDefaults()
        if (padrao.objectForKey("PosicaoDoAcionador") != nil){
            Interruptor.on = padrao.boolForKey("PosicaoDoAcionador")
        if (Interruptor.on == true){
            self.Luz.image = UIImage(named: "Light")
        }else{
            self.Luz.image = UIImage(named: "")

                }

        }
        // Do any additional setup after loading the view.



        //Make the profilePhotoImageView Circular

        profilePhotoImageView.layer.cornerRadius = profilePhotoImageView.frame.
            size.width / 2
        profilePhotoImageView.clipsToBounds = true


        profilePhotoImageView.layer.borderColor = UIColor.whiteColor().CGColor
        profilePhotoImageView.layer.borderWidth = 3

    }

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)

        //Reanding datas from DataBase and stored on NSUserDefaults
```

Fonte: Próprio autor

Classe atividade

A classe atividade contém seus devidos atributos e os métodos. A figura 38 representa apenas parte da classe.

Figura 38 - Classe Atividade

```
    func generateBoundaryString() -> String {
        //create and return a unique string
        return "Boundary-\(NSUUID().UUIDString)"
    }


func displayAlertMessege(userMessage:String)
    {
        let myAlert = UIAlertController(title:"Atenção", message:userMessage,
            preferredStyle: UIAlertControllerStyle.Alert);

        let okAction = UIAlertAction(title: "OK", style: UIAlertActionStyle.
            Default, handler:nil)

        myAlert.addAction(okAction);

        self.presentViewController(myAlert, animated: true, completion:nil)
    }


    @IBAction func Acionador(sender: AnyObject) {
        var padrao = NSUserDefaults.standardUserDefaults()
        let userId =
            NSUserDefaults.standardUserDefaults().integerForKey("userId")

        if Interruptor.on {
            registerActivity("ligou", users_user_id: userId)
            self.Luz.image = UIImage(named: "Light")
            padrao.setBool(true, forKey: "PosicaoDoAcionador")

        }else{
            registerActivity("desligou", users_user_id: userId)
            self.Luz.image = UIImage(named: "")
            padrao.setBool(false, forKey: "PosicaoDoAcionador")
        }
    }


    func registerActivity(action: String, users_user_id: Int){

        // Send HTTP POST (http request)

        let myUrl = NSURL(string:"http://arcalus.com.br/master/connect/
            SwiftAppAndMySQL/scripts/registerActivity.php");

        let request = NSMutableURLRequest(URL: myUrl!);
        request.HTTPMethod="POST";

        let postString = "userAction=\(action)&userIdUser=\(users_user_id)";

        request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)


        NSURLSession.sharedSession().dataTaskWithRequest(request,
            completionHandler: {
```

Fonte: Próprio autor

Pode se observar em destaque o método registerActivity(), que é um dos métodos principais desta aplicação, ele recebe por parâmetro a posição dos acionadores, ou seja, se estão ligados ou desligados e também a identificação do usuário que acionou, essa informação é enviada ao banco de dados de forma instantânea, esse método é implementado na classe atividade.

Classe usuário

A classe usuário contém todas as instâncias e métodos relacionados aos usuários, seja ele comum ou administrador. A figura 39 representa parte da classe usuário denominada de RegistroViewController.

Figura 39 - Classe Usuário

```
//
//  RegistroViewController.swift
//
//
//  Created by Diego Baltieri on 7/21/15.
//
//

import UIKit


class RegistroViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var userEmailAddressTextField: UITextField!
    @IBOutlet weak var userPasswordTextField: UITextField!
    @IBOutlet weak var userPasswordRepeatTextField: UITextField!
    @IBOutlet weak var userFirstNameTextField: UITextField!
    @IBOutlet weak var userLastNameTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.


    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func BotaoCancelarTapped(sender: AnyObject) {
        self.dismissViewControllerAnimated(true, completion: nil)
    }

    @IBAction func registroButtonTapped(sender: AnyObject) {
        let userEmail = userEmailAddressTextField.text
        let userPassword=userPasswordTextField.text
        let userPasswordRepeat=userPasswordRepeatTextField.text
        let userFirstName=userFirstNameTextField.text
        let userLastName=userLastNameTextField.text

        if(userPassword != userPasswordRepeat)
        {
            //Display alert message
            displayAlertMessege("A senha não confere")
            return
        }

        if(userEmail!.isEmpty || userPassword!.isEmpty || userFirstName!.isEmpty
            || userLastName!.isEmpty)
        {
            //Display an alert messege
            displayAlertMessege("Todos os campos devem ser preenchidos !")
            return
```

Fonte: Próprio autor

5 CONSIDERAÇÕES FINAIS

Um sistema de gerenciamento de controle e acionamento é muito importante para auxiliar profissionais do setor de segurança privada, devido à facilidade de gerenciamento de dados com segurança, confiabilidade e comodidade. Neste trabalho foi possível desenvolver um protótipo do aplicativo capaz de se comunicar com sistemas eletrônicos e realizar o gerenciamento do controle e acionamento para o setor de segurança privada. Foram estudados o referencial teórico para o desenvolvimento da aplicação, todos os requisitos funcionais e não funcionais foram definidos e também o banco de dados relacional foi implementado.

A versão desenvolvida da aplicação possui funções que ainda estão limitadas e serão posteriormente aprimoradas. A principal característica que será implementada nas próximas versões será a autenticação por biometria, já que os novos dispositivos e frameworks da *Apple* oferecem este recurso.

A maior dificuldade encontrada durante o desenvolvimento da aplicação foi a falta de conteúdo da linguagem de programação Swift, e sua constante atualização em um curto espaço de tempo. No instante em que o projeto começou a ser desenvolvido a versão disponível era a *Swift* 1.0, atualmente a linguagem já se encontra em sua versão 2.1

O trabalho pode contribuir de forma significativa com empresas do setor de segurança privada, além de incentivar a comunidade acadêmica a despertar interesses no desenvolvimento de aplicações voltada para a segurança de patrimônios e pessoas.

REFERÊNCIAS

APPLE, 2014 - 2015. ***Model - View - Controller.*** Disponível em < https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html > Acesso em 01/10/2015.

APPLE, 2015. **Site oficial de desenvolvedores**. Disponível em < https://developer.apple.com/swift/> Acesso em 07/10/2015.

ASTAH_COMMUNITY, 2015. **Site oficial Astah Comumunity.** Disponível em <http://astah.change-vision.com/en/product/astah-community.html > Acesso em 12/11/2015.

BROOKSHEAR, J. G. **Ciência da computação:** uma visão abrangente. Porto Alegre: Bookman, 2003.

COUTO, A. **Estatuto da segurança privada**. Disponível em <http://www.recantodasletras.com.br/artigos/3502006> Acessado em 12/07/2015

DAMAS, L. **SQL Structured Query Language**. Rio de Janeiro: LCT, 2005.

DATE**,** C. J. **Introdução a sistemas de bancos de dados.** Rio de Janeiro: Elsevier: 2003

GAMMA, E. H., R. J., R. **Padrões de projeto**. Porto Alegre: Bookman, 2000.

HEUSER, C. A. **Projeto de banco de dados**. Porto Alegre: Sagra, 2000.

IBM, 2001. **Site oficial IBM**. Disponível em <http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf> Acessado em 06/08/2015

IBMa, 1998. **Rational Unified Process.** Disponível em: < http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf > Acessado em: 25/10/2015.

IBMb, 2011. **Site oficial IBM.** Disponível em <http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004100.htm > Acessado em 25 de Junho de 2015.

KRUCHTEN, P. **The Rational Unified Process:** an introduction. São Paulo: Addison Wesley, 2003.

MARIANI, A. C. **O mundo dos atores:** uma perspectiva de introdução à programação orientada a objetos. Disponível em <http://www.inf.ufsc.br/poo/atores/sbie98/sbie98-atores.html> Acesso em 15/06/2015.

MAXPRESS, 2014. **Segurança privada**. Dispinível em <http://www.saopauloexpo.com.br/canal/?noticias/13944/maior+feira+de+seguranca+privada+do+pais,+voltada+a+prevencao+da+vida+e+do+patrimonio/+sp/#.VjbMsYRXZg> Acessado em 03/11/2015

MYSQL, 2015. **Site Oficial MySQL**. Disponível em < http://www.mysql.com/why- mysql/ > Acesso em 10/09/2015.

NETBEANSa, 2015. **Site oficial NetBeans.** Disponível em < http://netbeans.org/index_pt_PT.html> Acesso em 25/09/2015.

PHP, 2015. **Site oficial PHP**. Disponível em <http://php.net/manual/pt_BR/intro-whatis.php> Acesso em 08/11/2015

REZENDE, D. A. **Engenharia de *software* e sistemas de informação**. Rio de Janeiro: Brasport, 2005.

RICARTE, I. L. M., 2001. **Programação orientada a objetos:** uma abordagem com Java. Disponível em: http://www.dca.fee.unicamp.br/cursos/PooJava/Aulas/poojava.pdf. Acesso em 22/10/2015.

SIERRA, Kathy; BATES, Bert. **Use a cabeça! Java**. Rio de Janeiro: AltaBooks, 2010.
SILVA, R. P. **Uml2:** em modelagem orientada a objetos. Florianópolis: Visual Books, 2007.

SIS, 2007. **Site oficial BrModelo.** Disponível em <http://www.sis4.com/brModelo/ > Acesso em 18/08/2015.

SOMMERVILLE, I. **Engenharia de *software***. São Paulo: Pearson, 2007.

UML, 2015. **Site oficial UML.** Disponível em < http://www.uml.org/#UML2.0 > Acesso em 10/11/2015.

Wladimir da Costa
Possui graduação em Análise de Sistemas pela Universidade Metodista de Piracicaba (1992), Pós Graduação Lato Sensu em Analise de Sistemas pela Universidade Metodista de Piracicaba (1995) e Pós Graduação Stricto Sensu em Ciência da Computação pela Universidade Metodista de Piracicaba (2005). Em 2008, inicia a coordenação nos cursos da Fatec Americana. Atualmente é coordenador do curso de Tecnólogo em Análise e Desenvolvimento de Sistemas. Também ministra aulas na Escola de Engenharia de Piracicaba. Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Computação, atuando principalmente nos seguintes temas: Planejamento Estratégico de Tecnologia de Informação, Sistemas de Informação e Redes.
Contato: cwladi@hotmail.com
Fonte: CNPQ – Curriculo Lates