UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

LUCAS DYNCZUKI PALLA

DESENVOLVIMENTO E ESTUDO COMPARATIVO DE APLICAÇÃO EM BANCO DE DADOS RELACIONAL E BANCO DE DADOS ORIENTADO A OBJETOS

Bandeirantes

2013

LUCAS DYNCZUKI PALLA

DESENVOLVIMENTO E ESTUDO COMPARATIVO DE APLICAÇÃO EM BANCO DE DADOS RELACIONAL E BANCO DE DADOS ORIENTADO A OBJETOS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Me Glauco Carlos Silva

Bandeirantes
2013

LUCAS DYNCZUKI PALLA

DESENVOLVIMENTO E ESTUDO COMPARATIVO DE APLICAÇÃO EM BANCO DE DADOS RELACIONAL E BANCO DE DADOS ORIENTADO A OBJETOS

Plano de TCC apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Prof. Me Glauco Carlos Silva
UENP – *Campus* Luiz Meneghel

_____
Prof. Wellington Della Mura
UENP – *Campus* Luiz Meneghel

_____
Prof. Estevan Braz Brandt Costa
UENP – *Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2013

RESUMO

É constante a procura da combinação ideal entre linguagens de desenvolvimento de software e mecanismos de armazenamento de dados, já que alcançando este cobiçado objetivo as empresas poupam esforço e tempo ao desenvolverem suas aplicações. Para que se possa determinar qual a melhor união entre os paradigmas, é necessário consultar uma documentação comparativa. No entanto, a carência deste tipo de comparação é alta. Diante deste cenário, este trabalho possui como objetivo gerar um estudo comparativo entre o modelo relacional e o orientado a objetos. Para que isto seja alcançado, primeiramente foi realizada uma pesquisa bibliográfica acerca de banco de dados e também sobre as métricas de estimativa de software usualmente utilizadas. Posteriormente, foram desenvolvidos dois protótipos de uma agenda eletrônica, que possuem as mesmas funções, que serão empregadas em duas classes de banco de dados díspares, o Banco de Dados Relacional e Banco de Dados Orientado a Objetos. A análise comparativa foi realizada por meio de métricas de estimativas de projeto de software, avaliando os requisitos, tamanho do sistema, o esforço requerido, e o tempo de desenvolvimento das aplicações.

Palavras-chave: Banco de Dados Relacional, Banco de Dados Orientado a Objetos, Orientação a Objetos, COCOMO.

ABSTRACT

It’s constant the search for the ideal combination between software development languages and mechanisms for data storage, since reaching this coveted aims companies save effort and time to develop their applications. To be able to determine the best union between the paradigms, it is necessary to consult a documentation comparative. However, the deficiency of this type of comparison is high. Before this scene, this work has as objective to generate a comparative study between the relational model and the oriented to objects. For this to be reached, first of all was realized a bibliographic search about database and also on the software estimation metric most commonly used across the world. After, it was developed two prototypes of an electronic organizer, which have the same functions, which will be employed in two classes of databases disparate. The Relational Database and Database Oriented to Objects. The comparative analysis was realized through metrics software Project estimates, estimating requirements, system size, the required effort, and the time of application’s development.

Keywords: Relational Database, Database Oriented to Objects, Object Orientation, COCOMO.

LISTA DE FIGURAS

LISTA DE FÓRMULAS

LISTA DE TABELAS

LISTA DE SIGLAS

ANSI	*American National Standards Institute*
APF	Análise de Pontos por Função
BDOO	Banco de Dados Orientado a Objetos
BDR	Banco de Dados Relacional
COCOMO	*Construtive Cost Model*
DBA	*Database Administrator*
DLC	*Data Control Language*
DML	*Data Definition Language*
DML	*Data Manipulation Language*
IDE	*Integrated Development Environment*
ISO	*International Standards Organization*
KLOC	*Kilo Lines of Code*
LOC	*Lines of Code*
MR	Modelo Relacional
ODL	*Object Database Language*
ODMG	*Object Database Management Group*
OLAP	*On-line Analytical Processing*
OO	Orientação a Objetos
OQL	*Object Query Language*
ORM	*Object Relational Mapping*
POO	Programação Orientada a Objetos
SEQUEL	*Structured English Query Language*
SGBD	Sistema Gerenciador de Banco de Dados
SGBDOO	Sistema Gerenciador de Banco de Dados Orientado a Objetos
SGDBR	Sistema Gerenciador de Banco de Dados Relacional
SQL	*Structured Query Language*
UCP	*Use Case Points*
UML	*Unified Modeling Language*
Xerox PARC	Xerox Palo Alto Research Center
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Nos dias atuais, os desenvolvedores de software comumente utilizam o modelo orientado a objetos (OO) para a produção de sistemas (DEITEL, 2010), e Bancos de Dados Relacionais (BDR) para armazenar suas informações (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). De acordo com Santana (2013), o primeiro padrão vem ganhando espaço, principalmente por causa da linguagem de programação Java, por se tratar da linguagem mais utilizada em todo o mundo, devido aos seus diversos recursos. Por outro lado, os BDR firmaram-se no mercado e vem recebendo um grande investimento da indústria há muito tempo.

Todavia, esses padrões não são fáceis de serem integrados, pois a OO concentra-se em conceitos de objetos e os BDR trabalham com a álgebra e cálculo relacional. Dois paradigmas distintos que acabam convivendo juntos, gerando assim uma incompatibilidade semântica entre eles (VIESI, 2012).

Segundo Bauer e King (2005), tradicionalmente, foram subestimados a importância e o custo desta incompatibilidade e as ferramentas para solucionar o conflito têm sido insuficientes. Enquanto isso, os desenvolvedores Java culpam a tecnologia relacional pela divergência, e os profissionais de banco de dados culpam a tecnologia de objetos. Isto acabou criando uma solução chamada de mapeamento objeto relacional (ORM). O ORM é a persistência de objetos automatizada (e transparente) dentro de um aplicativo Java para as tabelas em um BDR, usando metadados que descrevem o mapeamento entre os objetos e o banco de dados.

Machado apresenta em sua obra outra dificuldade encontrada hoje em dia:

> A quantidade de informação a ser persistida cresce bruscamente, ano após ano, com a necessidade de se armazenar cada vez mais dados em todos os tipos de aplicações. Desta maneira, buscam-se sempre formas mais eficientes, rápidas, seguras, estáveis, viáveis e, porque não, fáceis para o armazenamento de dados. Atualmente não se armazenam apenas caracteres e simples valores numéricos, mas uma infinidade de dados como vídeos, imagens, áudios, objetos complexos (MACHADO, 2009, p.1).

No intuito de resolver os conflitos encontrados e atender as exigências atuais, um novo modelo de banco de dados foi agregado aos Sistemas de Gerenciamento de

Bancos de Dados (SGBD), chamado de Banco de Dados Orientado a Objetos (BDOO). Essa novidade traz consigo vantagens em relação ao padrão relacional, entre elas, a facilidade ao inserir OO ao banco e administrar dados complexos (PINHEIRO et al., 2013).

Os BDOO tornaram-se uma alternativa para solucionar as divergências mencionadas acima, uma vez que tanto a linguagem Java, quanto essa nova classe de bancos de dados trabalham com o paradigma OO, podendo trazer benefícios em relação a utilização de modelos diferentes.

Hoje em dia, as empresas investem pesado em soluções que possam diminuir o esforço e encurtar o prazo de desenvolvimento de suas aplicações. Por esse motivo, as organizações buscam a todo o momento a combinação ideal entre linguagens de programação e banco de dados para que consigam estar à frente neste mercado competitivo.

À vista disso, este trabalho tem por finalidade verificar o ganho no desenvolvimento, empregando paradigmas de desenvolvimento de software e mecanismos de armazenamento de dados compatíveis e distintos, avaliando por meio de métricas os benefícios e os problemas encontrados durante o processo de criação dos protótipos.

1.2 Formulação e Escopo do Problema

Nas obras de Deitel (2010) e Silberschatz, Korth e Sudarshan (2006), apontam que os softwares em sua maioria são desenvolvidos e armazenados nos paradigmas OO e relacional, respectivamente. Entretanto, Viesi (2012) afirma existir uma incompatibilidade semântica ao empregar modelos diferentes.

Diante deste cenário, soluções para esse conflito e outras combinações entre paradigmas são frequentemente buscados (MACHADO, 2009). O ideal para isto é desenvolver duas aplicações que realizam as mesmas funções, empregando dois paradigmas distintos de banco de dados, os BDR e BDOO, com o intuito de avaliar qual combinação, entre linguagem de programação e banco de dados oferece mais vantagens em um projeto de software.

Neste projeto são avaliados os requisitos tamanho da aplicação, esforço e prazo para o desenvolvimento. Portanto, quesitos como: manutenção e reutilização de código, validação de dados, velocidade de execução, entre outros assuntos serão abordados em trabalhos futuros.

1.3 Justificativas

O motivo da realização deste trabalho é devido à frequente procura e carência de informações a respeito de qual combinação entre linguagem de programação e banco de dados obtém mais benefícios em um projeto de software, no ponto de vista de tamanho da aplicação, tempo estimado e esforço requerido.

A busca de uma união ideal entre modelos de desenvolvimento de software e armazenamento de informações é constante (MACHADO, 2009).

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Realizar um estudo comparativo verificando o ganho no desenvolvimento de aplicações que usam paradigmas de linguagem de programação e armazenamento de dados compatíveis e distintos. Para isto serão criados dois protótipos, o primeiro utilizando o modelo orientado a objetos como linguagem de produção do software e do banco de dados. O segundo será empregado o paradigma relacional para o armazenamento das informações e o modelo orientado a objetos para o desenvolvimento do sistema. A análise comparativa das aplicações será feita por meio da métrica de estimativas de software *Construtive Cost Model* (COCOMO), que irá estimar os requisitos, tamanho, esforço requerido e prazo para o desenvolvimento dos dois softwares.

1.4.2 Objetivos Específicos

- Escolher os BDR e BDOO, determinando um banco de dados de cada paradigma para o prosseguimento do trabalho;
- Projetar um protótipo de sistema utilizando uma metodologia de desenvolvimento documentada por meio da *Unified Modeling Language* **(**UML);
- Desenvolver as aplicações na ferramenta escolhida com os SGBDs determinados; e
- Avaliar o desenvolvimento por meio do modelo COCOMO, analisando os requisitos, tamanho do protótipo expresso em linhas de código, prazo para o desenvolvimento e o esforço necessário.

1.5 Organização do Trabalho

O trabalho está divido em cinco capítulos: Introdução, Fundamentação Teórica, Método, Desenvolvimento e Conclusão. O primeiro traz uma contextualização do assunto que será abordado, a formulação e escopo do problema, o motivo para a realização do presente projeto e seu objetivo geral e específicos.

No capítulo 2 são apresentados os conceitos de banco de dados, SGBD, e duas classes de banco de dados, o relacional e o OO. Neste capítulo também será exposto as definições de estimativas de projeto de software, métricas de software e as características das métricas *Lines of Code* (LOC), Análise de Pontos por Função (APF), *Use Case Points* (UCP) e COCOMO.

O capítulo 3 apresenta a metodologia utilizada para atingir os objetivos definidos. No 4° capítulo é exibido o desenvolvimento do projeto, as especificações do sistema que foram definidas para a realização do trabalho, por fim, são apresentados os cálculos das estimativas de esforço e tempo de desenvolvimento das duas aplicações desenvolvidas e os resultados obtidos.

A conclusão encontra-se no último capítulo, nela estão apresentadas as considerações finais do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordadas as definições de banco de dados, SGBD, as principais características dos BDR e BDOO e uma breve comparação entre os dois paradigmas. Serão apresentados os conceitos de estimativas e métricas de projeto de software, as características dos modelos mais comumente empregados para a realização de avaliações de software, as métricas LOC, APF, UCP e COCOMO, descrevendo as vantagens e desvantagens de cada modelo que são relevantes para realização do trabalho.

2.1 Banco de Dados

Os bancos de dados estão cada vez mais presentes em nosso cotidiano, visto que a maioria das atividades que realizamos envolve, direta ou indiretamente, o uso de uma base de dados. É muito provável que você está usando um banco de dados toda vez que compra mercadorias em uma loja, confere um livro em uma biblioteca ou realiza uma aquisição pela Internet.

Segundo Setzer e Silva (2005), a expressão banco de dados vem do inglês, que foi primeiramente denominada de *databanks*, e logo foi substituída por *database*, que traduzindo para o português significa base de dados. Esse nome é bem mais sugestivo, uma vez que um banco de dados não atua como um banco, emprestando dados. Funciona como um repositório de dados, que são utilizados em diversas aplicações, isto é, uma base sobre a qual atuam essas aplicações, e está disponível para o desenvolvimento de outras que usam os mesmos dados.

Após elucidar o funcionamento de um banco de dados, Costa (2011) traz em sua obra algumas vantagens ao se empregar banco de dados:

- **Controle centralizado dos dados:** como os dados estão convergidos em um mesmo local, isto proporciona um maior controle;
- **Controle de redundância:** como dito acima, os dados estão centralizados, por isso os dados são armazenados apenas uma vez,

evitando a redundância e também gera uma economia no espaço de armazenamento;

- **Independência dos dados:** no sistema de armazenamento em arquivos, a especificação da estrutura e do método de acesso à informação está incorporada ao código das aplicações, com isso se torna impossível alterar a estrutura dos arquivos sem modificar a respectiva aplicação. Entretanto, os bancos de dados são independentes de dados, pois permitem uma abstração de dados; e
- **Eliminação de inconsistência e garantia da integridade:** diz-se que um arquivo é inconsistente, quando apresenta entradas diferentes de um mesmo dado. E se faltar consistência, não existe integridade (o arquivo possui informações divergentes). Em banco de dados é possível controlar a integridade e a consistência dos dados.

De acordo com Carvalho (2011), bancos de dados são gerenciadores de grandes grupos de informações utilizados com a finalidade de modelar algum tipo de organização ou processo organizacional. Esse gerenciamento é executado pelos SGBDs, softwares que possuem recursos que definem uma estrutura para o armazenamento de informações e fornecem mecanismos para manipulá-las.

2.1.1 Sistema Gerenciador de Banco de Dados

O primeiro SGBD comercial teve origem no final da década de 60. Este evoluiu do sistema de arquivos. Todavia, os sistemas não controlavam o acesso concorrente por vários processos e usuários (DATE, 2000).

Os SGBDs progrediram dos sistemas de armazenamento em disco criando novas estruturas de dados para o armazenamento de informações. Estes SGBDs utilizam diversos paradigmas de dados diferentes para representar a estrutura de informação no banco de dados, tais como: os modelos hierárquicos, de redes, relacional e o modelo orientado a objetos.

Sudarshan, Silberschatz e Korth (2000), definem o SGBD como uma coleção de programas com o propósito de facilitar os processos de definição, construção,

manipulação e compartilhamento de banco de dados entre vários usuários e aplicações.

O conceito divide-se em:

- **Definição:** especificar os tipos de dados, as restrições e estruturas;
- **Construção:** procedimento de armazenamento de informações em alguma mídia adequada;
- **Manipulação:** atividade de busca e atualização dos dados; e
- **Compartilhamento:** permite o acesso concorrente de múltiplos processos e usuários.

Para Setze (2005), o SGBD tem como principal finalidade proporcionar uma maneira de recuperar as informações armazenadas no banco de dados, que seja tanto conveniente como eficiente.

Date (2000) também cita outros objetivos em sua obra, tais como:

- Garantir a privacidade das informações por meio de métodos de segurança, como senhas, criptografia e permissões;
- Concede dados íntegros ao usuário; e
- Os dados são compartilhados de maneira organizada. O SGBD trabalha como um mediador entre as aplicações e o banco de dados.

A seguir serão listados alguns dos SGBDs mais utilizados.

- Postgres;
- MySQL;
- Oracle;
- IBM DB2; e
- Microsoft SQL Server.

2.1.2 Banco de Dados Relacional

Conforme Nascimento (2012), a tecnologia de banco de dados vem progredindo rapidamente nas últimas décadas desde a evolução e eventual domínio dos Sistemas Gerenciadores de Banco de Dados Relacional (SGBDR). Devido a essa evolução, os

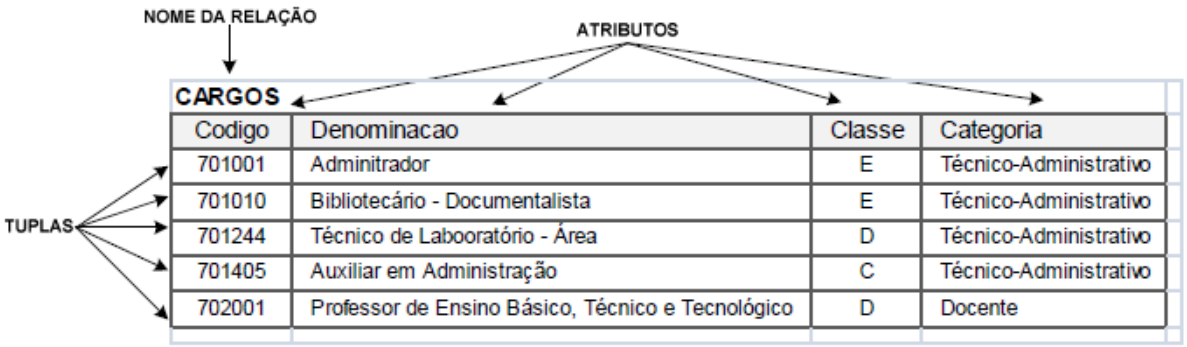
BDR geraram uma indústria multibilionária, e é o modelo mais utilizado em todo o mundo atualmente (SANTANA, 2013).

Um BDR é um banco de dados que segue o Modelo Relacional (MR). Para Elmasri e Navathe (2011), o MR é um padrão de dados representativo ou de implementação, fundamentando-se em conceitos matemáticos.

Segundo Date (2004), o MR refere-se a “três aspectos principais de dados: a estrutura de dados, a integridade dos dados e a manipulação dos dados”.

Neste modelo, o banco de dados é representado como um conjunto de relações. Considerando que uma relação é, de certo modo, análogo a uma tabela de valores, e em conformidade com a terminologia deste modelo, diz-se que as linhas e as colunas são denominadas de tuplas e atributos, respectivamente (ELMASRI; NAVATHE, 2011, p. 39). Na Figura 1 são apresentados os conceitos citados acima.

Figura 1 Representação das relações em um BDR



CARGOS

Codigo	Denominacao	Classe	Categoria
701001	Adminitrador	E	Técnico-Administrativo
701010	Bibliotecário - Documentalista	E	Técnico-Administrativo
701244	Técnico de Labooratório - Área	D	Técnico-Administrativo
701405	Auxiliar em Administração	C	Técnico-Administrativo
702001	Professor de Ensino Básico, Técnico e Tecnológico	D	Docente

Fonte: Costa (2013)

A ordem física dos registros ou campos em uma tabela é completamente irrelevante, e cada registro na tabela é identificada por um campo que contém um valor único.

O MR traz consigo o aspecto de integridade, e nesta parte será definido os conceitos de superchave, chave primária, chave estrangeira.

Na definição de Date (2004, p. 233), um aglomerado de atributos é dito chave se satisfazer as condições de:

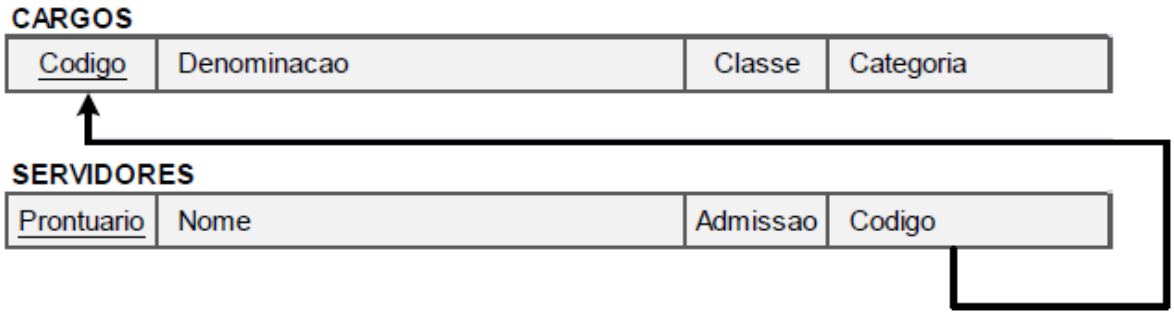
- **Unicidade:** esta propriedade garante que os dados contidos na coluna, ou no grupo de colunas são únicos em relação a todas as outras linhas da tabela; e

- **Irredutibilidade:** estabelece que não deva existir no conjunto de atributos chamado chave um subconjunto que tenha a propriedade de unicidade, ou seja, a chave deve ser uma coleção mínima de atributos.

Conforme Costa (2011), superchave é um agrupamento de um ou mais atributos que, tomados coletivamente, nos permite identificar de modo unívoco uma entidade em um conjunto de entidades.

A relação entre um par de tabelas é estabelecido implicitamente por meio de valores correspondentes de um campo compartilhado.

Figura 2 Demonstração da chave primária e estrangeira



Fonte: Costa (2011)

Na Figura 2, está sendo apresentando a relação entre as tabelas CARGO e SERVIDORES. Na tabela CARGOS é possível verificar que o campo “Codigo” está sublinhado, comumente o campo sublinhado faz referência à chave primária da tabela. De acordo com Nascimento (2012), as chaves primárias referem-se aos conjuntos de um ou mais campos, cujos valores, considerando a combinação de valores de todos os campos da tupla, nunca se repetem e que são utilizados como um índice para os outros campos da tabela do banco de dados. Em outras palavras, caso a chave primária seja simples, ou seja, é formado por apenas um único campo da tabela, esse campo não pode ter dois ou mais registros de mesmo valor. Entretanto, se a chave primária é composta, isto é, formada por mais de um campo, os valores de cada campo podem se repetir, mas não a combinação desses valores. É de suma importância frisar que em chaves primárias não pode haver valores nulos nem repetição de tuplas.

Continuando a analisar a Figura 2, podemos observar o relacionamento entre duas tabelas, na qual o campo “Codigo”, que é a chave primária da tabela CARGOS

está presenta na tabela SERVIDORES. Todavia, este campo não está sublinhado, pois faz referência à tabela CARGOS, neste caso o campo “Codigo” na tabela SERVIDORES é denominado de chave estrangeira. Para Setze (2005), a finalidade da chave estrangeira é garantir a integridade dos dados referenciais, uma vez que, apenas serão permitidos valores que vão aparecer na base de dados. Outra definição encontrada é a de Date (2000), onde ele diz que, uma chave estrangeira é um atributo ou uma combinação de atributos numa relação, cujos valores são fundamentais para equivaler à chave primária de uma relação.

2.1.2.1 SQL

A Structured Query Language (SQL), ou Linguagem de Consulta Estruturada é atualmente a linguagem de pesquisa declarativa padrão para os SGBDR (ELMASRI; NAVATHE, 2011, p. 57).

No início de 1970, a SQL foi criada pela IBM Research para o protótipo de um sistema de banco de dados denominado System R(DATE, 2004, p.71). Conforme Elmasri e Navathe (2011), este modelo foi baseado nas linguagens de álgebra relacional e cálculo relacional, e primeiramente chamado de SEQUEL (Structured English QUEry Language).

A SQL por ser um modelo seguido por uma classe de banco de dados, ele deve ser padronizada, e quem realiza esta função de forma conjunta são duas organizações, a American National Standards Institute (ANSI) e a International Standards Organization (ISO).

De acordo com Silberschatz, Korth e Sudarshan (1999), embora que conhecida e denominada como “linguagem de consulta”, este paradigma disponibiliza recursos para definir a estrutura dos dados (inserir, excluir e editar), especificar restrições de integridade e outros mecanismos.

Segundo Date (2004), a SQL possui os seguintes componentes: Data Definition Language (DDL) ou Linguagem de Definição de Dados, Data Manipulation Language (DML) ou Linguagem de Manipulação de Dados e Data Control Language ou Linguagem de Controle de Dados (DLC), entre outros.

A DDL é utilizada para especificar relações, domínios, regras de integridade (ELMASRI; NAVATHE, 2011, p. 58). Para Date (2004), a DML é um subconjunto da linguagem SQL que é usada para realizar consultas, inclusões, alterações e exclusões de dados presentes na base de dados. Outro componente da linguagem SQL é a DLC, a DLC de acordo com Silberschatz, Korth e Sudarshan (1999), controla os aspectos de autorização de dados e licença de usuários para fiscalizar quem tem acesso para consultar ou manipular os dados dentro do banco de dados.

2.1.3 Banco de Dados Orientado a Objetos

O desenvolvimento dos Sistemas Gerenciadores de Banco de Dados Orientados a Objetos (SGBDOO) teve origem na combinação de ideias dos modelos de dados tradicionais e de linguagens de programação OO (ISHIKI, 2004, p.12).

Conforme Teles et al. (2013), a criação dos BDOO ocorreu quando houve a necessidade de se trabalhar com aplicações complexas, implementadas em linguagens OO e com estrutura de difícil armazenamento de dados.

Segundo Elmasri (2005), um BDOO é um gerenciador de grandes volumes de informação que tem a estrutura de armazenamento de dados baseado no paradigma de OO, todos os dados são armazenados na forma de objetos que são regidos por uma determinada classe.

Como mencionado acima, os BDOO constituem modelos diferentes dos relacionais, a seguir será apresentado os conceitos básicos do modelo orientado a objetos e dos SGBDOO.

2.1.3.1 Paradigma da Orientação a Objetos

A OO não é um conceito novo. De acordo com Carvalho (2011), seu roteiro pode ser evidenciado pela Noruega, por volta de 1960, do qual se encontrava relacionado com uma linguagem denominada SIMULA-67.

Esta linguagem apresentava os conceitos de classes e subclasses que são muito parecidas com as linguagens OO atuais. Posteriormente, em meados da década de 70, cientistas da Xerox Palo Alto Research Center (Xerox PARC), desenvolveram a

primeira linguagem OO considerada robusta, que foi chamada de Smalltalk (ISHIKI, 2013, p.22).

Na década seguinte, a linguagem C tornou-se muito conhecida entre os desenvolvedores e diante disso proporcionou sua expansão para a linguagem C++ e mais adianta surgiu a linguagem Java, com isso ajudou a chamar a atenção da comunidade de desenvolvimento à Programação Orientada a Objetos (POO), conforme Carvalho (2011) traz em sua obra.

Portanto, a OO não é um paradigma recente e traz consigo diversas vantagens, tais como, abstrações do mundo real, reutilização de código e requisitos, a manutenção de código se torna mais fácil, entre outros benefícios.

2.1.3.2 Abstração

Segundo Carvalho (2011), a abstração consiste em considerar apenas as propriedades comuns de um aglomerado de objetos, omitindo os detalhes, utilizada com frequência na definição de valores similares e na formação de um tipo a partir de outro, em diferentes níveis de abstração.

2.1.3.3 Objeto

Um objeto é um elemento que podemos criado, destruído, manipular e acompanhar o seu comportamento (CARVALHO, 2011). De acordo com o dicionário, objeto é “tudo que é apreendido pelo conhecimento, que não é o sujeito do conhecimento; tudo que é manipulável; tudo que é perceptível por qualquer um dos sentidos; o que é conhecido, pensado ou representado”.

Portanto, objeto é a representação de elementos físicos do mundo real, que descobrimos estudando suas características (atributos) e seus comportamentos (ações) (ISHIKI, 2013).

2.1.3.4 Classe

Ao equiparar objetos distintos, podemos evidenciar que eles possuem características e ações semelhantes, dependendo da finalidade para a qual são analisados. Empregando a abstração é possível agrupar alguns objetos de acordo com seus atributos e comportamentos em comum.

À vista disso, uma classe é um modelo onde os objetos são instanciados (originados). Conforme Carvalho (2011), uma classe defini atributos e comportamentos de um tipo de objeto e é obtido pela classificação de objetos com a mesma estrutura de dados e o mesmo comportamento. Por exemplo, em um modelo, Lucas, Leonardo e Maria podem ser instanciados na classe Pessoa, visto que possuem atributos semelhantes (nome) e ações comuns (andar, falar, comer, entre outros).

2.1.3.5 Encapsulamento

Também conhecido como “ocultação de informações”, o encapsulamento segundo Carvalho (2011), é uma técnica que consiste em separar os aspectos internos dos aspectos externos de um objeto. Em outras palavras, pode-se afirmar que encapsulamento é uma prática para proteger a estrutura interna do objeto por traz dos métodos.

2.1.3.6 Polimorfismo

De acordo com Ricarte (1998), polimorfismo é o mecanismo que permite expressar que ações similares em classes distintas venham a ter um comportamento uniforme. Um exemplo de polimorfismo é, deseja-se que um método denominado “exibir” em duas classes diferentes tenham um mesmo comportamento, independentemente das diferenças internas de sua implementação.

2.1.3.7 Herança

Da mesma forma em que ocorre nas classes, ao compararmos e analisar os objetos do mundo real podem verificar atributos e comportamentos comuns entre objetos fazendo com que esses sejam agrupados em um mesmo tipo de classe, como dito acima. Todavia, ao investigarmos um pouco mais a fundo, descobrimos que alguns desses objetos pertencentes a essa classe possuem características singulares além daquelas em comum. Em vista disso, a técnica de herança consegue suprir essa diferença entre os objetos.

Ricarte (1998) traz em sua obra, herança é o mecanismo que permite reaproveitar declarações de classe, criando classes derivadas a partir de declarações de classes bases preexistentes. Já para Carvalho (2011), o mecanismo de herança é a capacidade de uma classe herdar os atributos e comportamentos de outra classe.

2.1.3.8 ODMG

O Object Database Management Group (ODMG) segundo Ishiki (2004) é um grupo criado em 1991, que tem por finalidade integrar e padronizar as funcionalidades de banco de dados em uma linguagem de programação OO. Os componentes do padrão são:

- Modelo de objetos;
- Linguagem de Definição de Dados (ODL);
- Linguagem de Consulta; e
- Acoplamento com linguagens de programação OO.

A ODL de acordo com Ishiki (2004) consiste em uma linguagem de definição para especificação de classes. Ela é simples, extensível e mapeável para toda linguagem OO.

A Object Query Language (OQL) para Carvalho (2011) é uma linguagem de consulta declarativa que foi definida pela ODMG. Este padrão é uma extensão da linguagem SQL, utilizada pelos BDR.

Os BDOO possuem algumas vantagens sobre o MR, eles oferecem rapidez na inserção de dados, utilizam pouco recurso computacional, acessam diretamente o banco de dados sem utilizar um ORM e existe a possibilidade de abrir mão de empregar um Administrador de Banco de Dados (DBA), pois a alterações e análises podem ser feitas pelos programadores (GALANTE et al. 2013).

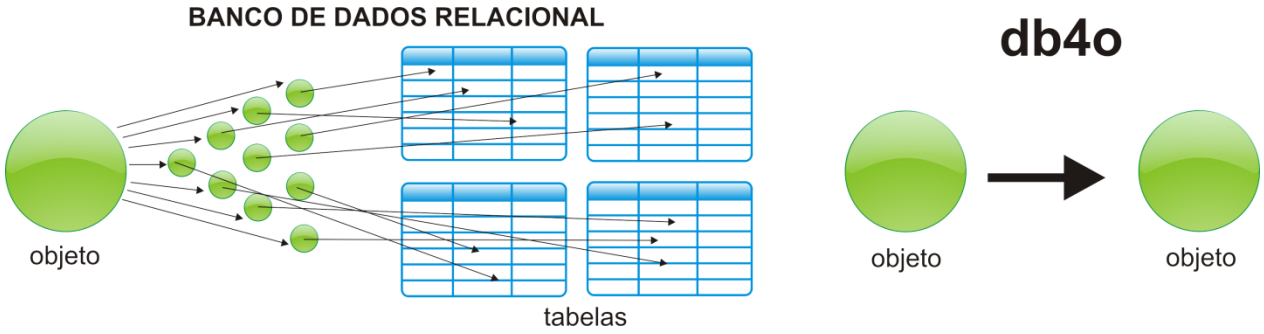
2.1.4 Banco de Dados Relacional x Banco de Dados Orientado a Objetos

Em determinadas situações o BDR predomina, porém, em outros casos estes não atendem de maneira eficiente todas as necessidades, com isto o modelo OO é empregado.

Ishiki (2004) traz em sua obra a diferença entre o tratamento da redundância nos dois paradigmas. No MR a redundância é tratada por meio da normalização, enquanto que no modelo OO o encapsulamento e a herança fornecem dados não redundantes.

A grande disparidade entre estes modelos é a maneira em que os dados são armazenados (MACHADO, 2009), a Figura 3 apresenta a maneira em que o BDR e o BDOO guardam suas informações.

Figura 3 Armazenamento de dados em um BDR e em um BDOO



Fonte: Galante et al. (2013)

Na Figura 3, é possível verificar uma das diversas vantagens ao se empregar um BDOO, este modelo armazena o objeto como está na aplicação no banco de dados, essa é o benefício citado por Galante et al. (2013) quando diz que a velocidade de inserção é superior aos outros bancos de dados. Em contrapartida, os BDR “quebram”

o objeto e armazenam suas informações em tabelas, o que pode se tornar um processo vagaroso.

Abaixo será apresentada alguma das vantagens e desvantagens entre os paradigmas.

2.1.4.1 Vantagens do Modelo Relacional

- Muito utilizado e com um grande legado (DATE, 2000);
- Utilização de diversas aplicações em um mesmo banco de dados (PATERSON et al., 2006);
- O banco de dados pode ser alterado e manipulado independentemente da aplicação (DATE, 2000); e
- Vasto número de ferramentas que implementam SQL, ferramentas de relatório, de *On-line Analytical Processing* (OLAP), backup, conectividade, recuperação de dados, entre outras ferramentas.

2.1.4.2 Desvantagens do Modelo Relacional

- Codificação de conversão entre objetos e tuplas opera fora do paradigma de programação OO e há um risco maior de erros não detectados (Silberschatz et al., 1999);
- Usa códigos sem significado semântico para que se possam identificar as tuplas de seleção (MACHADO, 2009);
- Necessidade de usar o ORM; e
- Necessidade de seleção de chaves entre tabelas;

2.1.4.3 Vantagens do Modelo Orientado a Objetos

- Permite a manipulação direta no banco de dados, utilizando a linguagem de programação (Silberschatz *et al.*, 1999);
- Possibilita o armazenamento direto dos tipos de dados complexos;
- Possui maior consistência na manipulação das informações;

2.1.4.4 Desvantagens do Modelo Orientado a Objetos

- Ainda é pouco empregado em relação ao outro paradigma (DATE, 2000);
- As linguagens de consultas não são padronizadas (MACHADO, 2009); e
- As chances de ocorrer um erro no banco de dados são maiores, visto que os dados podem ser acessados diretamente por meio da linguagem de programação.

2.2 Estimativas de Projeto de Software

Uma das atividades fundamentais do processo de gerenciamento de projetos de software é o planejamento. A ação de planejar consiste em um conjunto de operações que visam estimar tempo e recursos necessários ao projeto, definir tarefas e preparar uma estrutura para o gerenciamento (HUMPHREY, 1989).

Segundo Paulk et. al. (1993), para que um projeto de software possa ser efetivamente planejado, estimativas de custo, em termos de esforço, prazo e orçamento, devem ser derivadas logo no início, por meio de um processo bem definido – o processo de estimativa de custo de software.

Pressman traz em sua obra: “software é o elemento virtualmente mais caro de todos os sistemas baseados em computador. Para sistemas complexos, feitos sob medida, um erro de estimativa grande pode fazer a diferença entre o lucro e o prejuízo. Excesso de custo pode ser desastroso para o desenvolvedor” (2002, p. 117).

> “Em grande parte dos casos, as estimativas são feitas com base na experiência passada. No caso de se ter um projeto relativamente similar a um projeto já realizado, não fica difícil estimar questões como esforço, cronograma e custo, uma vez que estes serão muito próximos daqueles relativos ao projeto anterior” (PRESSMAN, 2002, p. 118).

Isto pode funcionar relativamente bem, se o projeto atual for bastante semelhante a esforços anteriores e as outras influências do projeto (exemplos: o cliente, os prazos) forem equivalentes. Infelizmente a experiência anterior nem sempre é um bom indicador de resultados futuros (PRESSMAN, 2002).

Segundo Sommerville (2003, p. 443).

> [...] cada técnica de estimativa tem seus próprios pontos positivos e negativos. No caso de grandes projetos, é necessário utilizar várias técnicas de estimativa

> de custos e comparar seus resultados. Se elas predizem custos radicalmente diferentes, isso sugere que não se tem informações suficientes para fazer a estimativa. É necessário obter mais informações e repetir o processo de estimativa, até que as estimativas sejam convergentes.

De acordo com Conte (1986), os modelos formais mais conhecidos são: o modelo de alocação de recursos (Putnam, 1992), o modelo APF (Albrecht, 1983), o UCP e o COCOMO (Boehm, 1981), na qual é o modelo mais conhecido, mais completo e documentado de todos os modelos de estimativa.

2.3 Métricas de Software

De acordo com Carvalho, Chiossi e Drach (2004), métricas de software são padrões quantitativos de medidas de vários aspectos de um sistema de software, e se constituem em uma poderosa ferramenta gerencial, contribuindo para a elaboração de estimativas de prazo e custo mais precisas e para o estabelecimento de metas plausíveis, facilitando assim o processo de tomada de decisões e a posterior obtenção de medidas de produtividade e qualidade.

Métrica é uma medida pura e simples. Ao medir as coisas de maneira consistente fica se a par do que está sendo feito e qual é a sua eficiência. Ela pode ser aplicada independentemente da área ou desígnio, porém comumente utilizada no mundo da engenharia (AMBLER,1998).

Há pouco tempo atrás, a única base para a realização de estimativas era a experiência da equipe técnica envolvida. Dessa forma, ocasionava vários problemas no desenvolvimento e na entrega do software, entre elas, projetos não realizados no tempo e custo previstos e produtos com deficiência funcional (CORDEIRO, 2000).

Conforme Pressman (1995) existem duas categorias em que as métricas podem se dividir, no ponto de vista da medição, são elas: medidas diretas e medidas indiretas. Incluídos no grupo das medidas diretas, pode-se considerar o custo e o esforço dedicado no desenvolvimento e manutenção do produto, o número de linhas de código produzido, a quantidade de defeitos registrados durante um determinado período de tempo, a velocidade de execução e a memória utilizada. Todavia, a qualidade, funcionalidade, confiabilidade e a capacidade de manutenção do software são unidades

de medidas mais complicadas de serem estimadas, dessa forma só podem ser medidas indiretamente (CORDEIRO, 2000).

Outra divisão das métricas de acordo com Pressman (1995) é:

- **Métricas orientadas ao tamanho:** são medidas diretas do software e do processo por meio do qual ele é desenvolvido. Estas medidas são derivadas a partir de atributos de tamanho do software como o modelo LOC;
- **Métricas orientadas à função:** são medidas indiretas do software e do processo por meio do qual o software é desenvolvido, é independente tecnologia empregada. Estas métricas concentram-se na funcionalidade ou qualidade do software, um exemplo destas métricas é a técnica APF; e
- **Métricas orientadas às pessoas:** estas métricas utilizam informações sobre a maneira pela qual as pessoas desenvolvem software e percepções humanas sobre a efetividade das ferramentas e métodos.

Além dessas categorias de métricas citadas acima, existem outras três no ponto de vista da aplicação, são elas, métricas de qualidade, métricas de produtividade e métricas técnicas. As métricas de qualidade indicam o quanto o software atende aos requisitos definidos pelo cliente, as métricas de produtividade concentram-se na saída do processo de engenharia de software e as métricas técnicas concentram-se nas características do software, tais como complexidade lógica e modularidade (CORDERIO, 2000).

2.4 Linhas de Código

O sistema LOC, é a técnica de estimativa mais antiga. Ela pode ser aplicada para estimar o custo do software ou para especificar igualdades de analogias (ARIGOFLU, 1993). Existem muitas discussões sobre esta métrica, uma delas é que muitos pesquisadores defendem o ideal de não incluir os comentários e as linhas em branco na somatória, visto que elas não contribuem para a funcionalidade do sistema.

As vantagens do sistema LOC são: a facilidade de obtenção do número de linhas de código e aplicabilidade, ela é utilizado por muitos modelos de estimativas de software. Em contrapartida, esta técnica é fortemente ligada à linguagem de

programação usada, impossibilitando a utilização de dados históricos para projetos que não utilizam a mesma tecnologia (PRESSMAN, 1995). A dificuldade de estimar no início do projeto também é apontada como um grande problema encontrado.

Portanto, nota-se que a técnica de somatória de linhas de código não é uma métrica a ser utilizada por si só, ela deve ser aplicada juntamente com um conjunto de outras métricas de estimativas, efetuando um comparativo de resultados. Deste modo uma técnica pode completar a outra, fornecendo informações que são pertinentes às características de cada uma (SEIBT, 2001).

2.5 Análise de Pontos por Função

De acordo com Freire (2013), em 1979, Allan Albrecht prosseguindo as pesquisas sobre métricas orientadas a função, introduziu a técnica de estimativa APF. Esta técnica baseia-se na ideia de que um sistema pode ter seu tamanho estimado de acordo com o número e complexidade dos requisitos funcionais que o compõem.

O objetivo deste modelo matemático é conseguir uma medida de tamanho do produto que possa estar disponível desde o início do processo de desenvolvimento, é um método independente de tecnologias, baseado na funcionalidade do sistema. O ponto inicial da execução da técnica é determinar certo número de itens que ocorrem no sistema (CORDEIRO, 2000).

Segundo o IFPUG (1991), determinam-se os pontos por função de uma aplicação em três etapas de avaliação. A primeira resulta no cálculo de função não ajustados, que refletem as funções específicas e mensuráveis do negócio, provida ao usuário pela aplicação, essas funções são: arquivos lógicos internos, arquivos de interface externa, entradas externas, saídas externas e consultas externas. A segunda etapa da avaliação gera o fator de ajuste, que representa a funcionalidade geral provida ao usuário pela aplicação. Para o cálculo do fator de ajuste devem ser consideradas 14 características gerais dos sistemas, são elas, comunicação de dados, processamento distribuído, desempenho, utilização de equipamento, volume de transações, entrada de dados "on-line", processamento complexo, reutilização de código, facilidade de implantação, facilidade operacional, múltiplos locais, facilidade de mudanças. A terceira

etapa resulta na contagem de pontos por função ajustados, que reflete o fator de ajuste aplicado ao resultado apurado na primeira etapa.

Para Ambler (1998), embora os pontos de função sejam uma abordagem útil para a comparação de aplicações, até mesmo para aplicações que são desenvolvidas utilizando diferentes técnicas e ferramentas, podem ser caros e difíceis de colecionar. A contagem de pontos é uma habilidade que poucas pessoas possuem, e é um processo que consome tempo até mesmo para pessoas que têm experiência nisto.

2.6 Use Case Points

Conforme Andrade (2004), UCP é um método de estimativa de projeto de software OO proposto em 1993 por Karner, com base na APF e no modelo de casos de uso. O UCP também estima o tamanho da função do software de acordo com a visão do usuário final.

O modelo UCP somente pode ser empregada em projetos de software cuja especificação tenha sido expressa em casos de uso. Nela contam-se os atores e os casos de uso e identifica-se sua complexidade, para então proceder com o cálculo inicial, determinação de complexidades e então chegar ao resultado final (HAZAN; FUKS; LUCENA, 2005).

Esta técnica surgiu alicerçada na análise de pontos por função. A principal diferença entre elas está relacionada no momento em que se coletam as informações para a realização das estimativas do projeto. Segundo Andrade (2004), APF possui resultados melhores na medida em que se tem mais informação da análise e do projeto de sistemas (tabelas, campos, associações). Já UCP tem como proposta ser utilizado logo no início do ciclo de desenvolvimento, na fase de definição dos requisitos, com base no modelo de casos de uso.

Para executar a medição de UCP, há seis passos que devem ser seguidos, conforme Andrade (2004) e Rocha (2005) e Vieira (2007), os quais são citados a seguir, verificar atores envolvidos e atribuir os devidos pesos, contagem e atribuição dos pesos dos casos de uso, apurar UCP não ajustados, determinação do fator de complexidade

técnica, determinação do fator de complexidade ambiental e cálculo dos UCP ajustados.

Finalizando esta técnica, Andrade (2004) por meio do estudo de alguns autores, pode concluir que UCP não é amplamente aplicado, pois existe uma série empecilhos, entre elas, ainda não alcançou o nível de padronização, trata-se de uma técnica relativamente nova, não apresenta bases históricas de produtividade e não incorpora ferramentas disponíveis de medição de software.

2.7 COCOMO

Apresentado em 1981 por Boehm, o COCOMO é um modelo desenvolvido para estimar esforço, prazo, custo e tamanho da equipe em um projeto de software (JÚNIOR; SANCHES, 2000). De acordo com Trindade, Pessoa e Spindola (1999), a variável custo neste modelo não há uma forma direta de dimensioná-lo. Porém, tendo conhecimento de prazo e equipe de trabalho, torna-se possível estimar um valor.

Segundo Júnior e Sanches (2000), As equações do COCOMO foram derivadas a partir do estudo de uma base de informações de 63 projetos realizados ao longo de um período de 15 anos, de 1964 até 1979, em sua maior parte na empresa *TRW Systems*, *Inc.*

O COCOMO é apresentado na forma de um conjunto de modelos divididos hierarquicamente em três níveis: Básico, Intermediário e Avançado. A seguir serão apresentadas as características dos modelos, segundo Júnior e Sanches (2000):

1) **Básico*:*** calcula o esforço do desenvolvimento de software em função do tamanho estimado do programa expresso em linhas de código;
2) **Intermediário*:*** realiza a contagem do esforço de desenvolvimento de software em função do tamanho do programa e de um conjunto de direcionadores de custo, alternativamente chamados atributos ou fatores de software, que incluem avaliações subjetivas do produto, do hardware, do pessoal e dos atributos do projeto; e

3) **Avançado:** incorpora todas as características da versão intermediária, incluindo a avaliação do impacto dos atributos do software e da equipe desenvolvedora em cada passo do processo de engenharia de software.

A técnica COCOMO estima o esforço em pessoas/meses de trabalho, e toma como principal fator de esforço o LOC, expressas em milhares de linhas de código, *Kilo Lines of Code* (KLOC) (PETERS; PEDRYCZ, 2001).

Para melhor análise podemos nos remeter a Sommerville (2003, p.446).

> [...] este modelo sugere que o desenvolvimento do software se dê em cascata (surgindo 'a partir do zero'), porém, aconteceram mudanças desde esta proposição. Softwares são implementados utilizando componentes reutilizáveis, são interligados por linguagens de roteiro, utilizam prototipação, desenvolvimento incremental e, em muitos casos, subsistemas de prateleira para a composição do software.

Após a análise dos requisitos funcionais do software o projeto deve ser classificado em um dos três modos de desenvolvimento, identificados por Boehm: Orgânico, Embutido ou Semidestacado (Peters e Pedrycz, 2001):

- No modo Orgânico, as equipes são relativamente pequenas e desenvolvem sistemas num ambiente altamente "familiar". Nesse modo de desenvolvimento, a maior parte das pessoas engajadas no projeto tem experiência prévia com sistemas similares na organização e entendimento completo do sistema. Outras características do modo orgânico são: ambiente estável de desenvolvimento com pouca necessidade de inovação e inexistência requisita de entrega rígida; uso de algoritmos simples; e tamanho relativamente pequeno, com projetos na faixa de até 50.000 linhas de código (Júnior e Sanches, 2000).
- Segundo Sanches e Júnior (2000), o principal fator que difere um projeto de software de modo Embutido ou Restrito é a necessidade de seguir restrições rigorosas. O produto a ser desenvolvido deverá operar dentro de um contexto complexo de hardware, software e regras e procedimentos operacionais. São projetos de software caracterizados por serem relativamente grandes, com muita necessidade de inovação, que demandam altos custos de verificação e validação. Exemplos de projetos

do modo Embutido são: projeto de sistema de transferência eletrônica de fundos e projeto de sistema de controle de tráfego aéreo.

- O modo Semidestacado é empregado em desígnios de software com características que estão entre os modos Eembutido e Orgânico. Suas características fundamentais são: a equipe mescla grande e pouca experiência com a aplicação e com a tecnologia e o tamanho do software pode chegar a 300.000 linhas de código (Peters e Pedrycz, 2001).

2.7.1 Tamanho da Aplicação

Segundo Peters e Pedrycz (2001), para iniciar os cálculos do esforço e do tempo de desenvolvimento, é necessário primeiramente receber o tamanho da aplicação expresso em KLOC, pois o COCOMO toma como base esse requisito para a realização de suas equações. A obtenção deste principal fator foi realizada por meio do plugin Metrics 1.3.6. O Metrics é uma ferramenta desenvolvida para a *Integrated Development Environment* (IDE) Eclipse e oferece diversas funções, tais como, gráficos de dependência de pacotes, apresenta a quantidade de pacotes, de classes, de linhas de código de cada classe e também a somatória delas. A única observação e cuidado que se deve tomar é que esta ferramenta não realiza a contagem da classe de ORM, a somatória foi feita linha por linha. Deste modo, foi possível receber a quantidade de linhas de código de cada aplicação e transformá-las em KLOC, que nada mais é que dividir o total de linhas do protótipo por mil. Toda a documentação, como instalar e utilizar o Metrics são encontrados neste site http://metrics.sourceforge.net/,

2.7.2 Estimativa de Esforço – COCOMO

Em seguida serão exibidas as equações para realizar as estimativas de esforço e prazo para um projeto de software de acordo com este modelo.

Fórmula 1 Cálculo de estimativa de esforço – COCOMO

$$E = a_i S^{b_i}$$

A Fórmula 1 apresenta como deve ser calculado a estimativa de esforço do modelo COCOMO, na qual ***S*** representa o tamanho do software expresso em milhares de linhas de código, excluindo as linhas em branco e comentários.

Tabela 1 Parâmetros do COCOMO – Estimativa de esforço

	Básico		***Intermediário***	
Modo	*ai*	*bi*	*ai*	*bi*
Orgânico	2.4	1.05	3.2	1.05
Semidestacado	3.0	1.12	3.0	1.12
Embutido	3.6	1.20	2.8	1.20

Fonte: (CONTE, 1986)

Os valores para os atributos a_i e b_i para níveis Básicos e Intermediários e para os modos Orgânico, Semidestacado e Embutido estão exibidos na Tabela 1.

2.7.3 Estimativa de Tempo de Desenvolvimento – COCOMO

Além de estimar o esforço, o COCOMO também apresenta equações para estimar o tempo de desenvolvimento nominal do projeto em meses e a divisão do esforço por fases e atividades do projeto (JÚNIOR; SANCHES, 2000, p.4).

Fórmula 2 Equação de estimativa de tempo de desenvolvimento – COCOMO

$$T = a_i E^{b_i}$$

As equações de estimativa de prazo de desenvolvimento são da maneira apresentada na Fórmula 2, na qual ***T*** é o tempo de desenvolvimento e ***E*** o esforço já calculado. De acordo com Júnior e Sanches (2000), os modos de desenvolvimento Básico e Intermediário utilizam as mesmas equações para determinar o prazo.

Tabela 2 Parâmetros do COCOMO – Estimativas de prazo

Modelos Básico e Intermediário		
Modo	*ai*	*bi*
Orgânico	2.5	0,38
Semidestacado	2.5	0,35
Embutido	2.5	0,32

Fonte: (FERNANDES,1995)

A Tabela 2 exibe a relação de valores para níveis Básicos e Intermediários e para os Modos Orgânico, Semidestacado e Embutido.

> “Salienta-se que estes formatos e seus respectivos parâmetros resultam de ajustes de dados obtidos em caráter experimental, advindos de projetos de software anteriores que foram analisados na criação do modelo. Uma limitação imposta por esta metodologia diz respeito ao fato de que certos parâmetros podem não se adequar a projetos diferentes daqueles utilizados como base de criação do modelo” (PETERS; PEDRYCZ, 2001, p. 488).

De acordo com Peters e Pedrycz (2001, p. 487), o modelo COCOMO é o mais concreto e documentado em termos de estimativas de esforço. Esta métrica baseia-se em uma análise de Boehm e fornece fórmulas que possibilitam determinar o cronograma do tempo de desenvolvimento, esforço geral envolvido no mesmo, assim como interrupção do esforço por fase/atividade e ainda o esforço de manutenção.

3 MÉTODO

Este trabalho será incialmente realizado por meio de pesquisas bibliográficas acerca de banco de dados e sobre duas classes de banco de dados, que são os BDR e os BDOO, apresentando também algumas vantagens e desvantagens ao empregar esses dois tipos de armazenamento de dados. Da mesma forma será demonstrado o conceito de SGBD.

Posteriormente completado este estágio, será feito um estudo sobre estimativas e métricas de projeto de software. Em seguida, foi definido que a métrica COCOMO é a mais indicada para medir os requisitos, tamanho, tempo e esforço de um desenvolvimento de software. A escolha iniciou-se realizando um levantamento das principais técnicas de estimativas de projetos de softwares, sendo elas, a LOC, a APF, o UCP e o COCOMO.

Conforme apresentado o conceito da métrica APF, ela consiste na ideia de que um sistema pode ter seu tamanho estimado de acordo com o número e complexidade dos requisitos funcionais que o compõem. Diante disso, tornaria inviável a utilização dessa técnica para medir o tamanho das aplicações, uma vez que os dois sistemas desenvolvidos realizam as mesmas funções, elas iriam possuir exatamente o mesmo tamanho, de acordo com a APF.

Outra métrica analisada foi o UCP, esta técnica poderia muito bem ser utilizada para estimar o tamanho dos softwares, visto que ela é um método de estimativa de projeto de software OO, que utiliza casos de uso para a realização dos cálculos. Todavia, Andrade (2004), pesquisando acerca deste método, pôde inferir que o UCP não é amplamente empregado devido a diversos empecilhos, os quais já foram citados no capítulo **2.6 Use Case Points** (p. 32).

O COCOMO baseia-se em 63 grandes projetos realizados durante um período de 15 anos, e suas fórmulas e teorias são comprovadas cientificamente, diferentemente do método UCP (CONTE, 1986). Além disso, o COCOMO foi a única métrica encontrada que possibilita medir todos os quesitos avaliados neste trabalho. A partir dessas pesquisas bibliográficas tornou-se possível determinar que o COCOMO seja a mais apropriada para realizar o estudo comparativo das aplicações desenvolvidas.

Dentro desta métrica escolhida existem três modelos distintos, o Básico, Intermediário e o Avançado, que são empregados conforme as características de sua aplicação e exigências. Diante desse cenário, foi estudado e determinado o modelo Básico para avaliar este projeto, visto que, será estimado o tamanho, tempo, esforço.

Posteriormente a análise dos requisitos funcionais da aplicação, o projeto deve ser classificado em um dos três modos de desenvolvimento existentes no modelo COCOMO, são eles, o modo Orgânico, o Semidestacado e o Embutido. O modo de desenvolvido determinado foi o Orgânico, pois conforme Júnior e Sanches (2000), neste modo, as esquipes são relativamente pequenas, num ambiente “familiar” e com projetos de até 50.000 linhas de código.

Em seguida, serão definidos os SGBDs, onde poderá ser analisada com mais detalhes no capítulo **4.1 Especificações do Sistema** (p. 40). Terminado esta etapa do processo, a aplicação será iniciada, primeiramente documentada por meio da linguagem de modelagem UML.

Para iniciar a avaliação do projeto de software, é necessário receber o tamanho estimado do sistema estimado em linhas de código. Para isto, será utilizado o plugin Metrics 1.3.6 na *Integrated Development Environment* (IDE) Eclipse Kepler. A partir disso é possível estimar o esforço requerido e o tempo de desenvolvimento do software.

Por fim, o protótipo iniciará seu desenvolvimento e ao mesmo tempo a sua avaliação, por meio da métrica COCOMO.

4 DESENVOLVIMENTO

Neste capítulo serão apresentadas as especificações da aplicação, quais foram os SGBDs escolhidos em cada paradigma, e quais motivos levaram para tal definição. Serão expostos também os diagramas da aplicação, utilizando para isto a UML para a modelagem do sistema. E por fim, será mostrado o resultado final da comparação entre os dois sistemas desenvolvidos, empregando o modelo COCOMO para a realização das estimativas.

4.1 Especificações do Sistema

A diversidade de SGBDs existentes no mercado atual é muito alto, cada um possui características próprias, vantagens e desvantagens em relação aos demais.

Portanto, é importante analisar qual a sua necessidade, e verificar quais possuem tais propriedades desejadas. Neste trabalho, não será necessário averiguar as peculiaridades de cada SGBD, pois o que será avaliado é o ganho no desenvolvimento de uma aplicação, utilizando para isto dois paradigmas de banco de dados díspares.

O SGBDR escolhido foi o PostgreSQL 9.0, pois ele possui uma vasta utilização e seu código é aberto. A versão 9.0 foi a determinada por possuir uma estabilidade maior em relação a outras mais recentes. Hoje em dia, o PostgreSQL é um dos SGBDs *open source* mais avançados, contando com recursos como: chaves estrangeiras, consultas complexas, controle de concorrência multi-visão, facilidade de acesso, entre outros (VILHANUEVA, 2007). Para a realização do ORM foi empregado o *framework* Hibernate.

Segundo a Documentação oficial (2004), o Hibernate é uma ferramenta de mapeamento objeto relacional desenvolvido para Java. A documentação também diz que o Hibernate além de cuidar do mapeamento de classes, oferece também recursos de consulta de dados e diversas facilidades, o que pode reduzir significativamente o tempo de desenvolvimento de uma aplicação.

O SGBDOO adotado para este trabalho foi o DB4O, versão 8.0. Conforme Galante et al. (2013), este banco de dados foi projetado para aplicações do tipo embarcada, cliente-servidor e desktop, possuindo modo nativo para Java e .Net. Uma das características deste banco que levou a sua escolha foi o fato dele ser também de código livre, igualmente ao PostgreSQL. Empresas como a Intel, Boing, Bosch, BMW, Hertz entre outras já utilizam este modelo de OO.

Neste presente trabalho já foi mencionado que a linguagem OO é a mais utilizada atualmente, devido suas vantagens e recursos também já citados acima. Desse modo, foi escolhida a linguagem de programação Java para o desenvolvimento das duas aplicações que serão empregadas no BDR e BDOO.

4.2 UML

A UML é a padronização das metodologias de desenvolvimento de sistemas baseados na OO, de forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com facilidade em se comunicar com outras aplicações, consistência, simples de ser atualizado e compreensível (PRESSMAN, 1995).

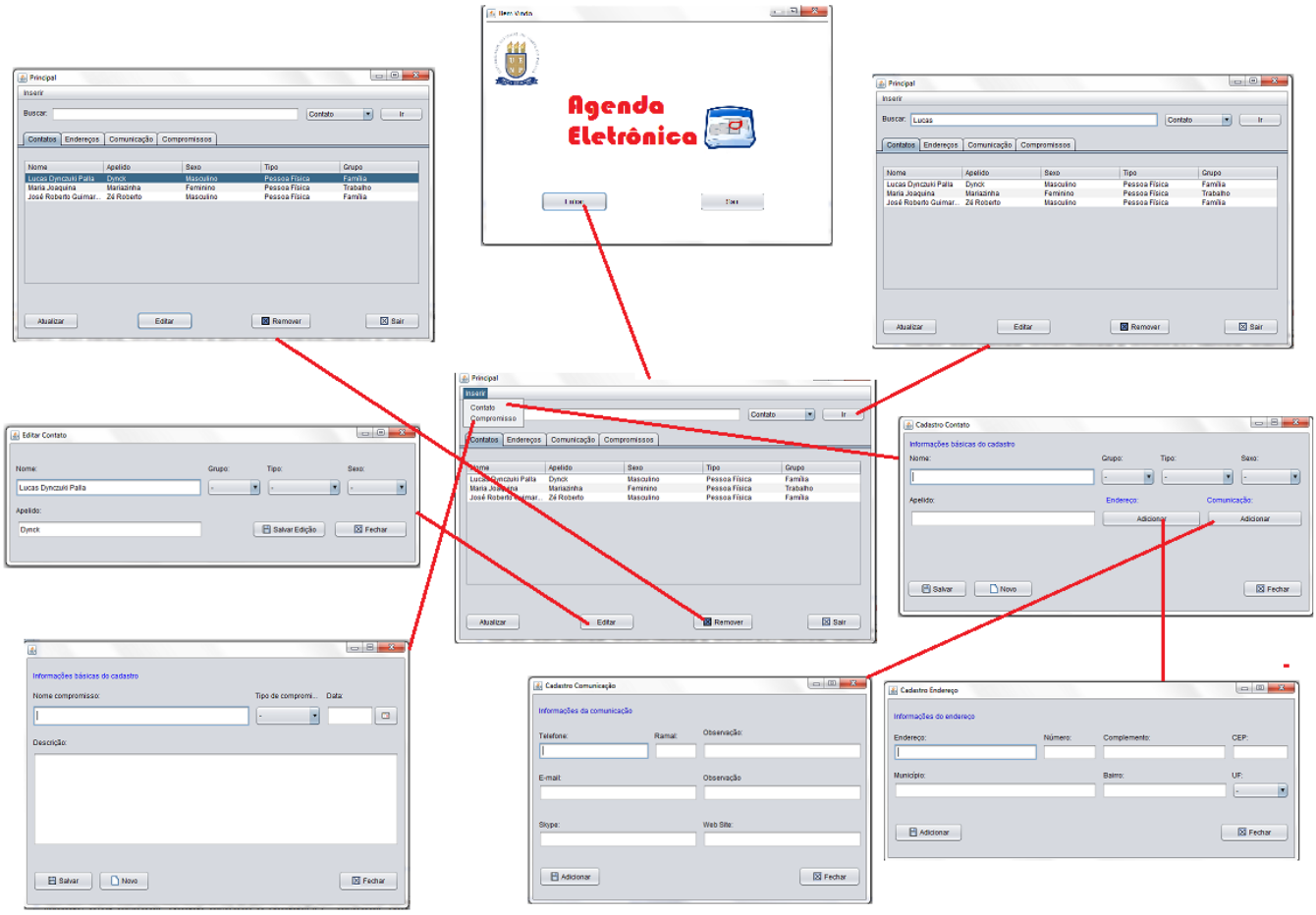
Segundo a Documentação oficial da UML, existem nove tipos de diagramas que são utilizados por essa padronização, são eles: diagrama de casos de uso, de classes, de objeto, de estado, de sequência, de comunicação, de atividade, de componente e o de execução.

Neste trabalho, foram projetados os diagramas de casos de uso, de classes, de atividade, de comunicação e o de sequência para apresentar a modelagem da aplicação desenvolvida. A modelagem e a documentação do protótipo encontram-se nos Apêndices A, B, C, D e E (p. 57).

4.3 Estudo Comparativo

O protótipo consiste em dois programas Java com interface gráfica, sendo que um emprega o BDR PostgreSQL e o outro o BDOO DB4O, ambos realizando as mesmas operações conforme os casos de uso descritos no Apêndice A (p. 57).

Figura 4 Diagrama de Telas do protótipo da Agenda Eletrônica



A Figura 4 demonstra o protótipo em uso e suas respectivas telas. Nas implementações dos softwares foi possível deparar-se com algumas particularidades de cada paradigma de banco de dados, tais peculiaridades serão abordadas a seguir.

No desenvolvimento do sistema em BDOO não foi necessário a utilização de ORM para o mapeamento das classes, visto que as duas utilizam a OO. Abaixo está sendo exibida a classe Contato, nela estão contidos todos os atributos desta classe (Figura 5).

Figura 5 Atributos da Classe Contato sem a necessidade do ORM

```
public class Contato {

    private String nomeContato;
    private String apelido;
    private Object grupo;
    private Object tipo;
    private Object sexo;
    private List<Endereco> endereco;
    private List<Comunicacao> comunicacao;
```

Diferentemente do que ocorre no BDOO, no desenvolvimento do protótipo empregando o PostgreSQL foi necessário a utilização do Hibernate para a realização do ORM, observe na Figura 6 o mapeamento dos atributos endereco e comunicacao.

Figura 6 Atributos da Classe Contato utilizando o Hibernate para o ORM

```
public class Contato {

    @Id
    @GeneratedValue (strategy = GenerationType.SEQUENCE, generator = "contato_seq")
    private int idcontato;
    private String nomeContato;
    private String apelido;
    private String grupo;
    private String tipo;
    private String sexo;
    @OneToMany(mappedBy = "contato")
    @LazyCollection(LazyCollectionOption.FALSE)
    @Cascade(CascadeType.ALL)
    private List<Endereco> endereco;
    @OneToMany(mappedBy = "contato")
    @LazyCollection(LazyCollectionOption.FALSE)
    @Cascade(CascadeType.ALL)
    private List<Comunicacao> comunicacao;
```

Outro ponto a se destacar na Figura 6 é a dispensabilidade de criar uma sequência para incrementar o atributo “id” a maneira em que vão sendo inseridos novos contatos ao banco de dados, já que o DB4O possui o mecanismo de incrementar o id de forma automática, e desta forma foi desnecessário criar o atributo “id” na classe Contato do software aplicado em BDOO.

O mapeamento foi realizado por meio de *annotations* e pelo *eXtensible Markup Language* (XML). O XML é uma linguagem de marcação responsável pelo mapeamento das classes no BDR. A seguir está apresentado o ORM do BDR (Figura 7).

Figura 7 ORM realizado pela linguagem de marcação XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.connection.username">postgres</property>
        <property name="hibernate.connection.password">lucas</property>
        <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/TCC</property
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="connection.pool_size">1</property>
        <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>

        <!-- classes mapeadas -->
        <mapping class = "entidades.Contato"/>
        <mapping class = "entidades.Compromisso"/>
        <mapping class = "entidades.Comunicacao"/>
        <mapping class = "entidades.Endereco"/>

    </session-factory>
</hibernate-configuration>
```

Conforme citado, no BDOO não é preciso fazer o ORM devido ao fato de que é desnecessário realizar a tradução de linguagens semelhantes, e também porque este paradigma trabalha com o conceito de objetos, armazenando-os utilizando o processo de serialização, persistindo o objeto como um todo. Neste trabalho também serão comparados as estruturas de armazenamento de ambos os bancos de dados empregados.

Além destas características mencionadas, foram encontradas outras diferenças, por exemplo, o método inserir dos dois protótipos é de certa forma simples de serem desenvolvidos, utilizando poucas linhas de código, como exposto a seguir (Figuras 8 e 9).

Figura 8 Método inserir da aplicação em BDOO

```
Banco bd = new Banco();
ObjectContainer db4o = bd.db();

public void inserir(T obj){
    db4o = bd.db();
    db4o.store(obj);
}
```

Figura 9 Método inserir da aplicação em BDR

```
private Session session;
private Transaction transacao;

public void inserir(T obj){
    session = HibernateUtil.getSession();
    transacao = session.beginTransaction();
    session.save(obj);
    transacao.commit();
}
```

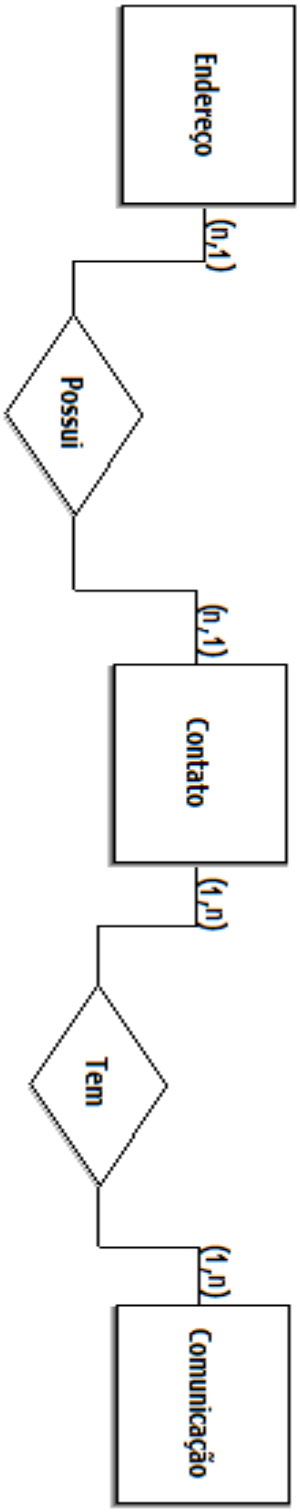
Todavia, a disparidade não se encontra no método inserir, mas sim na desnecessidade de se utilizar um método para editar as informações na aplicação em BDOO. Isto ocorre pelo fato de que neste banco de dados, o objeto é serializado e persistido, e quando é executado o comando de editar suas informações, o mesmo objeto passa pelo processo contrário da serialização, denominada desserialização. Desta forma é possível atualizar o objeto reutilizando o mesmo método inserir para armazenar seus novos dados, obtendo um grande ganho no desenvolvimento da aplicação em relação à quantidade total de linhas de código.

Conforme apresentado neste presente projeto, a SQL foi desenvolvida baseando-se na álgebra e cálculo relacional, desta forma, seus métodos de consulta são extremamente eficientes e eficazes, e pelo fato de que a estrutura em que os dados estão armazenados auxilia no processo de busca no BDR.

Este trabalho não teve por finalidade verificar a eficiência e velocidade da consulta de ambos os paradigmas, porém, de acordo com pesquisas realizadas acerca destes bancos e suas estruturas de armazenamento, foi possível apresentar uma breve comparação de como as informações estão elaboradas e como são os métodos de busca no BDR e no BDOO.

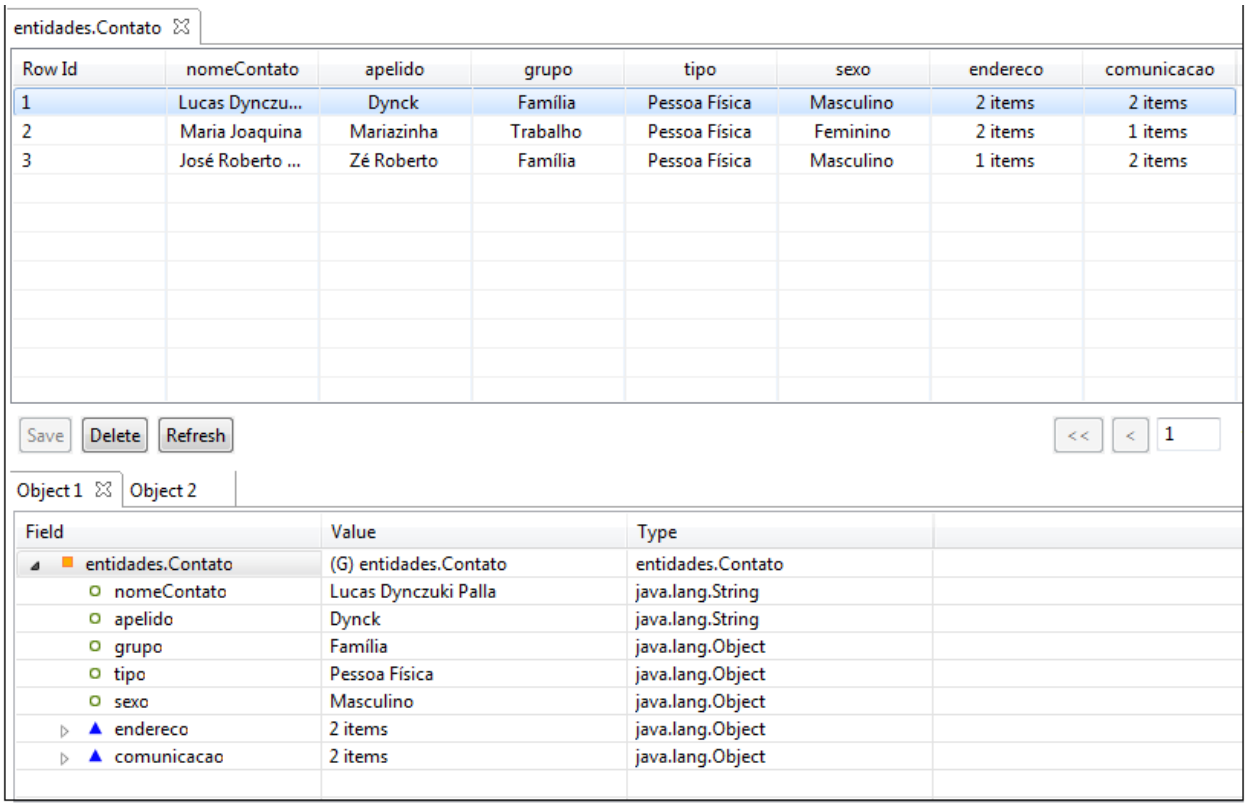
Na Figura 10 é possível verificar a “quebra” do objeto, mencionado no capítulo **2.1.4 Banco de Dados Relacional x Banco de Dados Orientado a Objetos** (p. 26), onde as informações do contato foram divididas em 3 tabelas, a tabela Contato, Endereco e Comunicacao, contendo as informações básicas do contato, seus endereços e suas formas de comunicação, respectivamente.

Figura 10 Estrutura de armazenamento dos dados no BDR



Além do fato de que a DML ser embasada em conceitos matemáticos e relacionais, este modo de estruturação (Figura 10) facilita o processo de consulta em um BDR, pois não é necessário percorrer todas as tabelas e informações para encontrar o que se deseja, diferentemente do que acontece nos BDOO. Na Figura 11 é exposta a elaboração das informações em um BDOO.

Figura 11 Estrutura de armazenamento dos dados no BDOO



Nos BDOO, os dados são persistidos como objetos, e são estruturados hierarquicamente, como é possível analisar na Figura 11, desta maneira, para realizar uma consulta de uma determinada informação é preciso explorar todo o objeto, o que torna um processo lento. Esta foi a principal desvantagem encontrada do BDOO em relação ao BDR. Contudo, este trabalho tem a finalidade de verificar o ganho no desenvolvimento, e conforme o objetivo real do projeto, não foi observado vantagens do BDR sobre o BDOO, estas verificações, comparações e conclusões sobre o ganho no desenvolvimento dos programas podem ser observadas abaixo.

4.3.1 Cálculo do Tamanho da Aplicação

Para que se possam iniciar as estimativas de esforço e tempo de desenvolvimento, é necessário receber o tamanho das duas aplicações desenvolvidas em quantidades de linhas de código. Por meio do plugin Metrics foi possível a coleta das informações requeridas, o protótipo desenvolvido em BDR possui 3048 linhas de código ao todo, sendo elas 3027 linhas em linguagem de programação Java e mais 21 linhas na linguagem XML. O XML é responsável pelo mapeamento das classes no BDR, na ausência desta linguagem de marcação não seria possível o relacionamento entre as classes e tabelas em um programa.

No BDOO a aplicação criada detém uma quantidade de linhas de código inferior, possuindo 2489 linhas de código em linguagem de programação Java, neste banco não é necessária a utilização do XML, visto que as informações são armazenadas em forma de objetos, como já apresentado neste presente trabalho.

De acordo com os dados colhidos, é notável a disparidade entre os softwares visando o requisito tamanho, a diferença entre eles é de 559 linhas de código, ou seja, a aplicação desenvolvida para BDOO possui 18,33% menos linhas de código em relação ao outro sistema.

4.3.2 Cálculo do Esforço

Após a etapa acima, poderá ser iniciada o cálculo das estimativas de esforço e tempo de desenvolvimento, empregando para isto a métrica COCOMO. Primeiramente será medido o requisito esforço da aplicação feita para BDR. Conforme a Fórmula 1, a letra ***S*** representa o tamanho do software expresso em milhares de linhas de código, assim sendo, **S** receberia o valor de 3,048. A variável a_i recebe o valor de 2,4 e b_i o valor de 1,05, pois de acordo com a Tabela 1 esses valores são referentes ao modo de desenvolvimento escolhido, no caso o Orgânico. Realizando este cálculo, foi gerado o resultado de 7.73 pessoas/mês.

Agora, será realizado o cálculo do esforço da aplicação desenvolvida para o BDOO. Conforme apresentado os valores das variáveis de a_i e b_i, será necessário

apenas saber o valor de ***S***, que no caso ***S*** adquire o valor de 2,489. Efetuando a somatório, o esforço deste software é de 6.25 pessoas/mês.

Segundo os números gerados pela métrica, ao empregar paradigmas de software compatíveis obtém a vantagem de se abdicar de um programador neste projeto em questão, desta forma o mesmo poderá ser realocado, acelerando o desenvolvimento de outro protótipo, resultando em um lucro indireto, uma vez que diminuindo o tempo de desenvolvimento das aplicações em uma empresa, a mesma será capaz aceitar novos projetos e assim gerando lucro direto.

4.3.3 Cálculo do Tempo de Desenvolvimento

Seguindo com as estimativas, neste momento serão realizados os cálculos para que se possa conhecer o prazo aproximado para a criação de ambas as aplicações. Conforme a fórmula de tempo de desenvolvimento apresentado na Fórmula 2, a variável ***E*** faz referência ao valor calculado dos esforços das aplicações. De acordo os valores da Tabela 2, as variáveis a_i e b_i, passam a valer a 2,5 e 0,38, respectivamente. A princípio será estimado o prazo do protótipo criado para BDR. Já conhecido os valores das variáveis, apenas será necessário fazer o cálculo. O resultado obtido nesta conta foi de 5.43 meses.

Da mesma maneira, será realizada a estimativa para a aplicação em BDOO, o produto gerado é de 5.01 meses para que o software seja desenvolvido.

De forma análoga ao resultado do esforço requerido, a aplicação desenvolvida para BDOO obteve um ganho em tempo de desenvolvimento em relação ao outro software. Esta vantagem convertida em dias têm o resultado de aproximadamente 12 dias úteis. Refletindo sobre estes dados, vamos supor que existam duas empresas fictícias, a empresa A e B. Digamos que estas organizações iniciem o desenvolvimento dessa agenda eletrônica, a empresa A decide optar por utilizar o BDR por ser tratar de um banco de dados tradicional e consagrado em todo mundo. Enquanto a empresa B utiliza paradigma de linguagem de programação de armazenamento de dados baseados na OO. Considerando que as duas empresas consigam criar suas agendas, a

empresa A levou 12 dias de trabalho a mais que a empresa B, desta forma a empresa B tomou a frente neste mercado de negócios.

4.3.4 Resultados Obtidos

Em seguida, serão apresentadas as comparações de acordo com os resultados alcançados por meio das equações da métrica COCOMO.

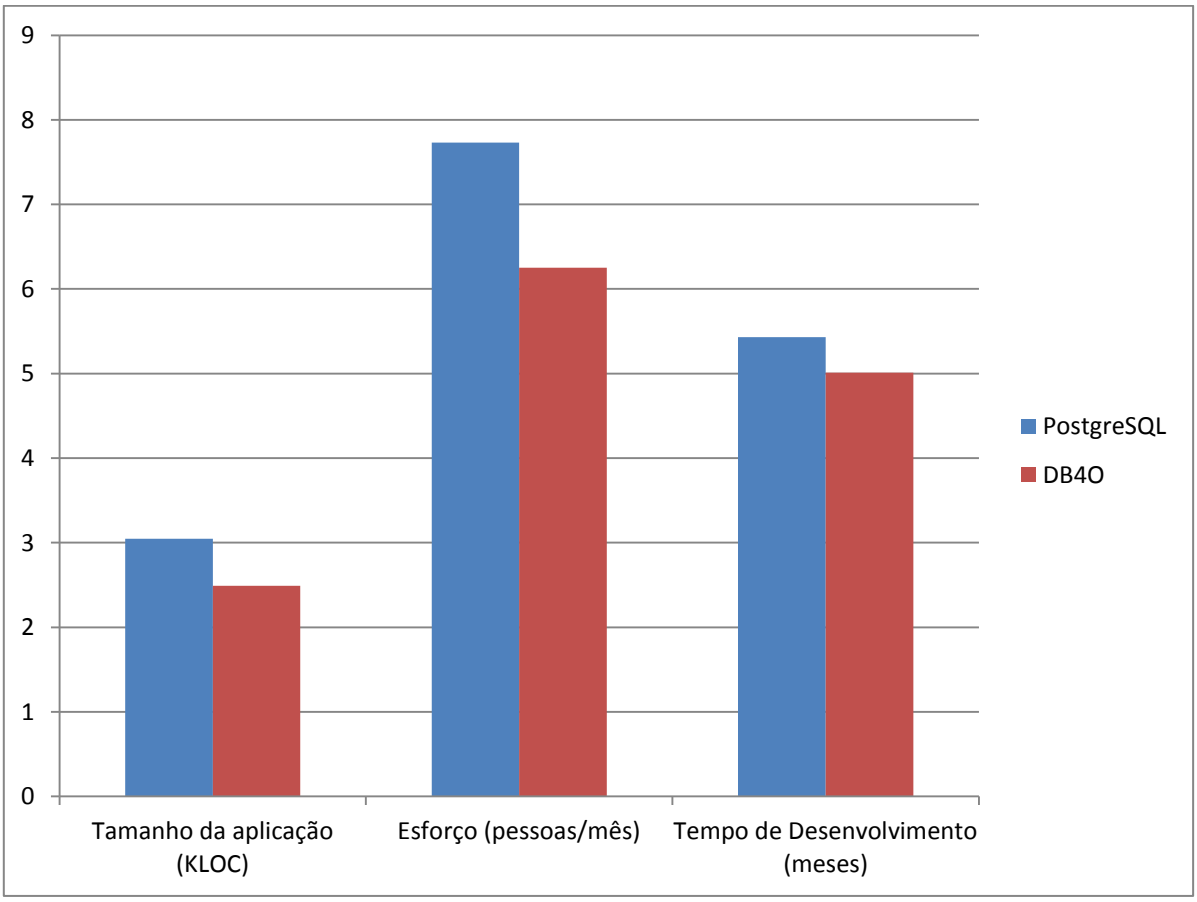
Tabela 3 Resultados do Estudo Comparativo

	PostgreSQL	DB4O	Diferença
Tamanho da Aplicação (KLOC)	3,048	2,489	559 (18,33%)
Esforço	7,73 pessoas/mês	6,25 pessoas/mês	1,48 pessoas/mês
Tempo de Desenvolvimento	5,43 meses	5,01 meses	0,42 meses

A Tabela 3 exibe os produtos das equações do COCOMO realizadas nas duas aplicações desenvolvidas. É notável a disparidade entre os sistemas, uma organização que optar por utilizar paradigmas de produção e armazenamento de dados compatíveis poderá destinar um programador em outro projeto, pois segundo os resultados obtidos, o software desenvolvido em BDOO teve uma diferença de 1,48 pessoas/mês trabalhando neste desenvolvimento, o que gera um lucro indireto, visto que realocando um profissional em outra tarefa, irá acelerar o outro serviço, e isso sucessivamente.

Além deste ganho, esta empresa que optou pelos modelos semelhantes, obteve um ganho de 0,42 meses em relação ao software desenvolvido para o PostgreSQL, número que transformado em dias apresenta um resultado de aproximadamente 12 dias de expediente, o que ocasiona mais lucro indireto para a organização, uma vez que agilizando um determinado desenvolvimento, a empresa poderá aceitar novos serviços e estará a frente no mercado de negócios.

Figura 12 Gráfico comparativo dos resultados obtidos



A Figura 12 apresenta de outra maneira os resultados atingidos por meio do estudo comparativo executado. O ganho no desenvolvimento da aplicação desenvolvida empregando o DB4O é visível, o seu tamanho, esforço e prazo para o desenvolvimento foram inferior ao sistema criado utilizando paradigmas distintos.

5 CONCLUSÃO

Espera-se que este trabalho contribua tanto aos leitores que estejam somente procurando informações sobre a diferença existente ao se empregar paradigmas distintos para o desenvolvimento de software, quanto aos que pretendem utilizar tais resultados obtidos em projetos futuros.

Após o desenvolvimento das aplicações, foi possível realizar análises em relação ao desenvolvimento dos protótipos. Uma das diferenças averiguadas é que além de não ser necessário utilizar o ORM, o método editar na aplicação para BDOO foi dispensado, devido ao fato de que a linguagem OO utiliza a serialização para armazenar as informações, com isso, o mesmo método inserir realizou a função de editar as informações na agenda eletrônica, diminuindo de forma efetiva a quantidade de linhas de código. Por outro lado, os mecanismos de busca no modelo relacional são mais eficientes, em consequência de que este paradigma foi baseado em fórmulas matemáticas.

Analisando os resultados comparativos, foi verificado que se uma determinada organização empregar o paradigma OO tanto na linguagem de programação, quanto no mecanismo de armazenamento de dados ela alcança algumas vantagens competitivas, tais como, o esforço será diminuído em 1.48 pessoas/mês e o tempo de desenvolvimento abaixou em 0.42 meses, aproximadamente 12 dias. É importante ressaltar que, com o aumento do número de linhas de código.

Portanto, apesar de que os BDR ainda serem predominantes neste mercado, é fato que o modelo OO constitui uma tecnologia eficiente e inovadora em termos de guardar suas informações. A vista disso, para a determinação da combinação dos paradigmas deve-se analisar quais as necessidades reais da empresa, caso a organização possui como objetivo a redução do esforço da equipe e o prazo de desenvolvimento, a escolha deve ser os modelos análogos. Entretanto se a meta é segurança das informações, velocidade na consulta de dados, um modelo banco de dados mais robusto e consagrado, o MR é o mais indicado para ser utilizado.

REFERÊNCIAS

AMBLER, Scott W. **Análise e projeto orientado a objeto: seu guia para desenvolver sistemas robustos com tecnologia de objetos**. Tradução Oswaldo Zanelli. Rio de Janeiro: Infobook, 1998.

ANDRADE, Edméia L. P. **Pontos de casos de uso e pontos de função na gestão de estimativa de tamanho de projetos de software orientados a objetos**. 2004. 143 f. Dissertação (Mestrado em Gestão do Conhecimento e Tecnologia da Informação) – Curso de Pós-graduação em Gestão do Conhecimento e Tecnologia da Informação, Universidade Católica de Brasília, Brasília.

ARIGOFLU, Ali. ***A methodology for cost estimation***. ACM - Software engineering notes. São Paulo, p. 96-105, vol 18, 1993.

BAUER, Christian; KING, Gavin. "**Hibernate Em Ação**", Ciência Moderna Ltda., 2005.

BRAGA, Antônio. **Análise de pontos por função***.* Rio de Janeiro: Infobook. 1996.

BOEHM, B. ***Software Engineering Economics*** Englewood Cliffs N.J.: Prentice-Hall Inc. 1981.

CARVALHO, Ariadne; CHIOSSI, Thelma; DRACH, Marcos. **Aplicabilidade de Métricas por Pontos de Função a Sistemas Baseados em Web.** Disponível em:<http://wer.inf.puc-rio.br/WERpapers/pdf_counter.lua?wer=WER06&file_name=carvalho.pdf >. Acesso em: 12 jun. 2013.

CARVALHO, Guilherme Cantuária de. **Sistema de Banco de Dados Orientado a Objetos.** Disponível em: <www.fatecsp.br/dti/tcc/tcc0002.pdf>. Acesso em: 04 set. 2013.

CONTE, S.D.; SHEN V.Y. ***Software Engineering Metrics and Models****.* Menlo Park, California: Benjamin/Cummings Publishing, 1985.

CORDEIRO, Marco Aurélio. **Métricas de Software.** Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=88>. Acesso em: 10 jun. 2013.

COSTA, Elisângela Rocha da. **BANCOS DE DADOS RELACIONAIS.** Disponível em: <http://www.fatecsp.br/dti/tcc/tcc0025.pdf>. Acesso em: 07 out. 2013.

DATE, C. J. **Introdução a Sistema de Banco de Dados.** São Paulo: Atlas, 2000.

DATE, C. J.. **INTRODUÇÃO A SISTEMAS DE BANCOS DE DADOS**. 8. ed. Rio de

Janeiro: Elsevier, 2003.

DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar.** Tradução Edson Furmankiewicz. 8ª. ed. São Paulo: Pearson, 2010. 1114 p.

DEKKERS, C. A. *Managing (the Size of) Your Projects*. ***A Project Management Look at Function Points*.** Quality Plus Technologies, Inc.

ELMASRI, Ramez. **Sistemas de Banco de Dados**. Ramez Elmasri e Shamkant B. Navathe; revisor técnico Luis Ricardo de Figueiredo. São Paulo: Addison Wesley, 2005.

ERIKSSON, Hans-Erik & PENKER, Magnus. **UML Toolkit**. Editora Wiley, 1998.

FREIRE, Herval. **Calculando Estimativas: o Método de Pontos de Caso de Uso.** Disponível em: <http://www.cnnt.com.br/files/usecasepoints.pdf>. Acesso em: 13 jun. 2013.

FERNANDES, Aguinaldo A. **Gerência de software através de métricas: garantindo a qualidade do projeto, processo e produto**. São Paulo: Atlas, 1995.

GALANTE, Alan Carvalho et al. **BANCO DE DADOS ORIENTADO A OBJETOS: UMA REALIDADE.** Disponível em: <http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf>. Acesso em: 28 out. 2013.

HAZAN, Cláudia. **Análise de Pontos de Função: Uma aplicação nas estimativas de tamanho de Projetos de Software**. Engenharia de Software, Rio de Janeiro, p.25-30, 01 jun. 2008.

HAZAN, Claudia; FUKS, Hugo; LUCENA, Carlos J. P. **Avaliação do tamanho funcional de ferramentas de e-learning**. 2005. 28 f. Monografia (Bacharelado em Ciência da Computação) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

HUMPHREY, W. S., ***Managing the Software Process***, Addison-Wesley Publs, 1989.

INTERNATIONAL FUNCTION POINT USER GROUP. **Análise de pontos por função**. [S.l.] : IFPUG, 1991. (Baseado na Release 3.4 do Manual de Práticas de Contagem do IFPUG)

ISHIKI, Ronaldo Tsuji. **Estudo Comparativo de Bancos de Dados Orientados a Objetos.** Disponível em: <http://www2.dc.uel.br/nourau/document/?down=153.>. Acesso em: 23 ago. 2013.

LÓPEZ, Pablo Ariel do Prado. **COCOMO II - Um modelo para estimativa de custos de Gerência de Projetos.** Disponível em: <http://www.sirc.unifra.br/artigos2005/artigo18.pdf>. Acesso em: 13 jun. 2013.

MACHADO, David Rodrigues. **Comparação entre Banco de Dados Relacionais e Banco de Dados Orientado a Objetos: Características e Implementação de um Protótipo.** Disponível em: <http://www.saofrancisco.edu.br/cursos/graduacao/producao_download.asp?arquivo=1716>. Acesso em: 14 jun. 2013.

Medeiros, E. **Desenvolvimento Software com UML 2.0: Definitivo**, Makron Books, 2006.

NASCIMENTO, Erinaldo Sanches. **Banco de Dados Relacional.** Disponível em: <http://erinaldosn.files.wordpress.com/2012/02/aula1-banco-de-dados-relacional1.pdf>. Acesso em: 10 out. 2013.

PAULK, M. C.;CURTIS, B.;CHRISSIS, M. B.; WEBER, C. V., ***Capability Maturity Model for Software****, version 1.1,* CMU/SEI-93-TR-24, fevereiro, 1993.

PETERS, James F.; PEDRYCZ, Witold. **Engenharia de software:** teoria e prática. Rio de Janeiro: Campus, 2001.

PINHEIRO, Daniel Ramon Silva et al. **Comparativo entre Banco de Dados Orientado a Objetos (BDOO) e Bancos de Dados Objeto Relacional (BDOR).** Disponível em: <http://sgclab.ic.uff.br/sgclab/index.php/publications/category/1-papers?download=2:study-of-database-reliability-and-performance>. Acesso em: 5 jun. 2013.

PRESSMAN, Roger S. ***Software engineering***: a practitioner's approach, 3ª ed. New York: McGraw-Hill, 1995.

PRESSMAN, Roger S. ***Software Engineering: A Practitioner's Approach***. 4ª ed. New York: McGraw-Hill, 1997.

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

RICARTE, Ivan Luiz Marques. **Sistemas de Bancos de Dados Orientados a Objetos.** Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/ricarte/apostilas/mc_sbdoo.pdf>. Acesso em: 22 out. 2013.

ROCHA, Cláudio M. **Explorando o relacionamento entre métricas baseadas em caso de uso e o número de casos de teste.** Curitiba: 2005. 81 f. Dissertação (Mestrado em Informática) – Setor de Ciências Exatas da Universidade Federal do Paraná, UFPR, 2005. Disponível em: <http://dspace.c3sl.ufpr.br/dspace/bitstream/1884/2580/1/dissfim.pdf>. Acesso em: 9 jun. 2012.

SANTANA, Otávio Gonçalves de. **Por que Java ?** Disponível em: <http://www.devmedia.com.br/por-que-java/20384>. Acesso em: 13 ago. 2013.

SEIBT, Patrícia Regina Ramos da Silva. **Ferramenta para Cálculo de Métricas de Software Orientado a Objetos.** Disponível em: <http://campeche.inf.furb.br/tccs/2001-II/2001-2patriciareginaramosdasilvaseibtvf.pdf>. Acesso em: 10 jun. 2013.

SETZE, V.; SILVA, F. S. **Banco de Dados. Aprenda o que são, melhore seu conhecimento, construa os seus**. São Paulo: Edgard Blücher, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.. **Sistema de Banco de Dados.** Tradução Daniel Vieira. 8ª ed. Rio de Janeiro: Elsevier, 2006. 781 p.

SOMMERVILLE, Ian. **Engenharia de software**. 6ª ed. São Paulo: Addison Weskey, 2003.

SOMMERVILLE, Ian. **Engenharia de Software**, 8ª ed. São Paulo, 2007.

SUDARSHAN, S.; SILBERSCHATZ, A.; KORTH, F. H. **Sistemas de Banco de Dados**. 3ª ed. São Paulo: Makron Books, 1999.

TEIXEIRA Júnior, Waine; SANCHES, Rosely. **Modelos de Estimativas de Custo de Software COCOMO & COCOMO II.** Disponível em: <http://www.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_RT_106.pdf>. Acesso em: 1 jun. 2013.

TELES, Jerônimo et al. **Banco de Dados Orientado a Objetos.** Disponível em: <http://homes.dcc.ufba.br/~jteles/artigoODBMS.pdf>. Acesso em: 20 out. 2013.

TRINDADE, André L. P.; PESSOA, Marcelo S. P.; SPINOLA, Mauro M. **COCOMO II: uma compilação de informações sobre a nova métrica**. In: CONGRESSO INTERNACIONAL DE ENGENHARIA INFORMÁTICA DA UNIVERSIDADE DE BUENOS AIRES, 5, 1999, Buenos Aires, Argentina.

VIESI, Kaio Thomeo. **A Ferramenta Hibernate: Desenvolvimento de uma Aplicação em Java para Persistência de Dados sem SQL.** Disponível em: <http://fotos.fatectq.edu.br/a/5906.pdf>. Acesso em: 8 jun. 2013.

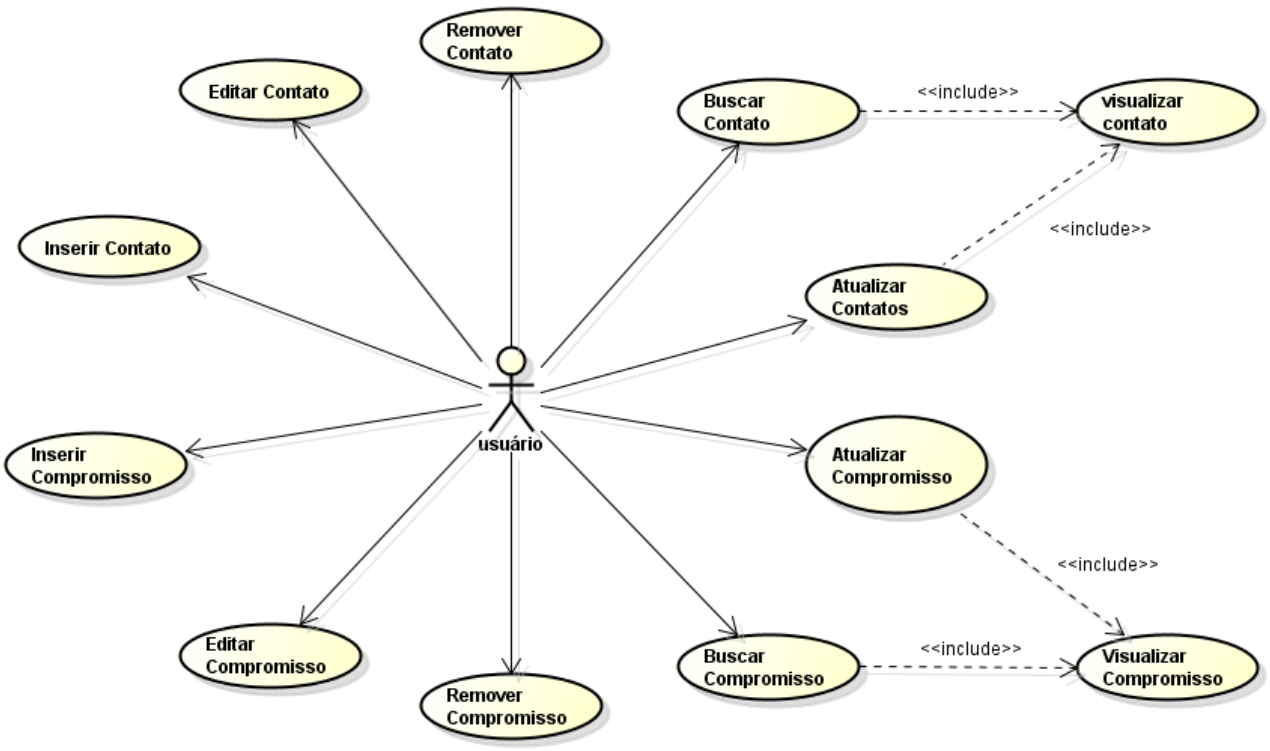
VILHANUEVA, Marcos Pinheiro. **Persistência em banco de dados relacionais usando linguagens orientada a objetos.** Disponível em: <http://www.espweb.uem.br/monografias/2005/Monografia2007.pdf>. Acesso em: 9 jun. 2013.

Apêndice A – Diagrama de Casos de Uso

O diagrama de casos de uso é uma técnica utilizada para descrever e especificar os requisitos funcionais de um sistema (ERIKSSON; PENKER, 1998). Esta modelagem é representada por atores e os casos de uso. Os atores desempenham o papel de uma entidade externa ao sistema, como um usuário, ou outro sistema que interage com a aplicação modelada. Os casos de uso representam uma sequência de ações executadas pelo sistema.

A seguir será apresentado o diagrama de casos de uso do sistema desenvolvido, contendo a documentação de todos os requisitos funcionais que são executados pela aplicação.

Diagrama de Casos de Uso da agenda eletrônica desenvolvida



1 Inserir Contato

1.1 Breve Descrição

Este caso de uso ocorre quando o usuário quer cadastrar um novo contato em sua agenda eletrônica.

1.2 Ator

Usuário.

1.3 Pré-Requisito

Não existe pré-requisito neste caso de uso.

1.4 Fluxo de Eventos

1.4.1 Fluxo Principal

1. O usuário clica no item “Contato” do menu na página principal.
2. O sistema exibe os campos para serem preenchidos.
3. O ator deve preencher os campos de cadastro e clicar no botão “Salvar”.
4. Depois do cadastro o usuário volta à página principal.

1.4.2 Fluxo Alternativo

1. O ator clica no botão “Salvar” sem preencher todos os campos de cadastro.
2. O sistema exibe uma mensagem indicando qual campo não foi informado.
3. O usuário retorna na página de cadastro do contato.

1.5 Pós-Condição

O contato está cadastrado.

2 Editar Contato

2.1 Breve Descrição

Neste caso de uso o usuário quer editar as informações do contato desejado.

2.2 Ator

Usuário.

2.3 Pré-Requisito

O contato deve estar previamente cadastrado.

2.4 Fluxo de Eventos

2.4.1 Fluxo Principal

1. O usuário escolhe a aba “Contatos” e seleciona a linha do contato cadastrado que deseja editar.
2. Caso o ator queira editar os dados do endereço ou da comunicação do contato selecionado, o mesmo deve trocar de aba, na qual contém as informações desejadas.
3. Escolhido o contato, o ator deve clicar no botão “Editar”.
4. O sistema retorna os campos para serem editados.
5. O usuário preenche os campos e clica no botão “Salvar Edição”.
6. O ator retorna para a página principal.

2.4.2 Fluxo Alternativo

1. O usuário não selecionado nenhum contato cadastrado e clica no botão “Editar”.
2. O sistema retorna uma mensagem indicando que nenhuma linha foi selecionada.
3. O ator retorna para a página principal.

2.5 Pós-Condição

O contato cadastrado é editado.

3 Remover Contato

3.1 Breve Descrição

Este caso de uso acontece quando o ator pretende excluir algum contato já cadastrado.

3.2 Ator

Usuário.

3.3 Pré-Requisito

O contato que o usuário deseja remover deve estar cadastrado.

3.4 Fluxo de Eventos

3.4.1 Fluxo Principal

1. O usuário escolhe a aba “Contatos” na página principal.
2. O mesmo seleciona contato pretendido.
3. O ator clica no botão “Remover”.

3.4.2 Fluxo Alternativo

1. O ator clica no botão “Remover” sem que esteja selecionado nenhum contato.
2. A aplicação exibe um aviso comunicando que nenhuma linha foi selecionada.

3.5 Pós-Condição

O contato foi removido.

4 Buscar Contato

4.1 Breve Descrição

Neste caso de uso, o ator deseja buscar um determinado nome Banco de Dados.

4.2 Ator

Usuário.

4.3 Pré-Requisito

Não há pré-requisito neste caso de uso.

4.4 Fluxo de Eventos

4.4.1 Fluxo Principal

1. Na página principal, o usuário deve escolher a opção “Contato” no ComboBox.
2. O usuário digita o nome pretendido para buscar no Banco de Dados.
3. O ator clica no botão “Buscar”.
4. O sistema retorna os contatos encontrados que condizem com o nome desejado.

4.4.2 Fluxo Alternativo

O sistema não encontra nenhum contato referente ao nome digitado e retorna uma tabela vazia, na aba “Contatos”.

4.5 Pós-Condição

Caso existe no Banco de Dados, o sistema apresenta os contatos identificados condizentes ao nome informado pelo usuário.

5 Atualizar Contatos

5.1 Breve Descrição

Após cadastrar, editar ou remover um contato, o usuário poderá desejar ver quais são as informações atuais existentes no Banco de Dados.

5.2 Ator

Usuário.

5.3 Pré-Requisito

Este caso de uso não possui pré-requisito.

5.4 Fluxo de Eventos

5.4.1 Fluxo Principal

1. Na página principal, o usuário escolhe a aba “Contatos”.
2. O ator clica no botão “Atualizar”.
3. O sistema exibe na tabela, as informações atualizadas contidas no Banco de Dados.

5.4.2 Fluxo Alternativo

1. O ator não seleciona a aba correta, com isso não poderá atualizar os contatos.

5.5 Pós-Condição

Os contatos cadastrados são expostos na tabela em que está situada na aba “Contatos”.

6 Inserir Compromisso

6.1 Breve Descrição

Este caso de uso ocorre quando o usuário deseja inserir um novo compromisso.

6.2 Ator

Usuário.

6.3 Pré-Requisito

Não existe pré-requisito.

6.4 Fluxo de Eventos

6.4.1 Fluxo Principal

1. O usuário escolhe a opção “Contato” no menu presente na página principal.
2. Os campos para serem preenchidos são exibidos pelo sistema.
3. O ator insere os dados e clica no botão “Salvar”.
4. O usuário é retornado para a página anterior.

6.4.2 Fluxo Alternativo

1. O ator clica no botão “Salvar” sem inserir as informações em todos os campos de cadastro.
2. É exibido um aviso informando qual campo não foi preenchido.
3. O usuário retorna à página de cadastro de compromisso.

6.5 Pós-Condição

Um novo compromisso é inserido na agenda eletrônica.

7 Editar Compromisso

7.1 Breve Descrição

Caso o ator deseje editar as informações de um determinado compromisso cadastrado.

7.2 Ator

Usuário.

7.3 Pré-Requisito

Para atualizar os dados de algum compromisso, o mesmo deve estar previamente cadastrado.

7.4 Fluxo de Eventos

7.4.1 Fluxo Principal

1. Primeiramente, o usuário deverá escolher a aba “Compromissos”.
2. Após o passo anterior, o ator escolhe a linha em que contém o compromisso que pretende atualizar.
3. O usuário clica no botão “Editar”.
4. O sistema expõe os campos para a edição do compromisso selecionado.
5. O usuário preenche os campos e clica no botão “Salvar Edição”.
6. O ator é retornado à página principal.

7.4.2 Fluxo Alternativo

1. O usuário clica no botão “Editar”, porém não seleciona nenhum compromisso.
2. É retornada uma mensagem avisando que nenhuma linha foi selecionada.
3. O ator volta á página anterior.

7.5 Pós-Condição

As informações do compromisso escolhido foram atualizadas.

8 Remover Compromisso

8.1 Breve Descrição

Neste caso de uso, o ator deseja excluir algum compromisso existente na agenda eletrônica.

8.2 Ator

Usuário.

8.3 Pré-Requisito

Para que este caso de uso ocorre, é necessário que exista algum compromisso inserido no Banco de Dados.

8.4 Fluxo de Eventos

8.4.1 Fluxo Principal

1. O ator deve selecionar a aba “Compromissos”.
2. Escolhida a aba, o usuário deverá apontar qual compromisso deseja remover de sua agenda, clicando na linha em que contém as informações do compromisso selecionado.
3. O usuário clica no botão “Remover”.

8.4.2 Fluxo Alternativo

1. O usuário deseja remover um determinado compromisso existente na agenda eletrônica, mas não seleciona nenhuma linha e clica no botão “Remover”.
2. O sistema apresenta a mensagem que não foi selecionado nenhuma linha.

8.5 Pós-Condição

O compromisso escolhido foi removido da agenda eletrônica.

9 Buscar Compromisso

9.1 Breve Descrição

Este caso de uso ocorre quando o ator quer procurar o nome de algum compromisso existente no Banco de Dados.

9.2 Ator

Usuário.

9.3 Pré-Requisito

Não há pré-requisito neste caso de uso.

9.4 Fluxo de Eventos

9.4.1 Fluxo Principal

1. Na página principal, o usuário escolhe a opção “Compromisso” no ComboBox.
2. O ator digita o nome que pretende buscar na caixa de texto.
3. Posteriormente, ele clica no botão “Buscar”.
4. Caso existam compromissos que correspondem ao nome digitado, o sistema exibe na tabela que está situada na aba “Compromissos”.

9.4.2 Fluxo Alternativo

O sistema não localiza nenhum compromisso referente ao nome digitado e retorna uma tabela vazia, na aba “Compromissos”.

9.5 Pós-Condição

Desde que o sistema encontre pelo menos um compromisso condizente ao nome informado pelo usuário, é exibido na tabela que está localizado na aba “Compromissos”, caso contrário, a tabela estará vazia.

10 Atualizar Compromissos

10.1 Breve Descrição

Este caso de uso acontece quando o usuário quer analisar as informações atuais de sua agenda eletrônica.

10.2 Ator

Usuário.

10.3 Pré-Requisito

Neste caso de uso não existe pré-requisito.

10.4 Fluxo de Eventos

10.4.1 Fluxo Principal

1. O ator seleciona a aba “Compromissos”.
2. Após o passo anterior, o usuário clica no botão “Atualizar”.
3. O sistema exibe as informações atualizadas que estão contidas no Banco de Dados.

10.4.2 Fluxo Alternativo

1. O usuário escolhe a aba errada, com isso não poderá atualizar os compromissos.

10.5 Pós-Condição

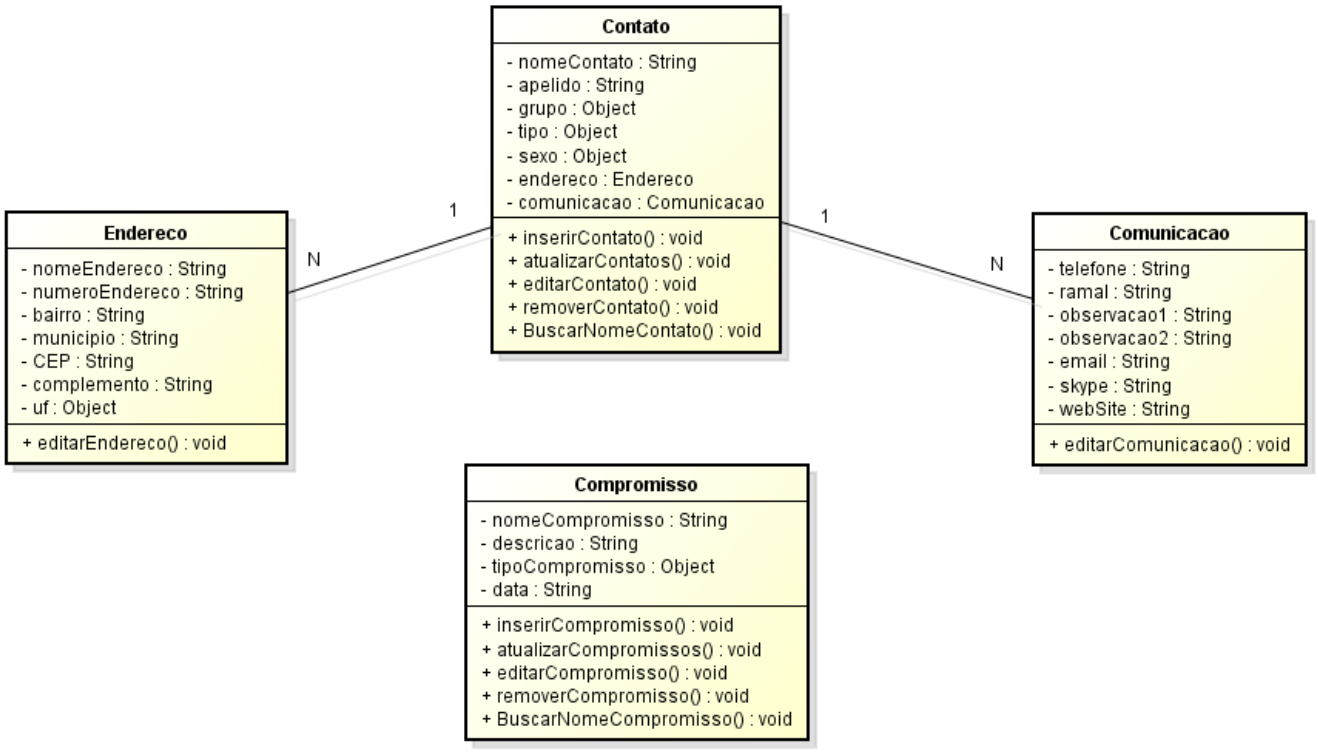
O sistema exibe as informações atualizadas dos compromissos.

Apêndice B – Diagrama de Classes

Segundo Eriksson e Penker (1998), este diagrama é de suma importância para o processo de modelagem, pois ele demonstra a estrutura estática das classes de uma aplicação, onde estas representam os atributos e os comportamentos que serão gerenciados pelo software.

Neste diagrama, as classes podem se relacionar com outros, por meio de algumas maneiras, tais como, dependência, associação, especialização ou em pacotes.

Diagrama de Classes da aplicação desenvolvida



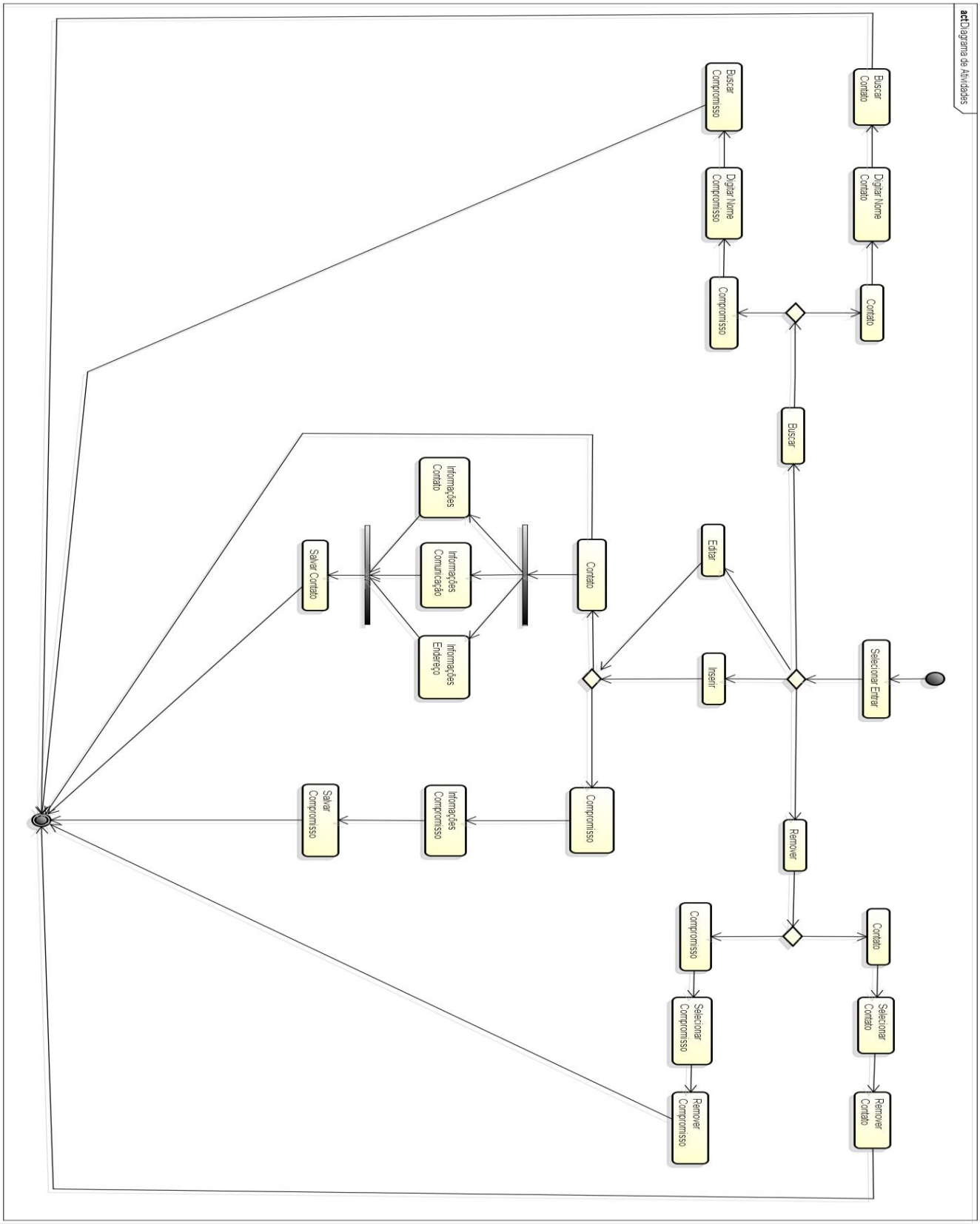
A Figura 5 representa o diagrama de classes da agenda eletrônica criada, nesta imagem é possível verificar o relacionamento entre as classes Contato e Endereco, por meio de associação, da qual possui a cardinalidade de 1:N, respectivamente. Outra associação visível é entre as classes Contato e Comunicacao, possuindo também a cardinalidade de 1:N, respectivamente.

Apêndice C – Diagrama de Atividade

Conforme Sommerville (2007), a ênfase do diagrama de atividade é capturar as ações e seus resultados, focando no trabalho executado na implementação de um método. Este diagrama exibe o fluxo sequencial das atividades do sistema, ele é comumente empregado para demonstrar as atividades executadas por uma operação específica do sistema.

As alterações de estado no diagrama de atividade ocorrem quando uma ação é executada. Neste diagrama também é apresentado as decisões e condições que ocorrem durante a decorrer do sistema e execuções em paralelos.

Diagrama de Atividade da protótipo desenvolvido



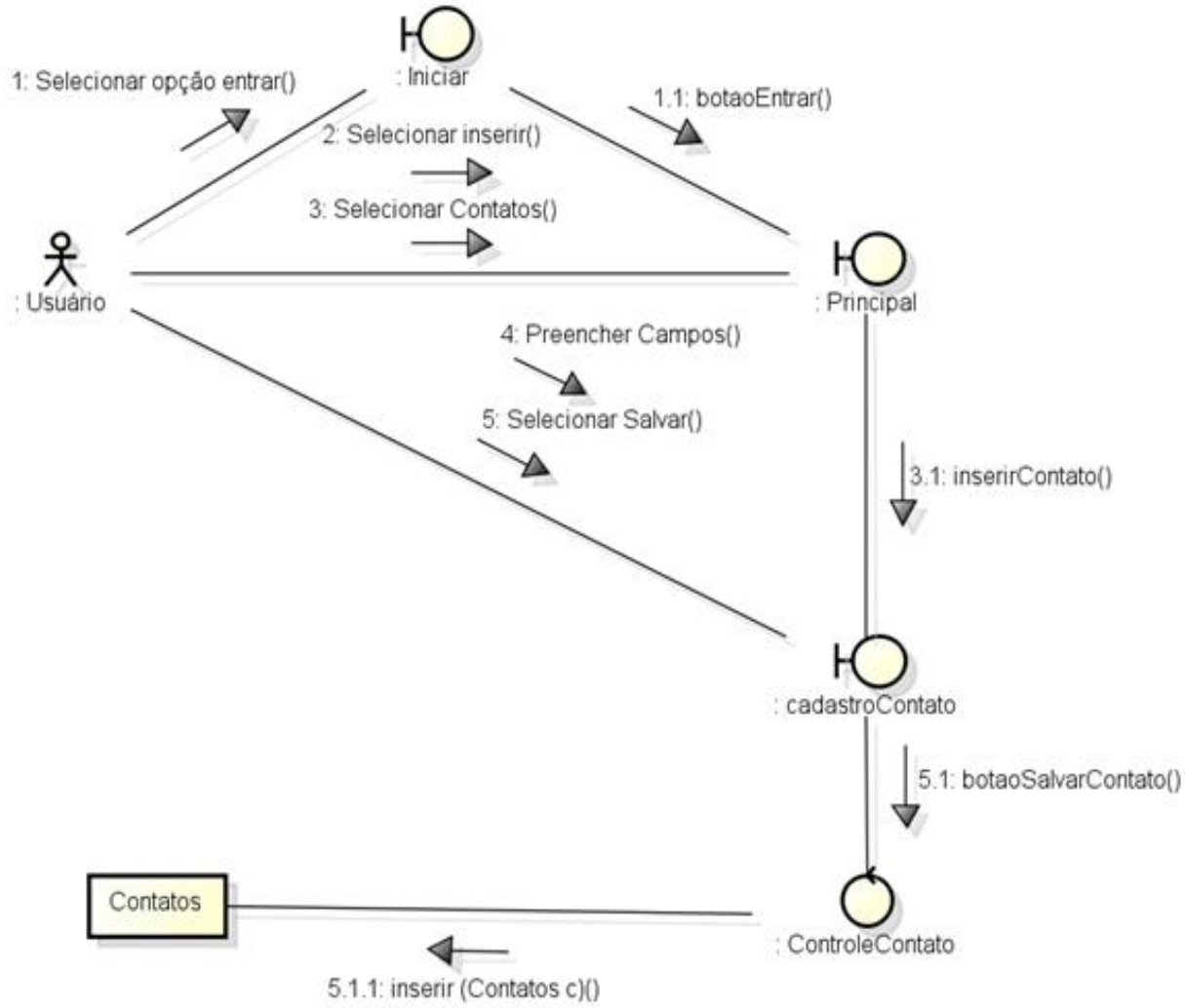
Apêndice D – Diagrama de Comunicação

O diagrama de comunicação era conhecido como diagrama de colaboração até a versão 1.5 da UML, tendo seu nome modificado para o nome atual a partir da versão 2.0 (SOMMERVILLE, 2007). Este diagrama é um complemento do diagrama de sequência, que veremos posteriormente.

Nesta modelagem, as informações são as mesmas apresentadas no diagrama de sequência, entretanto esses dois esquemas possuem um enfoque diferente, visto que o diagrama de comunicação concentra-se em como os objetos estão vinculados e quais mensagens trocam entre si durante o processo. Abaixo serão expostos os diagramas de comunicação de cada requisito funcional do sistema.

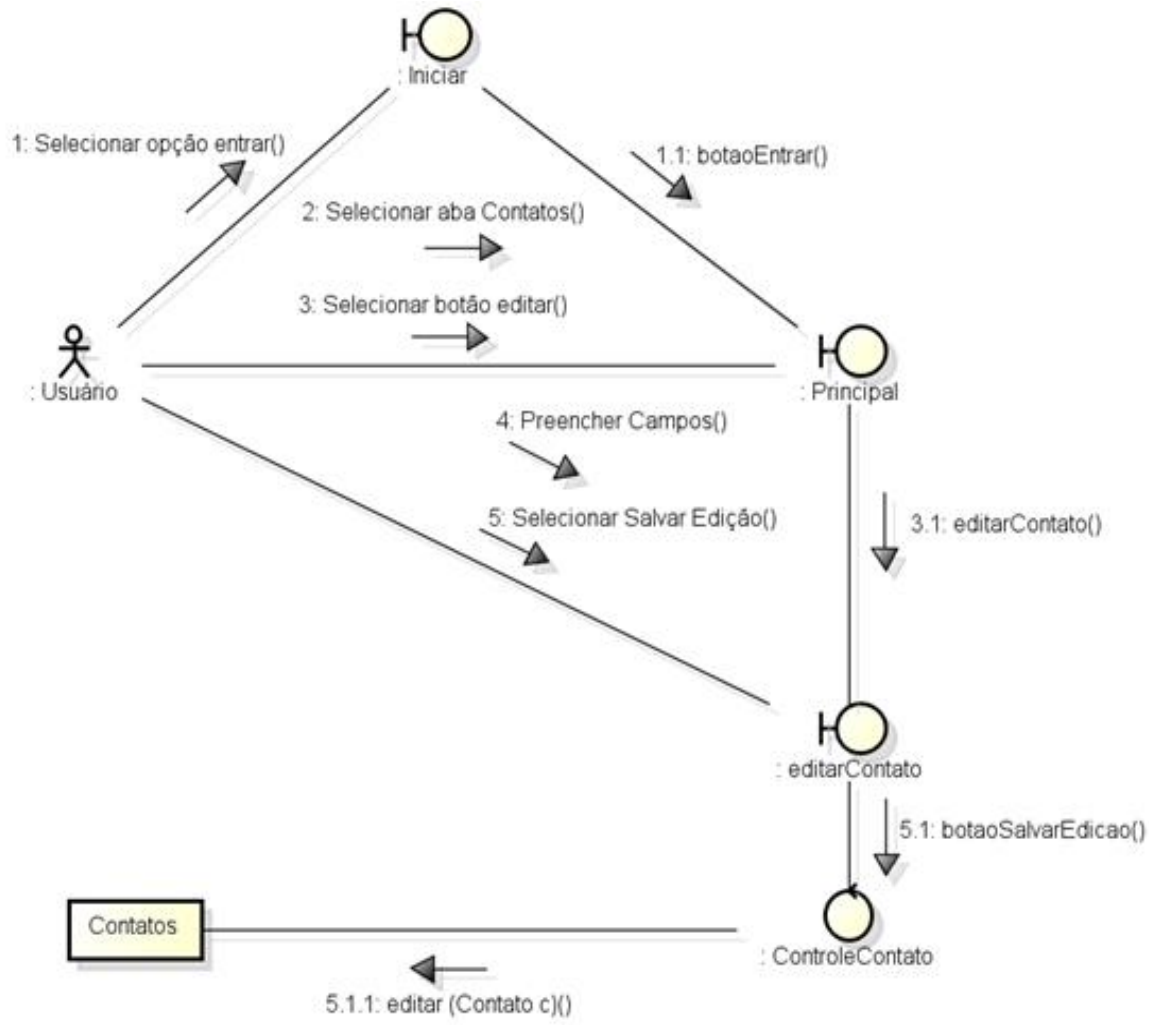
1 Inserir Contato

Diagrama de Comunicação do Caso de Uso Inserir Contato



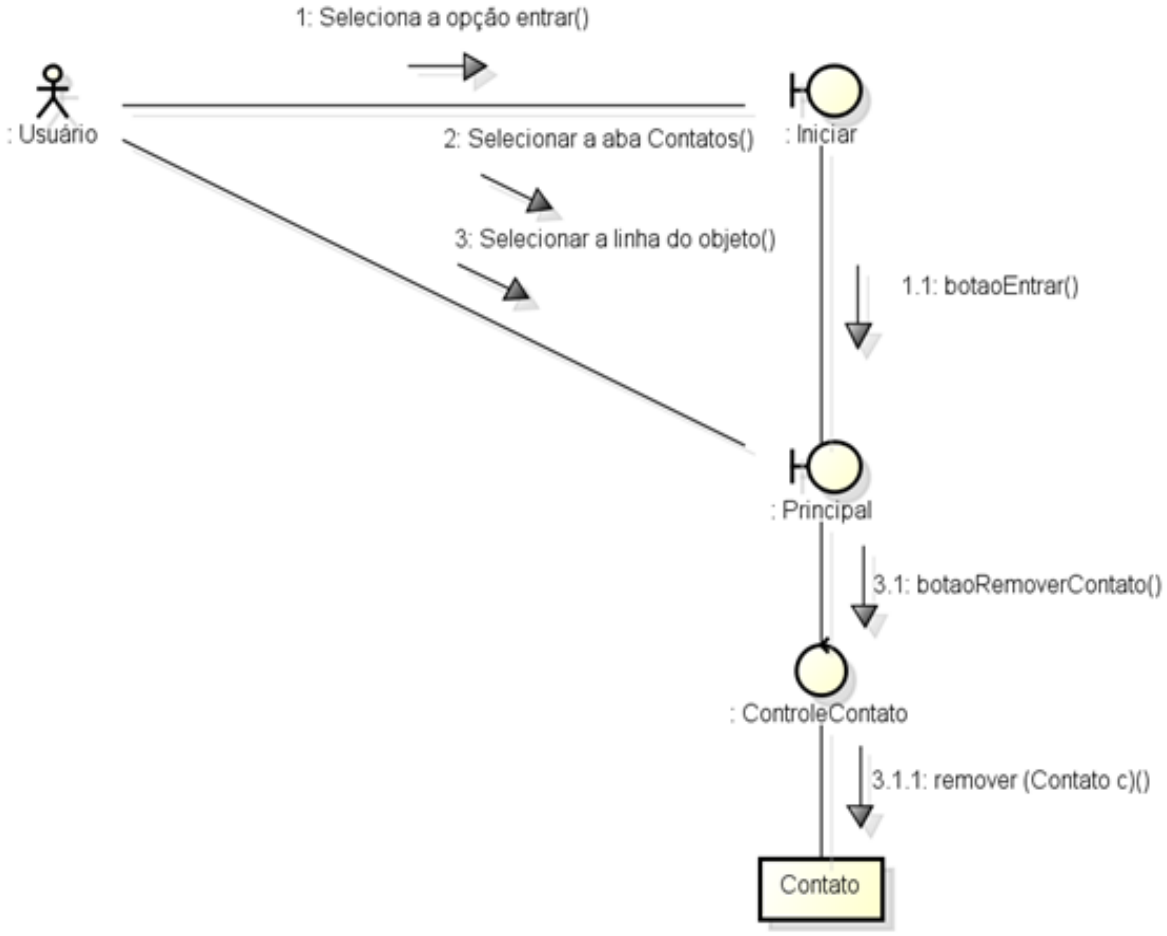
2 Editar Contato

Diagrama de Comunicação do Caso de Uso Editar Contato



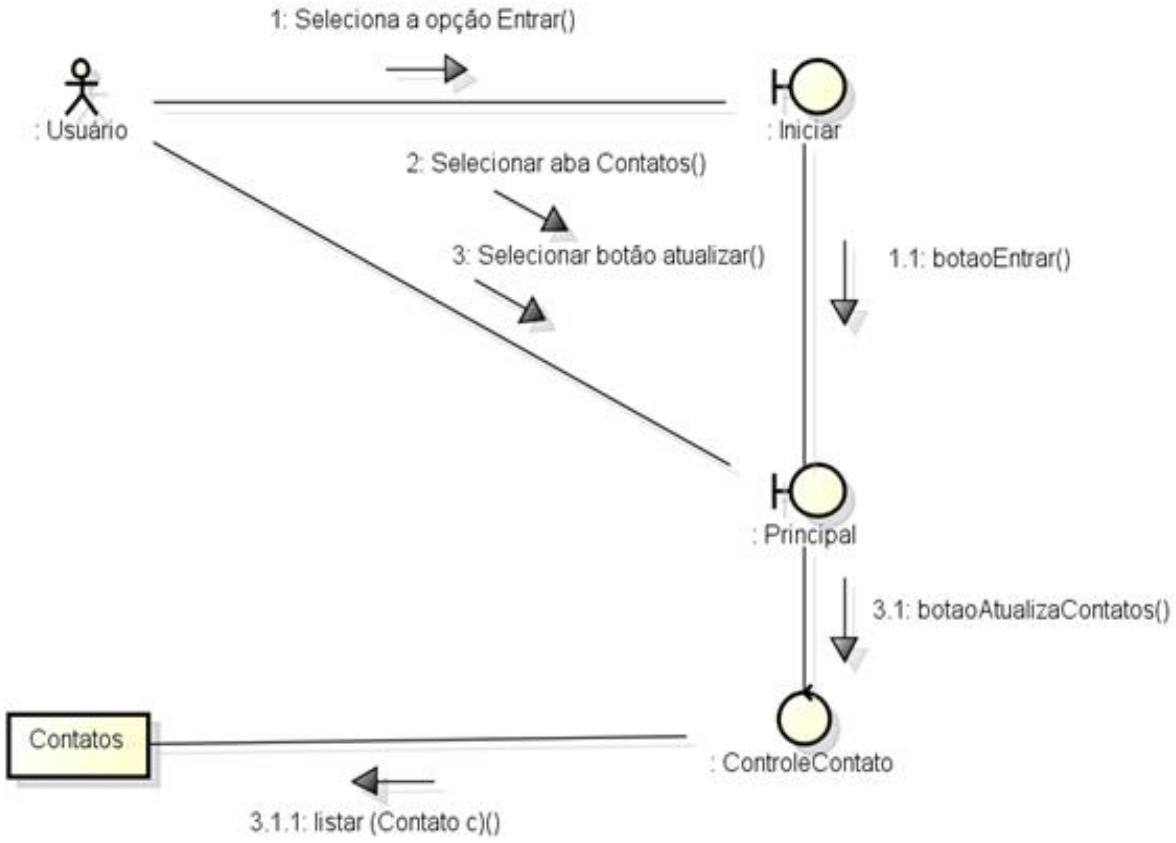
3 Remover Contato

Diagrama de Comunicação do Caso de Uso Remover Contato



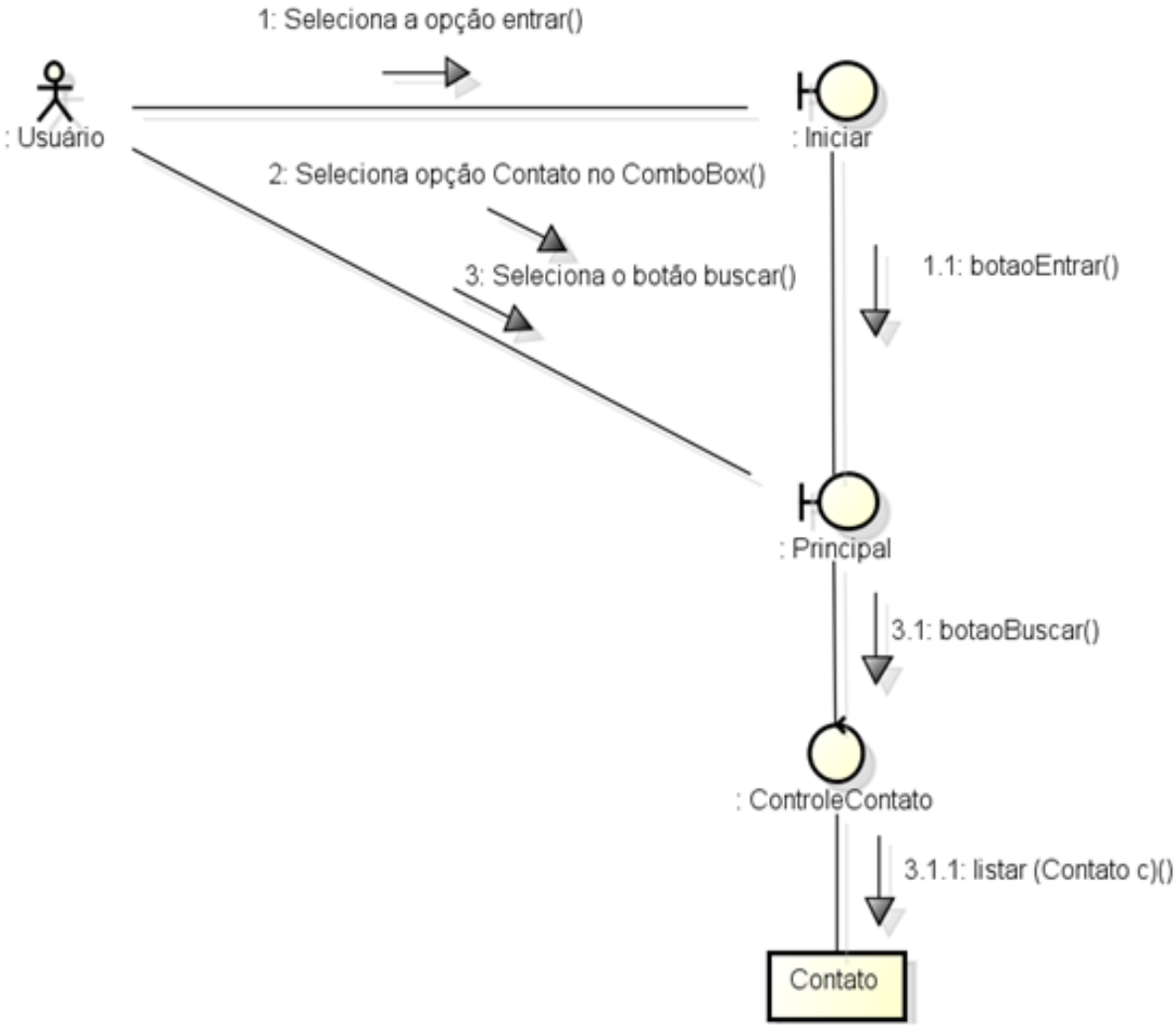
4 Atualizar Contatos

Diagrama de Comunicação do Caso de Uso Atualizar Contatos



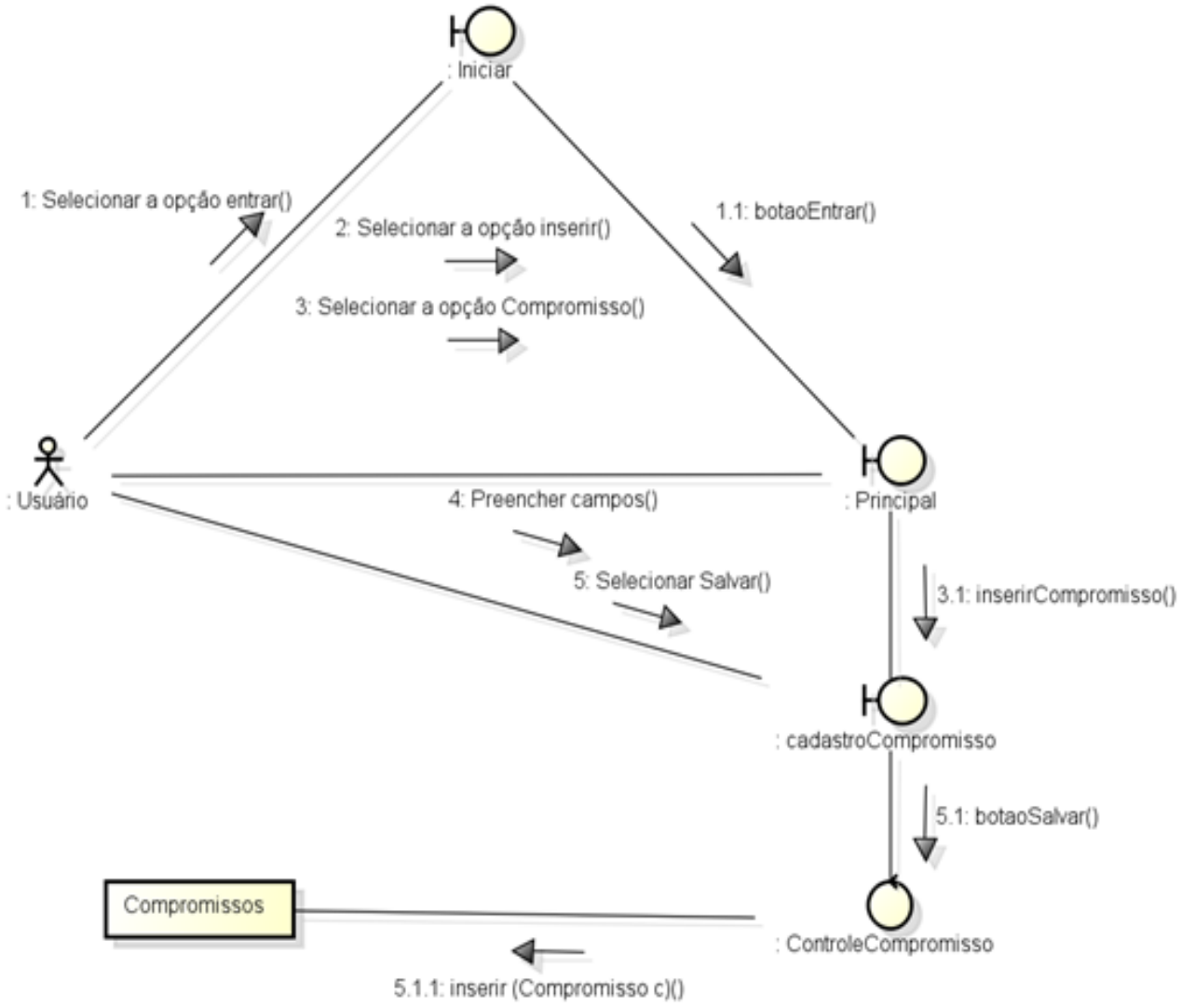
5 Buscar Nome Contato

Diagrama de Comunicação do Caso de Uso Buscar Nome Contato



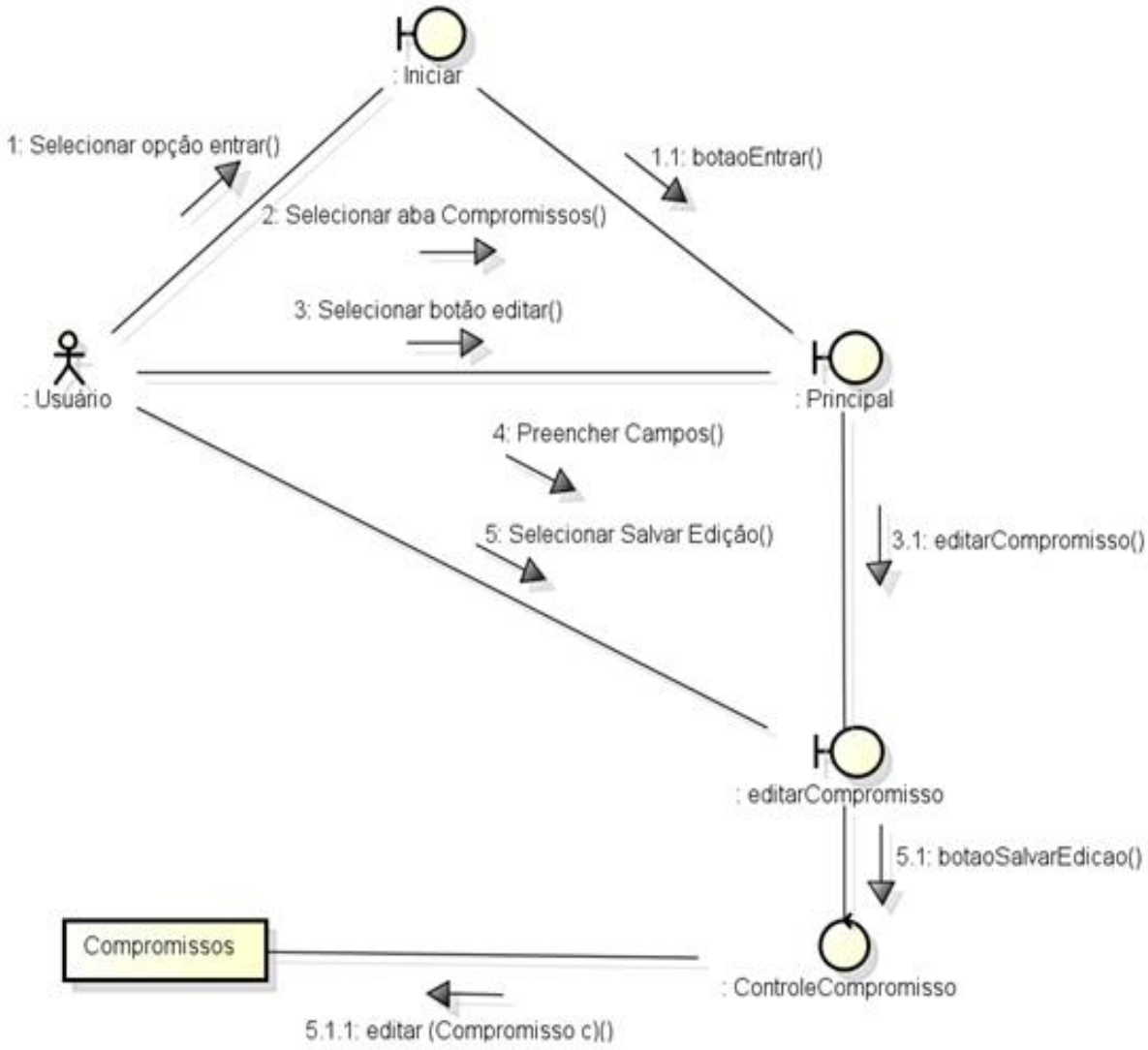
6 Inserir Compromisso

Diagrama de Comunicação do Caso de Uso Inserir Compromisso



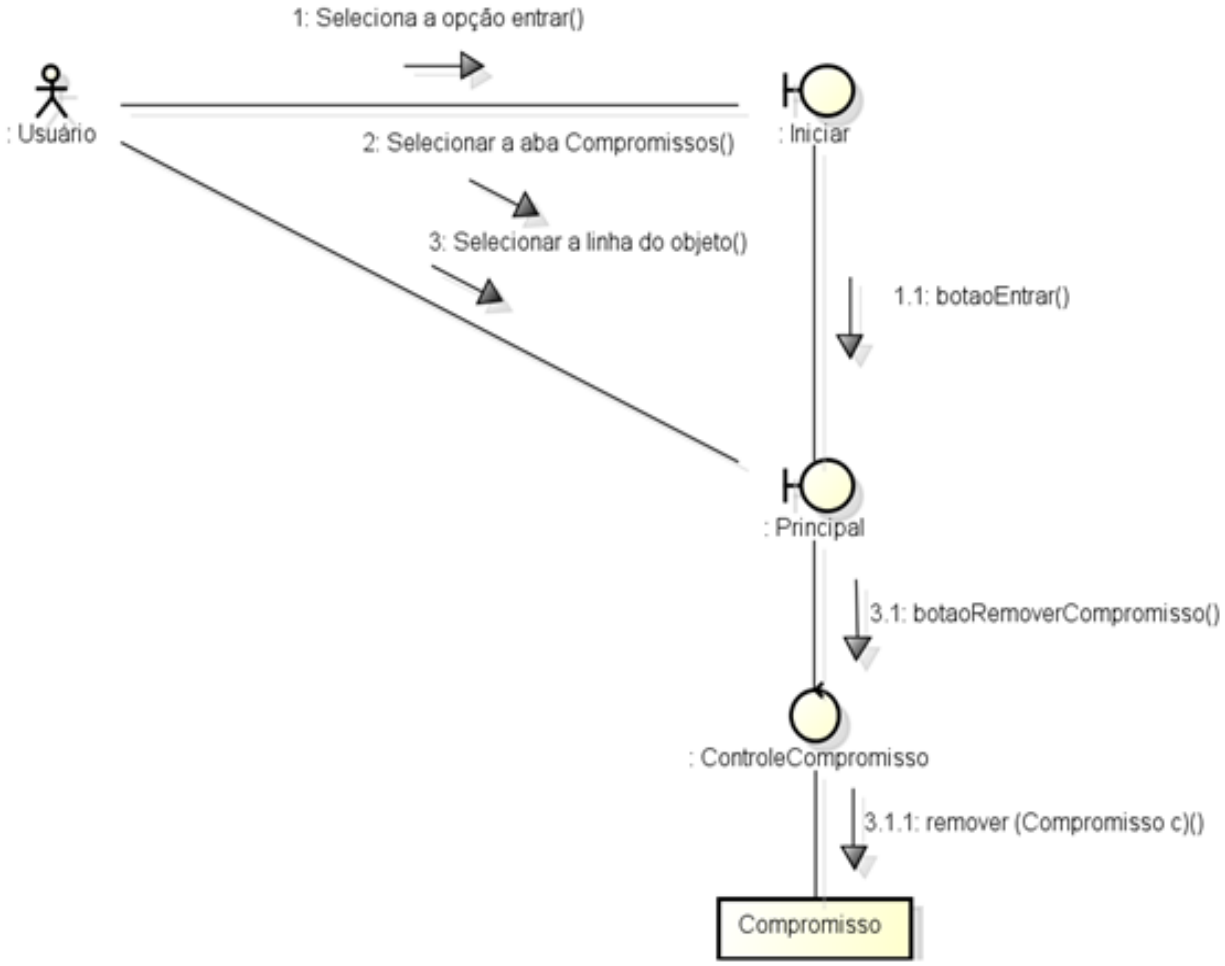
7 Editar Compromissos

Diagrama de Comunicação do Caso de Uso Editar Compromissos



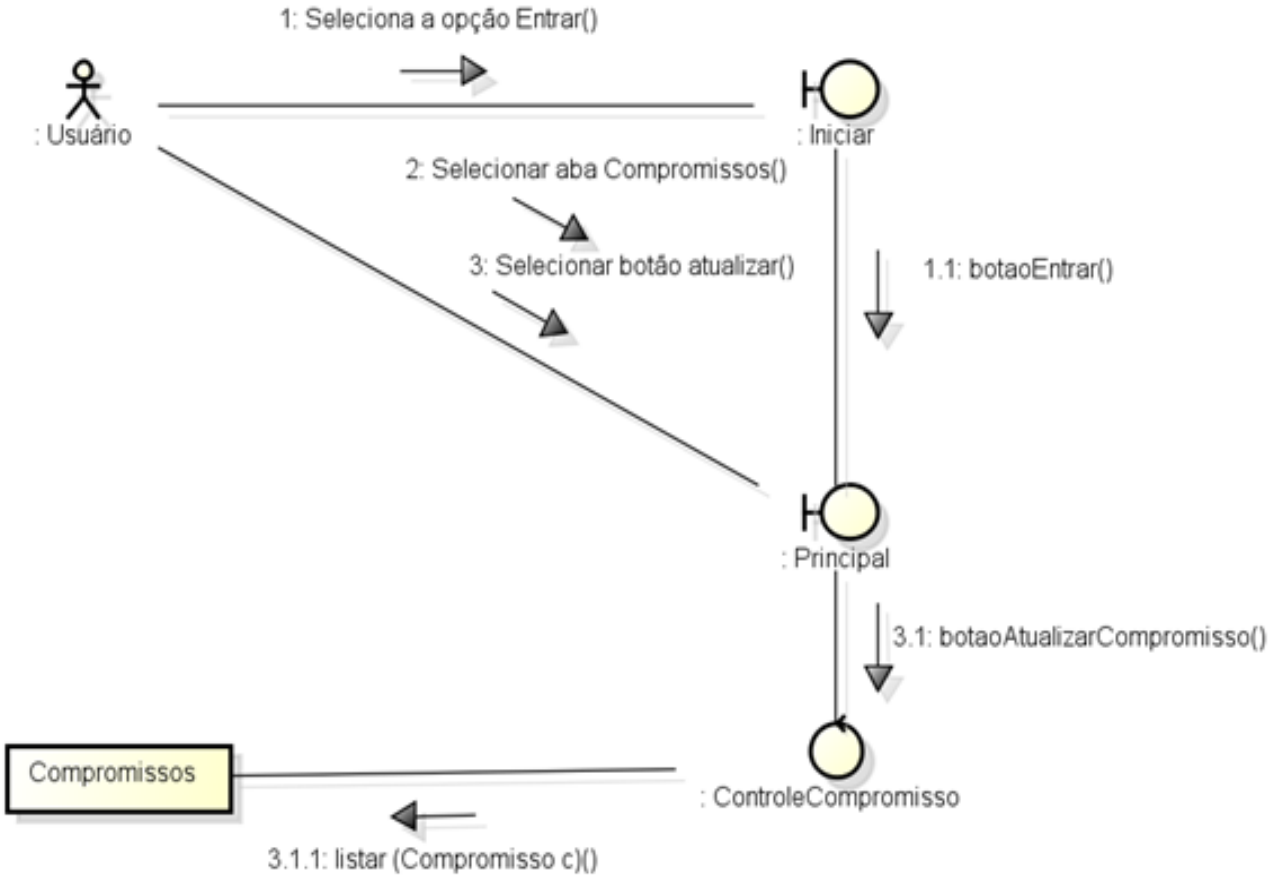
8 Remover Compromisso

Diagrama de Comunicação do Caso de Uso Remover Compromisso



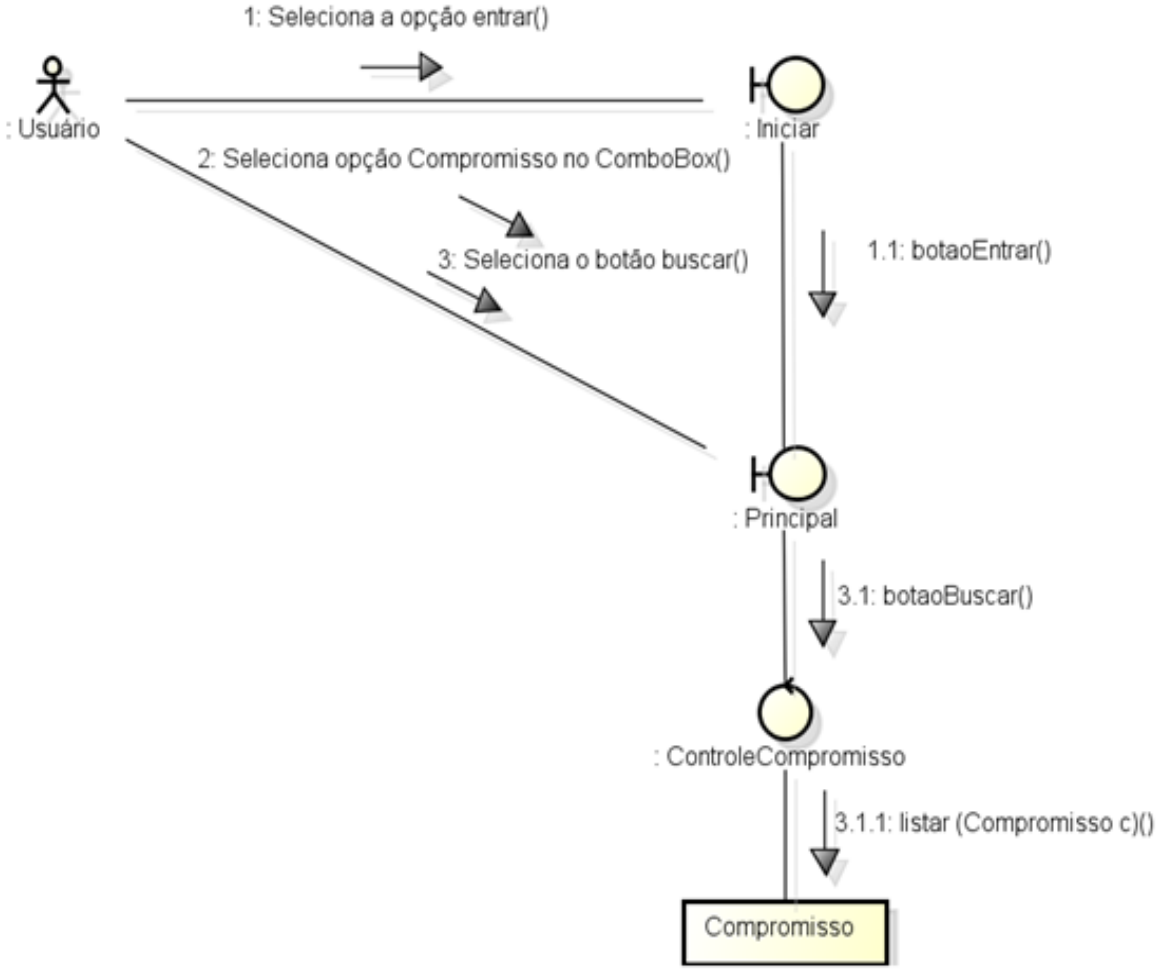
9 Atualizar Compromissos

Diagrama de Comunicação do Caso de Uso Atualizar Compromissos



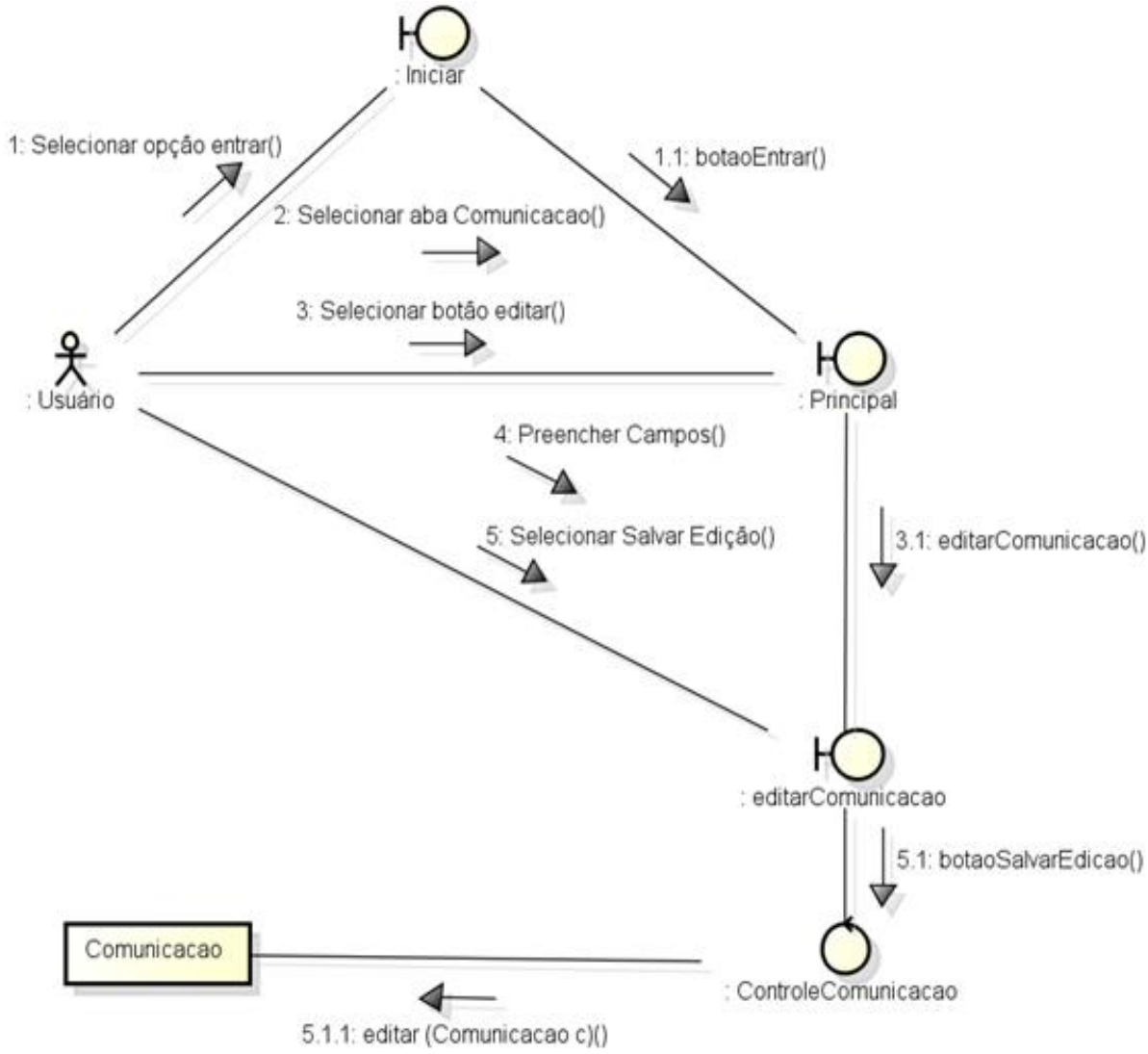
10 Buscar Nome Compromisso

Diagrama de Comunicação do Caso de Uso Buscar Nome Compromisso



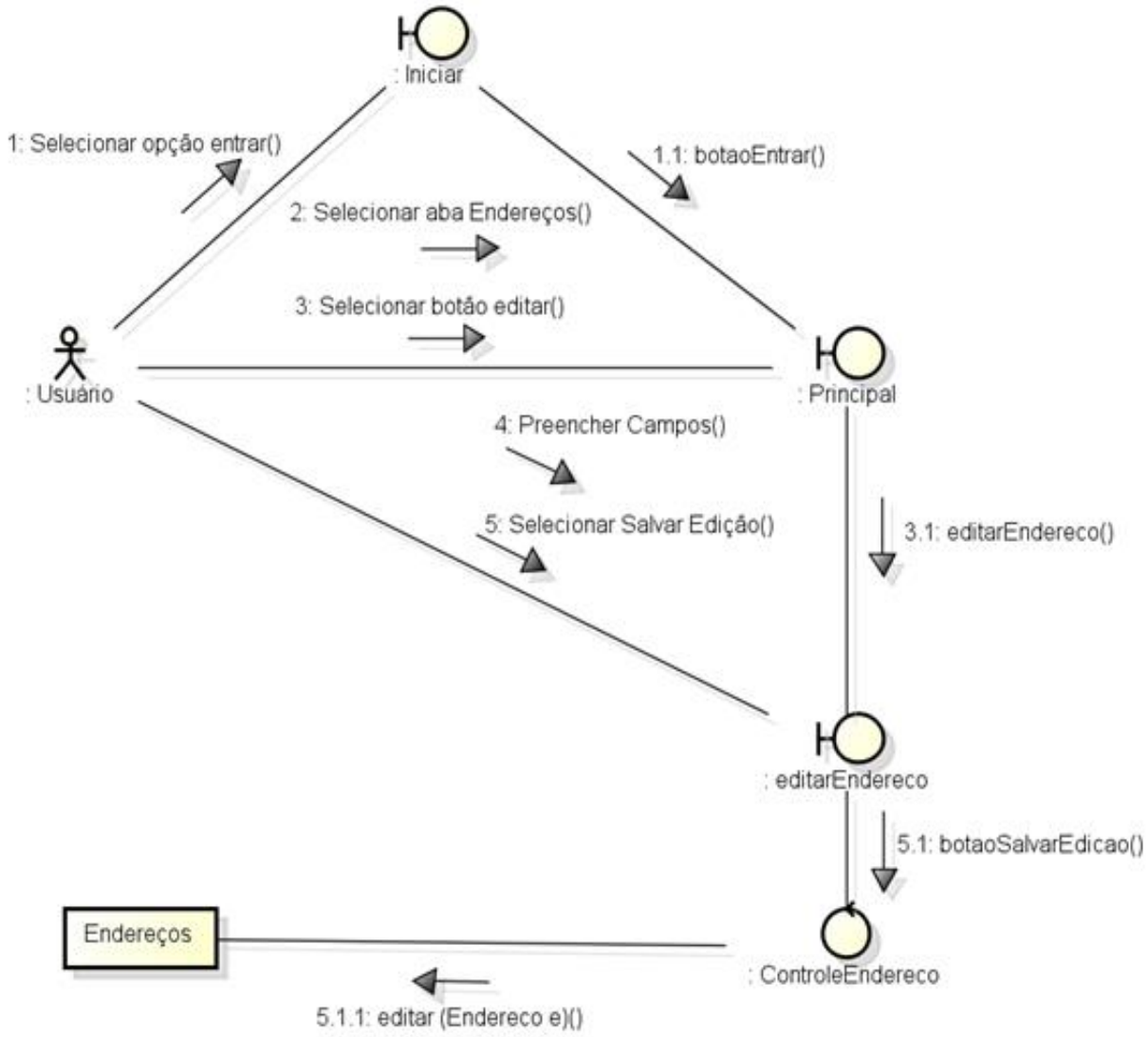
11 Editar Comunicação

Diagrama de Comunicação do Caso de Uso Editar Comunicação



12 Editar Endereço

Diagrama de Comunicação do Caso de Uso Editar Endereço

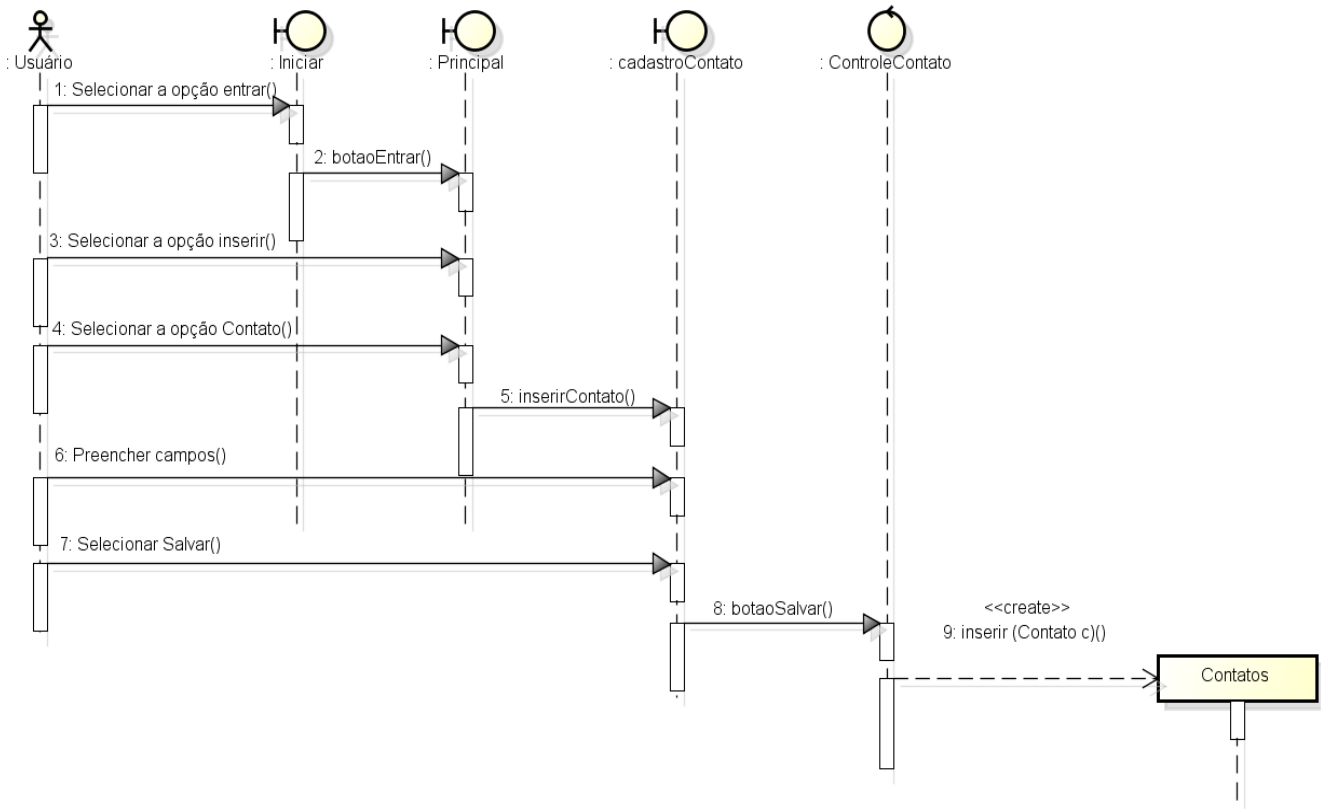


Apêndice E – Diagrama de Sequência

Conforme apresentado o conceito sobre o diagrama de comunicação, apesar de que as informações contidas nos dois esquemas sejam análogas, o foco entre eles é totalmente diferente. Segundo Medeiros (2006), os diagramas de sequência enfatizam a perspectiva temporal, descrevendo a maneira como os objetos colaboram em algum comportamento ao longo do tempo. A seguir serão apresentados os diagramas de sequência da aplicação desenvolvida.

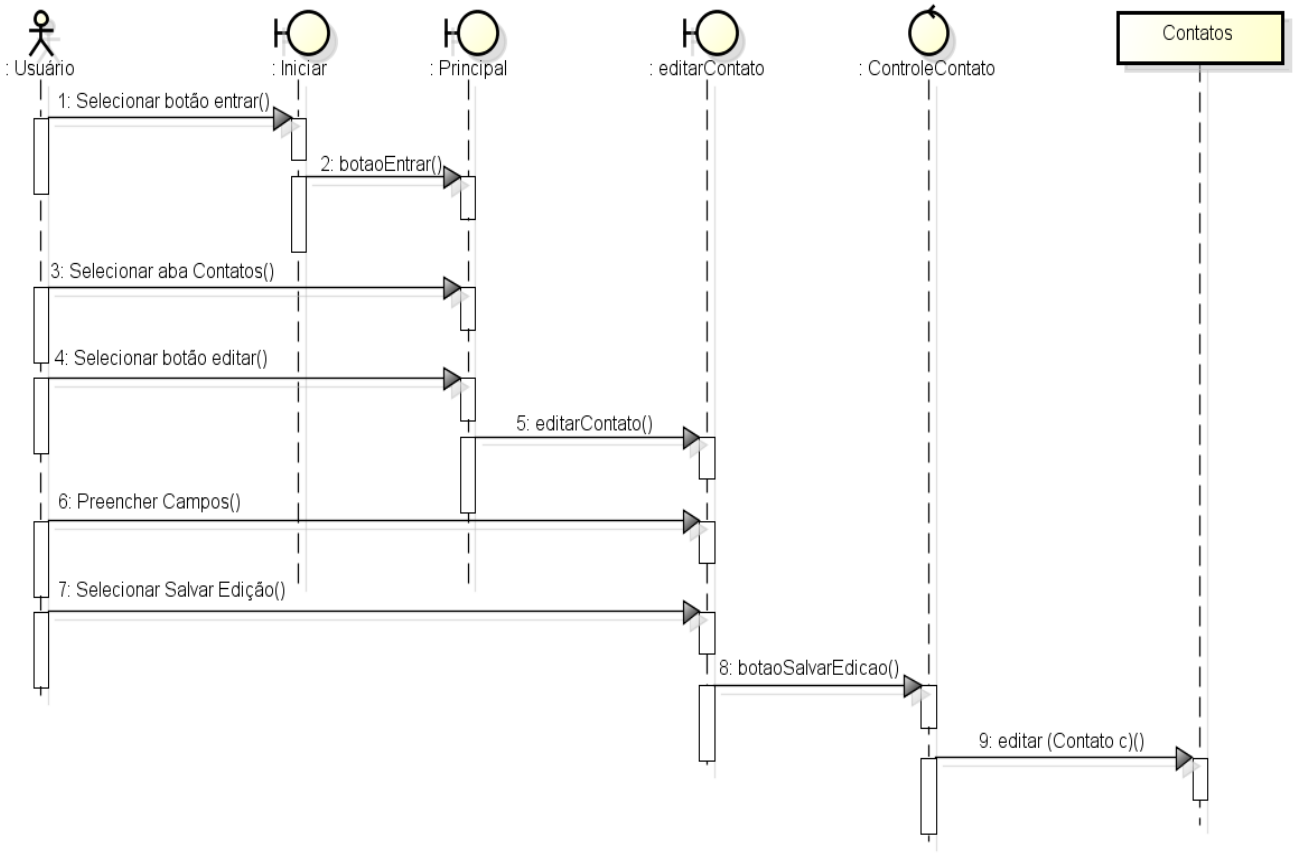
1 Inserir Contato

Diagrama de Sequência do Caso de Uso Inserir Contato



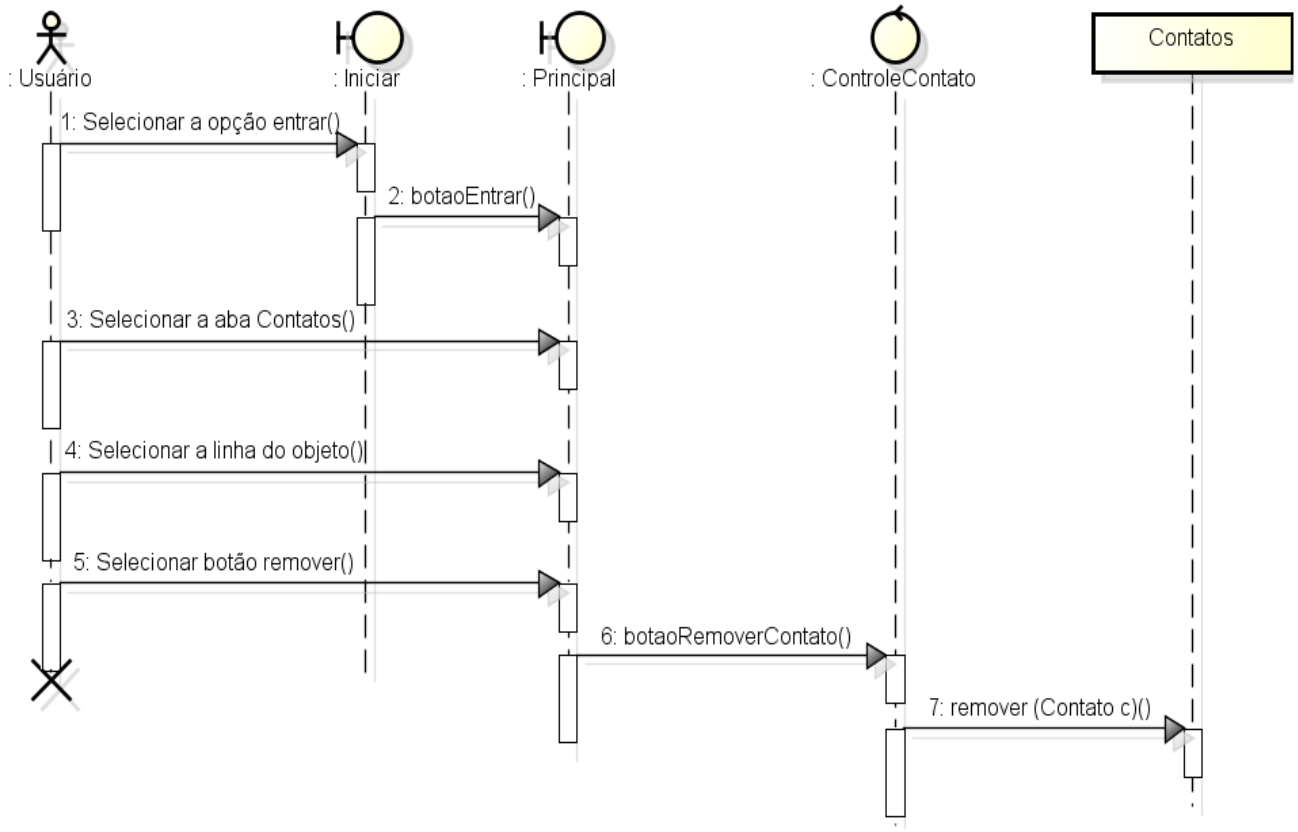
2 Editar Contato

Diagrama de Sequência do Caso de Uso Editar Contato



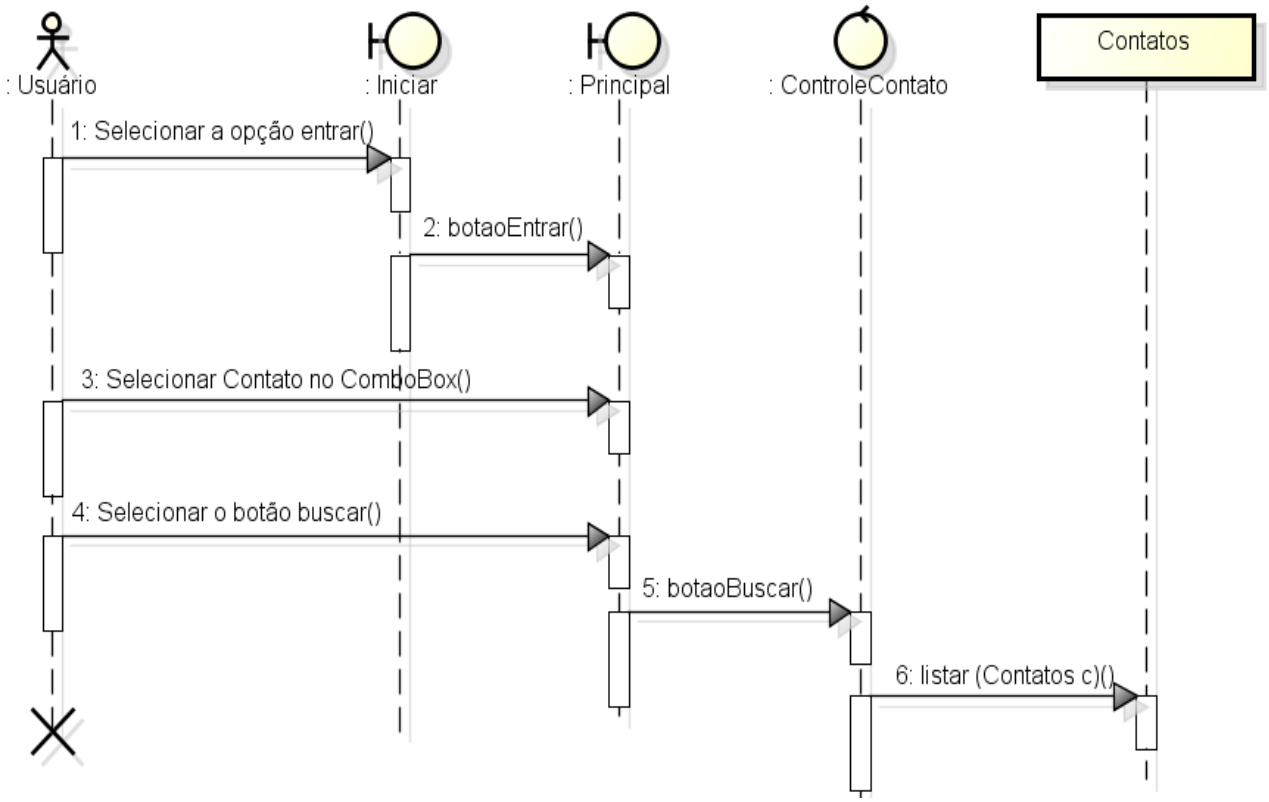
3 Remover Contato

Diagrama de Sequência do Caso de Uso Remover Contato



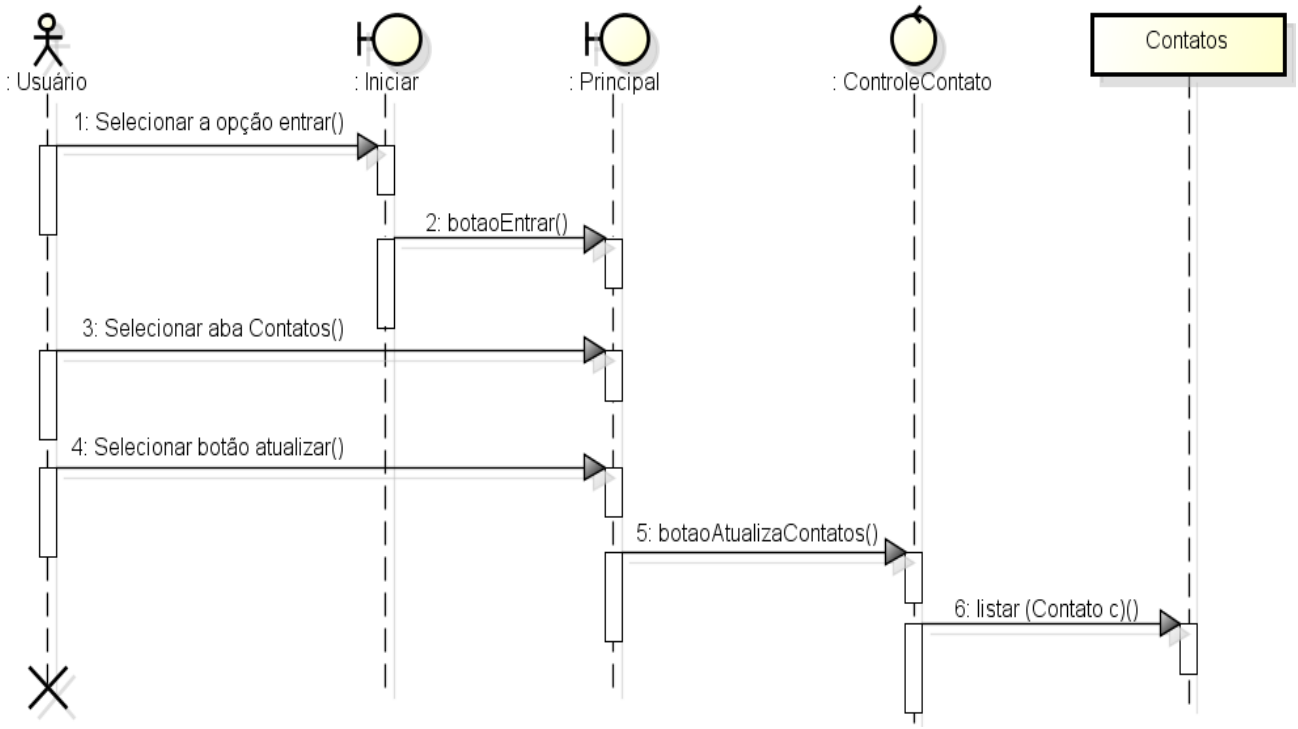
4 Buscar Nome Contato

Diagrama de Sequência do Caso de Uso Buscar Nome Contato



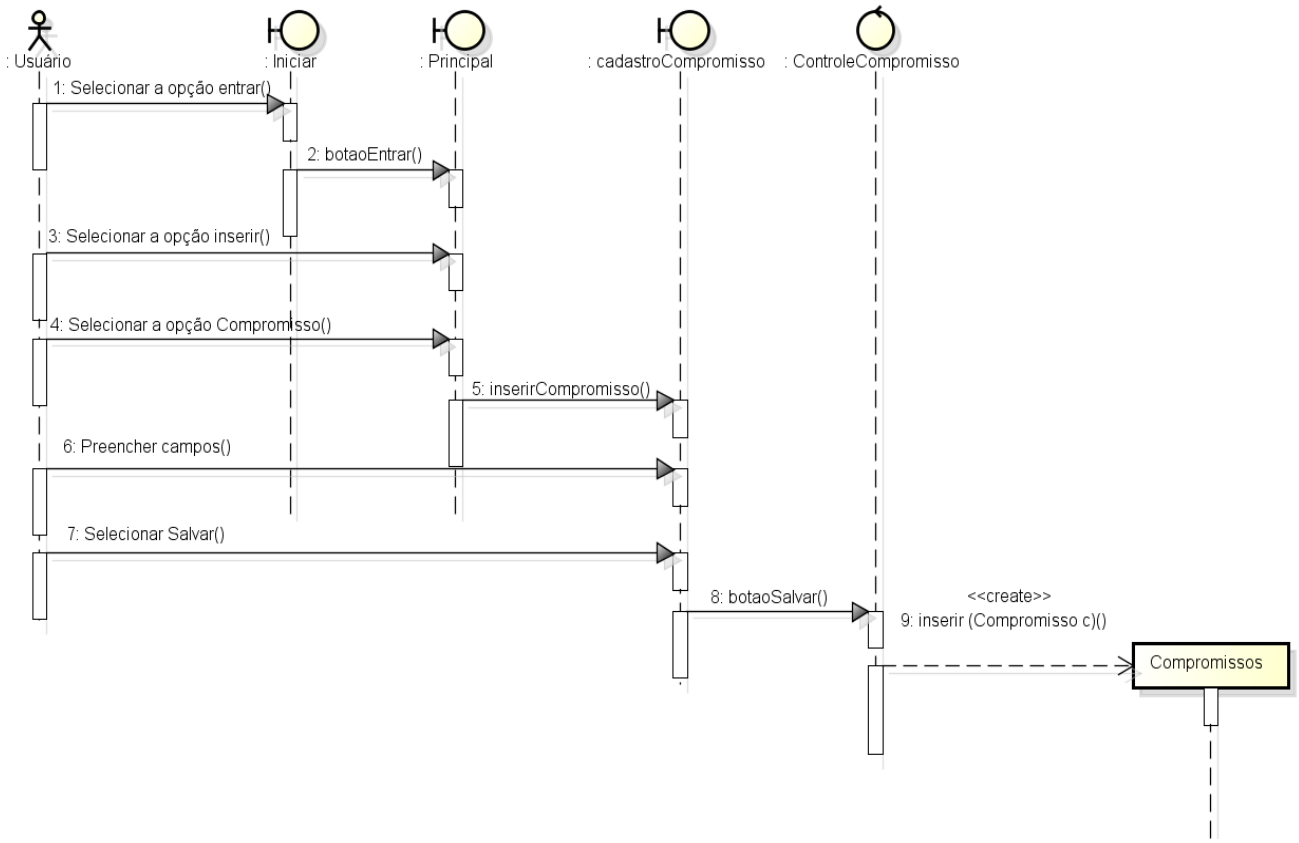
5 Atualizar Contatos

Diagrama de Sequência do Caso de Uso Atualizar Contatos



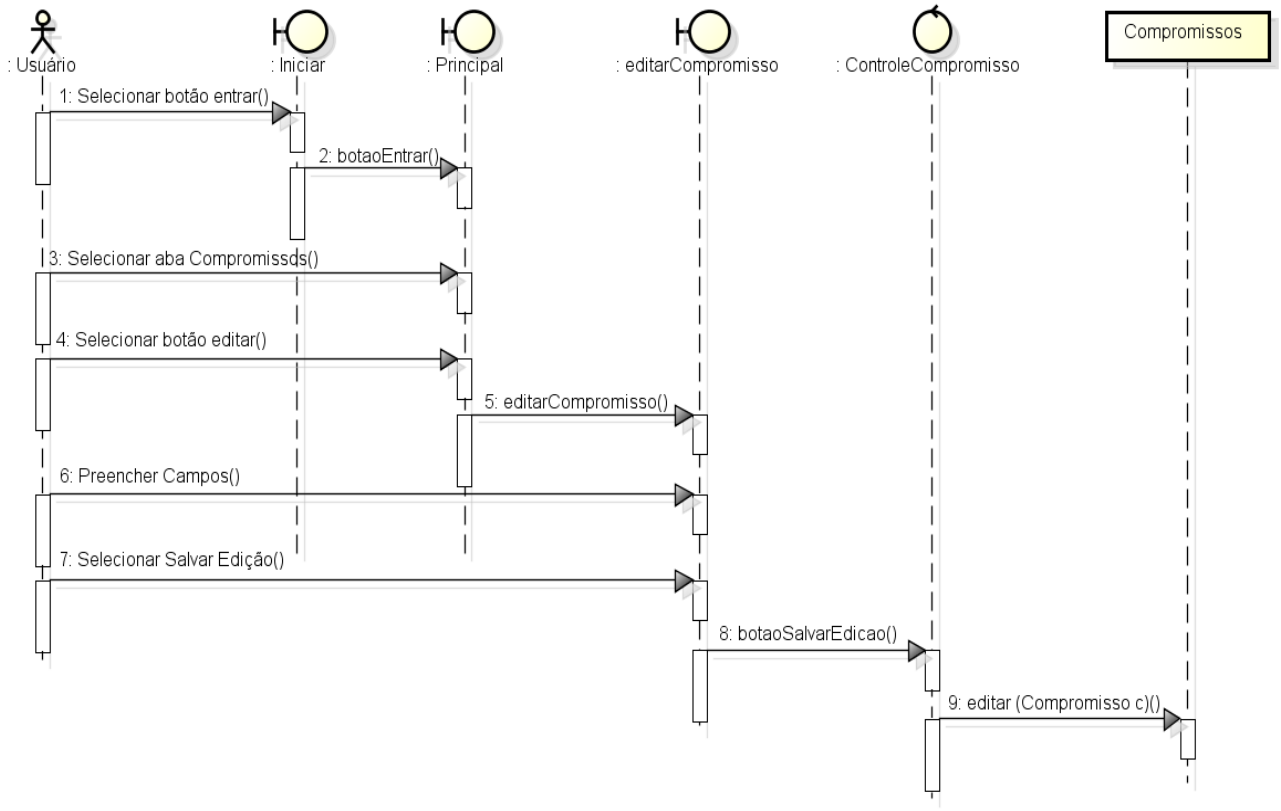
6 Inserir Compromissos

Diagrama de Sequência do Caso de Uso Inserir Compromissos



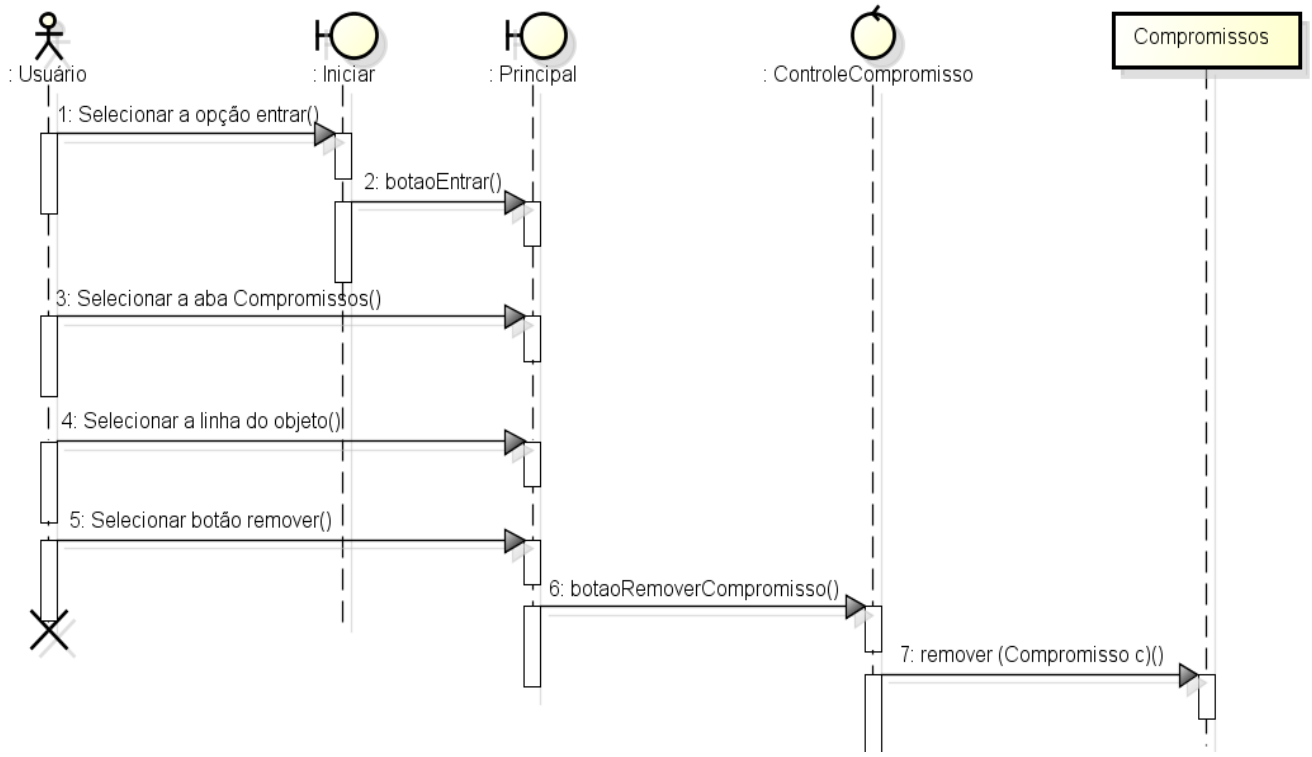
7 Editar Compromissos

Diagrama de Sequência do Caso de Uso Editar Compromissos



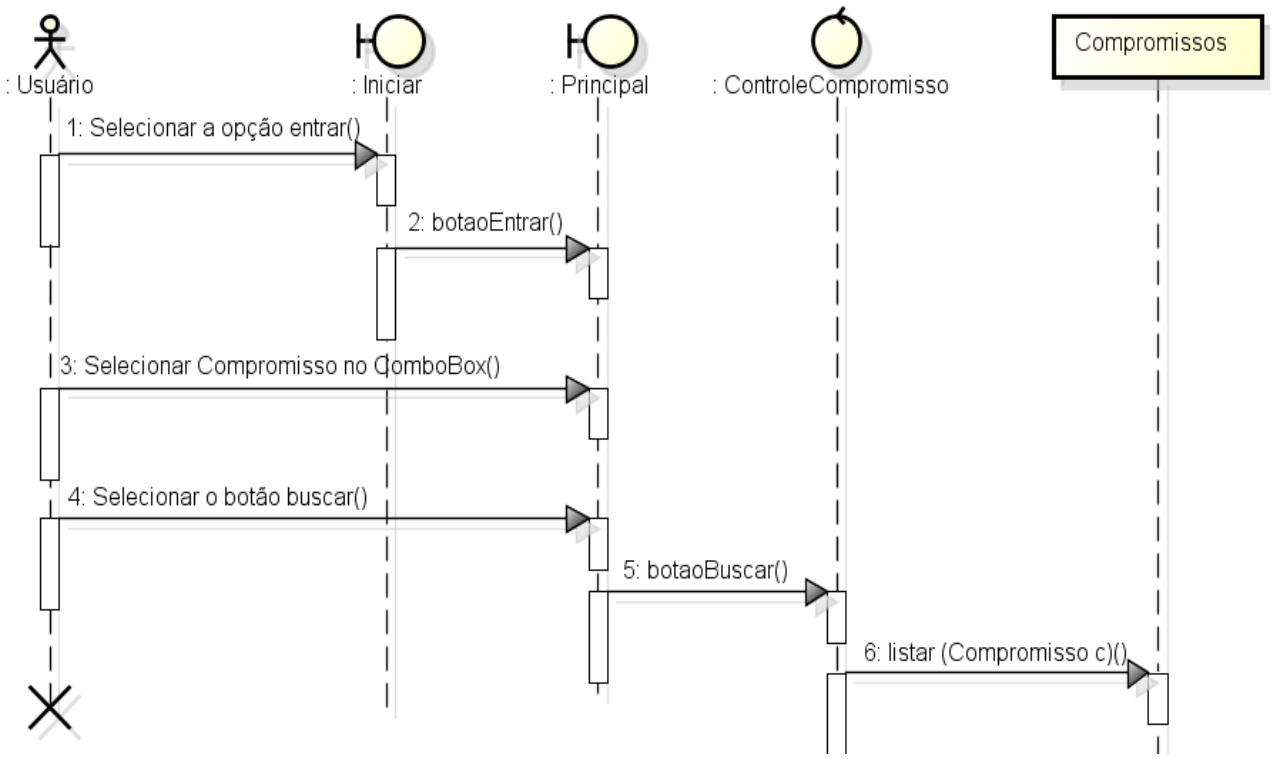
8 Remover Compromisso

Diagrama de Sequência do Caso de Uso Remover Compromisso



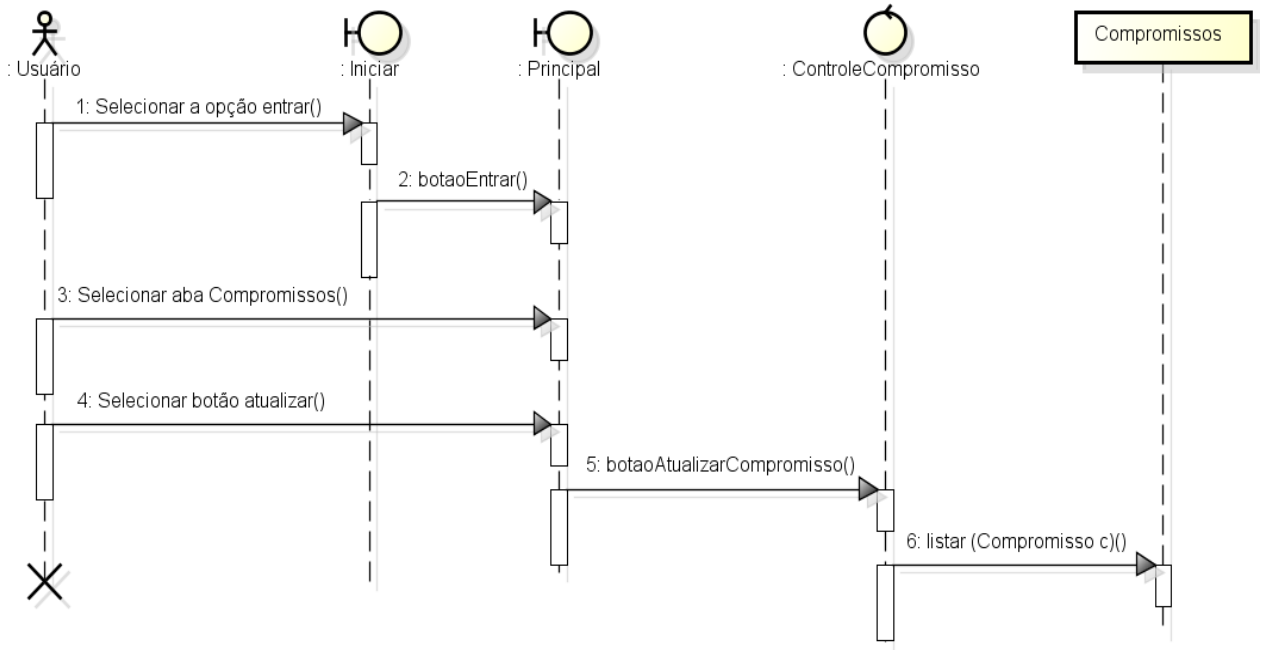
9 Buscar Nome Compromisso

Diagrama de Sequência do Caso de Uso Buscar Nome Compromisso



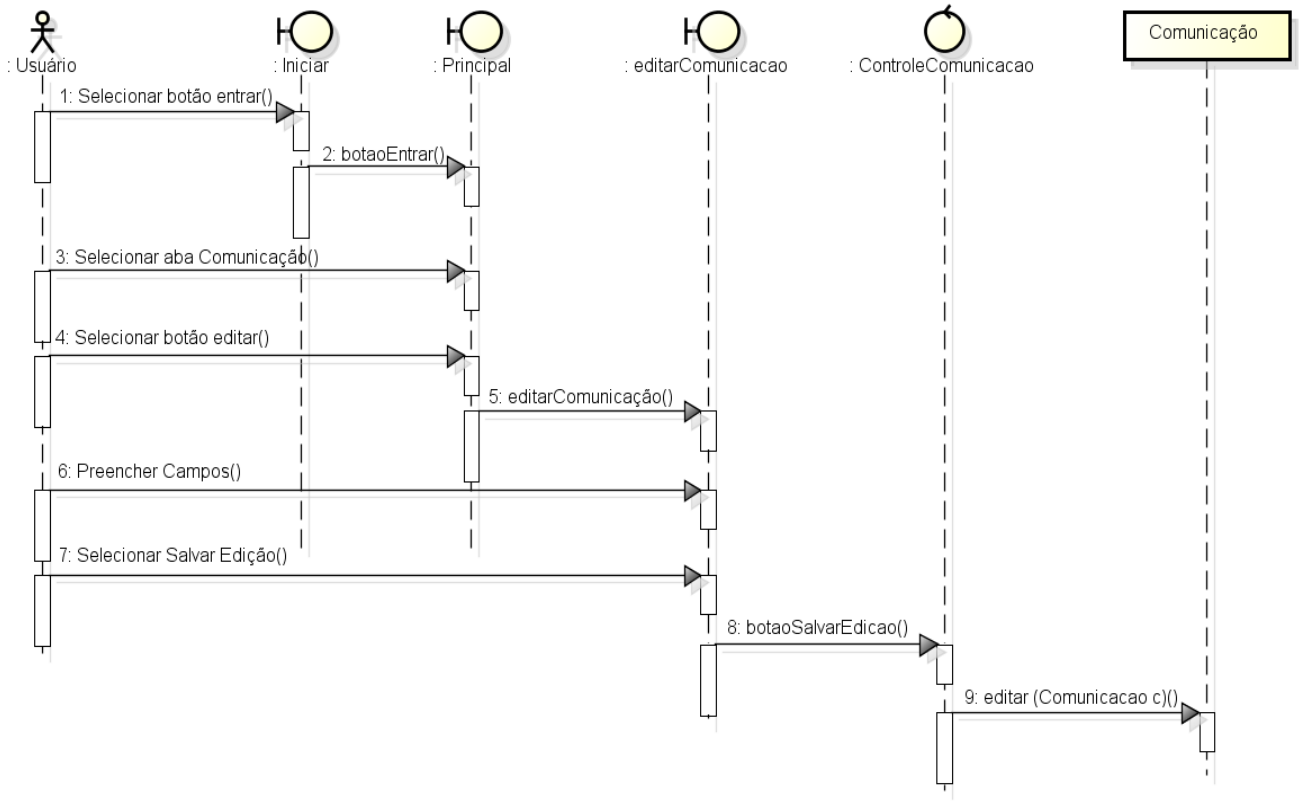
10 Atualizar Compromissos

Diagrama de Sequência do Caso de Uso Atualizar Compromissos



11 Editar Comunicação

Diagrama de Sequência do Caso de Uso Editar Comunicação



12 Editar Endereço

Diagrama de Sequência do Caso de Uso Editar Endereço