

**CESAR AUGUSTO COUTO SANTOS
JOÃO PAULO PEREIRA**

**FINDPRO: SISTEMA DE RECUPERAÇÃO TEXTUAL
EM PUBLICAÇÕES ACADÊMICAS**

**UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE
2012**

**CESAR AUGUSTO COUTO SANTOS
JOÃO PAULO PEREIRA**

**FINDPRO: SISTEMA DE RECUPERAÇÃO TEXTUAL
EM PUBLICAÇÕES ACADÊMICAS**

Trabalho de Conclusão de Curso apresentado
ao Departamento de Sistemas de Informação
como requisito parcial para a obtenção do
título de Bacharel em Sistemas de Informação
desenvolvido sob orientação do prof. Roberto
Ribeiro Rocha.

**UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE
2012**

**CESAR AUGUSTO COUTO SANTOS
JOÃO PAULO PEREIRA**

**FINDPRO: SISTEMA DE RECUPERAÇÃO TEXTUAL
EM PUBLICAÇÕES ACADÊMICAS**

Trabalho de Conclusão de Curso defendido e aprovado em ___/___/___ pela banca
examinadora constituída pelos professores:

Professor ROBERTO RIBEIRO ROCHA
Orientador

Professor
Examinador

Professor
Examinador

Dedicado às nossas famílias, professores e amigos
que sempre nos incentivaram ao longo de nossas vidas.

AGRADECIMENTOS

Agradecemos primeiramente a Deus, que nos deu forças nos momentos mais difíceis. Sem Ele não seríamos capazes de superar os obstáculos.

Ao Prof. Roberto Ribeiro Rocha, orientador deste trabalho, pelo auxílio e tempo dedicado ao esclarecimento de nossas dúvidas.

À Prof^a. Dra. Joelma Faria, Prof^a. Isis Mendes Fernandes e ao Prof. Ms. José Luiz da Silva, pela atenção em corrigir nossos textos e nos guiar para que o trabalho sempre atendesse às normas técnicas e ortográficas.

De Cesar Augusto:

Agradeço a todos os meus amigos e familiares, especialmente meus pais, minha esposa Isis e aos meus filhos Estela e Julio Cesar, pelo apoio e incentivo ao longo de toda a minha vida e pela compreensão nos momentos de ausência durante o desenvolvimento deste trabalho.

De João Paulo:

Agradeço aos meus pais e meus irmãos pelo apoio e incentivo ao longo da vida, estes foram fundamentais para a minha formação. Gostaria de agradecer também a todos os meus amigos, que ao longo da faculdade foram fundamentais e serão lembrados por toda vida.

“Apenas busquem conhecimento”.
Autor Desconhecido.

PEREIRA, João Paulo; SANTOS, Cesar Augusto Couto. **FindPro – Sistema de Recuperação Textual em Publicações Acadêmicas.** Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, 2012.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de recuperação textual que possibilita aos seus usuários usufruir, de maneira dinâmica, do conteúdo informacional contido nas publicações acadêmicas desenvolvidas na universidade. Para o desenvolvimento do sistema foi utilizado como metodologia o processo ICONIX, bem como, as técnicas de pesquisa aplicada. Para a implementação do projeto foram utilizadas tecnologias baseadas na plataforma JavaEE como, JavaServer Faces e JBoss Seam, assim como o framework Lucene para a indexação e recuperação das informações. Desta forma, o presente trabalho expõe ao longo de cinco capítulos todas as etapas percorridas durante o desenvolvimento do projeto, combinadas com uma descrição das ferramentas empregadas. Visando a disseminação do conhecimento adquirido durante esta pesquisa, disponibiliza-se também, todo o código fonte da aplicação para download.

Palavras-chave: Sistema de Recuperação Textual. JavaEE. Apache Lucene. JBoss Seam. ICONIX.

PEREIRA, João Paulo; SANTOS, Cesar Augusto Couto. **FindPro – Textual Retrieval System in Academic Publications.** Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, 2012.

ABSTRACT

This work shows the development of a textual system retrieval that makes possible to the users enjoy, in dynamic way, of informational content contained in academic publications developed at the university. In development of this system was used the ICONIX Process's methodologies, as well as applied research techniques. For implementation were used JavaEE platform based on technologies as JavaServer Faces and JBoss Seam as well as the framework Lucene for indexing and retrieval of information. Thus, this present work exhibit on five chapters all steps were taken during project development, combined with a description of the tools employed. Aiming to spread the knowledge acquired during this research, It also provides, the entire application source code for download.

Key-words: Textual Retrieval System. JavaEE. Apache Lucene. JBoss Seam. ICONIX.

LISTA DE FIGURAS

Figura 1 - Processo de busca em um sistema de recuperação textual	18
Figura 2 - Divisão de camadas em uma aplicação Java EE	23
Figura 3 - Componentes de uma aplicação Java EE.....	24
Figura 4 - <i>Containers</i> de aplicações Java EE	25
Figura 5 - Ciclo de vida das requisições AJAX do RichFaces.....	28
Figura 6 - Tipos de EJB.....	30
Figura 7 - Integração do JBoss Seam com a arquitetura Java EE	32
Figura 8 - Trabalho do <i>framework</i> Apache Lucene junto da aplicação	34
Figura 9 - Modelo de Visão Dinâmico e Estático	39
Figura 10 - Diagrama de Casos de Uso do sistema de recuperação textual FindPro	44
Figura 11 - Fluxo de Eventos Gerenciar Projetos (Cadastrar Projeto).....	46
Figura 12 - Modelo de Domínio do sistema de recuperação FindPro.....	47
Figura 13 - Diagrama de Robustez Gerenciar Projeto (Cadastrar Projeto)	48
Figura 14 - Formulário de cadastro de novos projetos	49
Figura 15 - Modelo de Domínio atualizado	50
Figura 16 - Tabelas do banco de dados findpro	50
Figura 17 - Fluxo de Eventos Cadastrar Projeto	51
Figura 18 - Diagrama de Sequência Cadastrar Projeto	52
Figura 19 - Fragmento do Diagrama de Classes	53
Figura 20 - Página de cadastro de novos projetos	54
Figura 21 - Página de busca do sistema de recuperação	57
Figura 22 – Teste de cadastro de publicações	58
Figura 23 - Publicações cadastradas.....	59
Figura 24 - Resultado de uma procura no sistema	60
Figura 25 - Resultado de uma procura utilizando o campo “palavra-chave”	61

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AJAX	<i>Asynchronous Javascript and XML</i>
AOL	<i>America Online</i>
API	<i>Application Programming Interface</i>
BSD	<i>Berkeley Software Distribution</i>
EJB	<i>Enterprise JavaBeans</i>
HP	Hewlett-Packard
HTML	<i>Hypertext Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IBM	<i>International Business Machines Corporation</i>
ISO	<i>International Standardization Organization</i>
JCP	<i>Java Community Process</i>
JEE	<i>Java Enterprise Edition</i>
JME	<i>Java Micro Edition</i>
JSE	<i>Java Standard Edition</i>
JSF	<i>JavaServer Faces</i>
JSP	<i>JavaServer Pages</i>
JVM	<i>Java Virtual Machine</i>
MDB	<i>Message Driven Beans</i>
NBR	<i>Denominação de Norma da Associação Brasileira de Normas Técnicas</i>
PDF	<i>Portable Document Format</i>
POO	<i>Programação Orientada a Objetos</i>
RUP	<i>Rational Unified Process</i>
SQL	<i>Structured Query Language</i>
UML	<i>Unified Modeling Language</i>
UNIVAS	<i>Universidade do Vale do Sapucaí</i>
WWW	<i>World Wide Web</i>
XHTML	<i>Extensible HyperText Markup Language</i>
XML	<i>Extensible Markup Language</i>
XP	<i>Extreme Programming</i>

SUMÁRIO

INTRODUÇÃO	12
2 QUADRO TEÓRICO	16
2.1 Sistemas de Recuperação Textual	16
2.1.1 Indexação	18
2.2 Tecnologias	18
2.2.1 Java.....	19
2.2.1.1 Linguagem de Programação Orientada a Objetos.....	19
2.2.1.2 Linguagem de Programação Interpretada.....	20
2.2.1.3 Linguagem de Programação Multiplataforma	21
2.2.2 Java Enterprise Edition (JEE).....	22
2.2.3 JavaServer Faces.....	25
2.2.4 RichFaces	26
2.2.5 Enterprise JavaBeans	28
2.2.6 JBoss Seam	30
2.2.6.1 Contextos	31
2.2.6.2 Componentes.....	32
2.2.7 Apache Lucene	33
2.2.8 Apache PDFBox	34
2.2.9 PostgreSQL.....	35
2.3 Engenharia de Software	36
2.3.1 Processo de Desenvolvimento de Software	36
2.3.2 ICONIX.....	37
3 QUADRO METODOLÓGICO	40
3.1 Tipo de Pesquisa.....	40
3.2 Contexto de Pesquisa	41
3.3 Metodologia de Desenvolvimento	42
3.3.1 Análise de Requisitos	42
3.3.2 Análise e Projeto Preliminar	47
3.3.3 Projeto Detalhado	50
3.3.4 Implementação	52

4	DISCUSSÃO DE RESULTADOS	54
5	CONCLUSÃO	61
	REFERÊNCIAS	63
	APÊNDICE I – Artefatos Do Projeto	66
	APÊNDICE II – Tutorial Apache Lucene.....	125
	APÊNDICE III – Tutorial Integração Lucene-PDFBox.....	137

INTRODUÇÃO

O objetivo geral deste trabalho foi desenvolver um sistema web que sirva de ferramenta e facilite a pesquisa em trabalhos acadêmicos, tornando-os mais acessíveis, disponíveis e possibilitando que o usuário tenha um melhor proveito dos recursos informacionais neles contidos.

Para alcançar este objetivo geral, os seguintes objetivos específicos foram colocados: a) modelar o sistema com a metodologia de desenvolvimento de *software* ICONIX; b) utilizar no desenvolvimento ferramentas que ofereçam suporte à plataforma web, mais precisamente JEE.

A internet revolucionou a maneira como ordenamos nossos conhecimentos e informações. Ela derrubou barreiras, encurtou distâncias, modificou nossos hábitos, comportamento e tornou veículos de comunicação, antes considerados perfeitos, agora obsoletos e inviáveis.

Hoje em dia é comum lermos jornais, revistas e até mesmo publicações acadêmicas em páginas na internet. São necessários apenas alguns segundos para que tenhamos as informações que buscamos na tela de nosso computador, e desta forma, a internet provocou uma mudança estrutural e passou a ser facilitadora e disseminadora de informação, considerando que a informação utilizou diversos suportes desde a imprensa até a era digital.

Essa mudança estrutural também atingiu as universidades e algumas de suas características já são visíveis há algum tempo, como por exemplo, a informatização das secretarias, das listas de chamada e do resultado das avaliações.

As universidades têm como ponto central as bibliotecas, que preservam o conhecimento da humanidade com seu acervo impresso. Uma característica das bibliotecas tradicionais é que tanto as publicações quanto os seus registros utilizam o papel como suporte ao catálogo dos trabalhos acadêmicos, fazendo com que os interessados dependam de sua localização física e da disponibilidade de cópias para terem acesso a esses documentos.

Contudo, neste trabalho, não nos deteremos a discutir os conceitos estruturais das bibliotecas tradicionais ou modernas, mas sim, apresentaremos uma alternativa fundamentada nos princípios da web para auxiliar os usuários a fazerem um melhor proveito do conteúdo por ela mantido, como por exemplo, agilizar a procura detalhada em uma grande quantidade de documentos, permitindo que o aluno possa realizar suas pesquisas com maior rapidez.

Percebemos a necessidade que os alunos têm de encontrar informações específicas em um número muito grande de documentos, e então decidimos desenvolver um sistema web que facilite a procura por meio de buscas simples desde autor, tema, ou orientador, até pesquisas mais complexas como palavras ou frases inteiras, tornando essa tarefa mais dinâmica além de possibilitar o uso de todas as vantagens proporcionadas pela internet.

Para tanto, entender as características da web e quais os elementos dos sistemas de informações são pertinentes a este projeto é uma tarefa fundamental e serão discutidas a seguir.

Surgida no início da década de 1990, a World Wide Web (www) é apenas mais um serviço da internet, como os serviços de correio eletrônico ou lista de discussão. Foi criada com a intenção de oferecer interfaces mais amigáveis e intuitivas para as informações que eram disponibilizadas via internet. Baseado nesse conceito, a ideia de hipertexto de Theodor Nelson e Douglas Engelbart concebida em 1962 foi aplicada.

Os sistemas de hipertexto consistem em uma abordagem de estruturação e manipulação de textos, permitindo uma leitura não linear do mesmo. Em tais sistemas os documentos são dispostos em uma base de dados repleta de conexões, formando então a rede hipertextual.

Nessa rede, cada unidade de informação da base de dados é conectada por links de acordo com associações entre seus conteúdos. A estrutura de hipertextos determina e descreve o sistema de ligações e de relacionamentos entre os documentos, sendo um fator decisivo na facilidade de criação, de uso e de atualização dos hiperdocumentos.

Já os sistemas de recuperação textual são desenvolvidos para indexar e recuperar documentos do tipo textual, ou seja, documentos cujas informações são descritas através da linguagem natural. São sistemas que trabalham basicamente com informações em texto, mas, que através de filtros adequados podem analisar outros formatos que contenham textos, figuras, gráficos, tabelas e imagens, desde que possuam as características de um documento textual, como o PDF ou DOC.

Pesquisando sobre o assunto, observamos que diversos trabalhos sobre como aperfeiçoar a recuperação da informação e a procura em grandes volumes de documentos foram desenvolvidos, porém a maioria tem foco específico em uma determina área do conhecimento, como por exemplo, o trabalho de Amorim; Cheriaf (2007), que desenvolveram um sistema de indexação e recuperação de informação na área de arquitetura, engenharia e construção, e que, segundo os próprios autores seria o único sistema desta categoria no Brasil.

Sayão e Barros (1995) também desenvolveram um trabalho que apoia a ideia de que os sistemas de recuperação textual organizam as informações convenientemente, já que nem sempre elas estão dispostas da maneira como necessitamos, e, analisando esse trabalho, percebemos que tal sistema seria uma possível solução para atender o problema da busca em grande quantidade de arquivos e informações.

A proposta do presente trabalho é desenvolver uma ferramenta de recuperação textual capaz de utilizar um conjunto de conhecimentos das características dos documentos existentes em um banco de dados para melhorar um processo de pesquisa. Em outras palavras, um sistema de busca que melhoraria a qualidade dos resultados através da análise e combinação das palavras procuradas.

Para o desenvolvimento deste sistema utilizaremos ferramentas e recursos tecnológicos, dentre eles podemos citar a linguagem de programação JavaTM em sua versão para aplicações corporativas (JEE) e o framework Lucene da fundação Apache, além de outras tecnologias que servirão de apoio para o projeto.

Esta ferramenta seria disponibilizada estritamente para os usuários de uma instituição de ensino superior localizada no sul de Minas Gerais, mas sua infraestrutura e organização estariam preparadas para receber requisições de qualquer pessoa ou lugar com acesso à internet.

Existem hoje, na instituição de ensino a qual este projeto se aplica, duas opções de pesquisa em trabalhos acadêmicos: uma delas é a pesquisa in loco nos documentos disponíveis e, para isso o aluno precisa locomover-se até a biblioteca e acessar o documento, além da limitação de empréstimos de volumes; a outra é uma pesquisa online oferecida pelo portal do aluno, na qual só é possível procurar documentos por termos muito generalizados, como autor ou título, o que não abrange o texto contido nas publicações e ainda assim não exclui a necessidade do comparecimento à biblioteca, pelo fato de não ser permitido à visualização do conteúdo nem da situação do documento, como por exemplo, se ele se encontra emprestado ou se há fila de espera.

Para os alunos que moram fora da cidade onde está localizada a instituição, e que constituem uma parte significativa do corpo discente, o deslocamento, por vezes, se torna um obstáculo, seja por motivos financeiros, seja disponibilidade de tempo ou até mesmo obstáculos físicos no caso de alunos portadores de deficiência. Dentro deste contexto, ainda há outro obstáculo que ocorre na biblioteca: não haver exemplares disponíveis e a possibilidade de existir uma longa lista de reservas.

O sistema proposto visa unir o melhor de cada tipo de pesquisa citada anteriormente, por meio de um sistema que busque informações tanto no corpo quanto nos elementos pré-textuais dos trabalhos, possibilitando uma procura muito mais abrangente e que elimina tanto a necessidade de se locomover a biblioteca quanto à formação de filas de espera.

O projeto apresenta grande relevância acadêmica, pois permite que vários usuários da biblioteca acessem, de forma simultânea, o mesmo documento independente do número de cópias existentes, além de tornar mais eficiente a forma como a pesquisa é realizada. No aspecto financeiro evita atrasos e multas, e vale ressaltar que esses atrasos na devolução também prejudicam os alunos que esperam para ter acesso ao documento emprestado.

Este trabalho se encontra dividido em cinco capítulos nos quais são discutidas todas as ferramentas e metodologias empregadas no desenvolvimento do sistema de recuperação textual. No próximo capítulo apresentam-se as teorias que embasaram esta pesquisa bem como as ferramentas que auxiliaram em seu desenvolvimento. No terceiro capítulo discute-se sobre as etapas que foram cumpridas para que os objetivos fossem alcançados, como, levantamento de requisitos, documentação e implementação. Os resultados obtidos são apresentados no quarto capítulo e finalmente, no quinto e último capítulo, são expostas as conclusões obtidas pelos autores.

2 QUADRO TEÓRICO

Neste capítulo serão discutidas as técnicas, metodologias e tecnologias empregadas neste projeto e na próxima seção serão abordados os sistemas de recuperação textual, um tipo de *software* voltado para a procura de informações em documentos de texto.

2.1 Sistemas de Recuperação Textual

Geralmente é considerado um sistema de recuperação textual aquele que tem a função de levar o usuário aos documentos de texto que melhor satisfazem a sua necessidade de informação.

Essa necessidade é caracterizada, por exemplo, por uma pessoa que se encontra em uma situação na qual precise de determinada informação e busque resolver essa carência realizando uma consulta em um sistema de recuperação.

Estudos afirmam que os sistemas de recuperação da informação, mais precisamente, os sistemas de recuperação textual são caracterizados por uma variedade de maneiras, que vão desde uma descrição de seus objetivos, até modelos relativamente abstratos de seus componentes e processos. Embora essas caracterizações nem sempre concordem uma com a outra, todas elas tendem a compartilhar alguns traços comuns quanto aos objetivos e resultados (Belkin; Croft, 1992, p. 2).

Um sistema de recuperação ideal é aquele que apresenta uma taxa mínima de “silêncio” e “ruído”, muito comuns neste tipo de *softwares*. O silêncio é caracterizado pela não recuperação de informações existentes nos documentos indexados. A informação existe e é relevante, no entanto, não é recuperada pelo sistema que a negligenciou. O ruído, por sua vez, consiste no excesso de informações irrelevantes recuperadas juntamente com as informações relevantes, em outras palavras, uma alta taxa de ruído seria a recuperação de informações imprestáveis para a pesquisa em questão (Associação Brasileira de Normas Técnicas, 1992).

Tanto o silêncio quanto o ruído prejudicam o resultado da pesquisa e exigem um grande esforço por parte do desenvolvedor do sistema para que durante o processo de recuperação das informações essa situação seja amenizada.

Outro fator importante consiste na comparação entre uma consulta e seus resultados, ou, em alguns casos, a interação direta entre o usuário e os textos encontrados, que conduz possivelmente à seleção de algum documento retornado. Esse documento é então avaliado e caso atenda às necessidades de informação, o usuário deixa o sistema de busca, caso contrário, a consulta é modificada para a realização de uma nova pesquisa. Essa modificação nos termos da consulta é chamada por Belkin; Croft (1992) de realimentação de relevância.

A Figura 1 representa o trabalho de um sistema de recuperação:

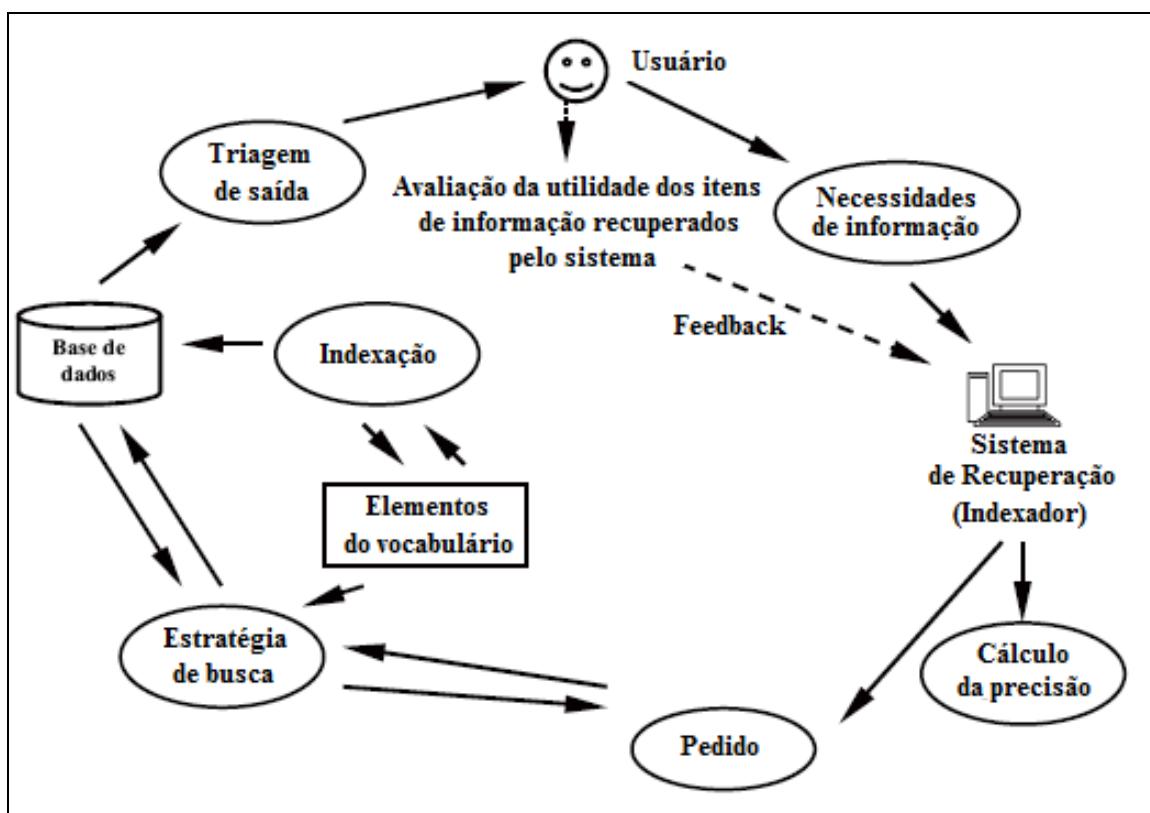


Fig. 1 - Processo de busca em um sistema de recuperação textual.

Fonte: Adaptado pelos autores baseados em Lancaster (2004).

Contudo, o foco destes sistemas é a rapidez e a relevância dos recursos de informação que o usuário vai eventualmente acessar, e para tanto, é preciso que os documentos estejam agrupados, organizados e suas palavras representadas em base de dados. Todo esse processo é conhecido como indexação e é o ponto central de um sistema de recuperação textual.

2.1.1 Indexação

O termo indexação se originou no contexto da chamada “explosão bibliográfica”, após a segunda guerra mundial, nos centros de documentação especializados onde eram realizados os controles bibliográficos. Este conceito surgiu da elaboração de índices e hoje está vinculado ao conceito de análise do assunto.

Para Lancaster (2004), a indexação é definida como uma atividade do tratamento temático da informação documental e tem como objetivo extrair os termos que representam os documentos com a finalidade de referenciá-los para uma melhor recuperação textual.

A determinação do assunto ocorre por meio da análise conceitual realizada pela leitura do texto, e se tratando de um sistema de recuperação onde o documento é cadastrado por um usuário antes da realização da indexação, cabe a este informar o assunto e alguns outros parâmetros pré-estabelecidos para facilitar o trabalho do sistema indexador.

Existem regulamentações que norteiam o processo de indexação, e a primeira norma publicada a esse respeito é de responsabilidade da *International Standardization for Organization* (ISO), publicada em 1985 com o título: *Documentation – methods for examining documents, determining their subjects, and selecting indexing terms*, de número 5963.

Em 1992, a Associação Brasileira de Normas Técnicas (ABNT) traduziu a norma ISO 5963, publicando-a como NBR 12.676: Métodos para análise de documentos – determinação de seus assuntos e seleção de termos de indexação.

2.2 Tecnologias

A seguir são apresentadas as tecnologias empregadas neste projeto, abordando desde a linguagem de programação utilizada, até o sistema gerenciador de banco de dados cujas informações foram persistidas.

2.2.1 Java

Java é uma linguagem de programação desenvolvida no início da década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems.

Segundo Gosling *et al* (2006), Java foi originalmente chamada de Oak e desenhada para ser embarcada em dispositivos eletrônicos, entretanto, com o passar dos anos foi totalmente revisada, se tornando uma linguagem de propósito geral e muito parecida com suas predecessoras C e C++, o que fez com que se tornasse a plataforma de desenvolvimento adotada mais rapidamente do que qualquer outra na história da computação.

O Java é considerado uma linguagem de alto nível e está disposta em três edições que atendem diferentes áreas do desenvolvimento:

- Java Micro Edition (JME) - é a tecnologia voltada para dispositivos móveis e possuí limitações quanto ao processamento e memória.
- Java Standard Edition (JSE) – é a tecnologia voltada para computadores pessoais que tenham um poder de processamento e memória considerável.
- Java Enterprise Edition (JEE) – é a tecnologia voltada para aplicações mais robustas. Normalmente utilizada em aplicações corporativas que podem estar ou não na internet.

Das edições citadas acima, utilizaremos a versão dirigida às aplicações WEB, neste caso JEE, que será discutida posteriormente.

As principais características do Java são a orientação a objetos, ser uma linguagem interpretada e Multiplataforma, as quais serão detalhadas a seguir.

2.2.1.1 Linguagem de Programação Orientada a Objetos

O conceito de programação orientada a objetos foi criado por Alan Kay, autor da linguagem de programação Smalltalk no início da década de 70, porém mesmo antes da criação dessa linguagem, algumas ideias da POO (Programação Orientada a Objetos) já eram aplicadas, sendo que a primeira a realmente utilizar estas ideias foi a Simula 67 desenvolvida em 1967.

Atualmente, um grande número de linguagens possuem características de orientação a objetos, tais como Java, C#, Python etc. O paradigma da programação orientada a objetos é baseado em alguns conceitos que definem uma forma de criar programas para computadores.

De acordo com Pressman (2006), a filosofia principal desse conceito é solucionar um problema por meio de uma representação computacional dos objetos existentes nessa questão, fazendo com que essa maneira de desenvolver programas fique mais próxima das situações como elas acontecem no mundo real.

2.2.1.2 Linguagem de Programação Interpretada

Fischer; Grodzinsky (1992) apresentaram as diferenças entre as linguagens de programação compiladas das interpretadas.

Quando se utiliza uma linguagem compilada, após escrever o código fonte, é necessário que o mesmo seja traduzido, ou seja, compilado para código binário executável. Essa tradução é feita por um software chamado de compilador.

As linguagens compiladas têm como vantagem produzir código executável de alto desempenho, o qual está ajustado para funcionar em um ambiente computacional específico. Estes programas só podem ser executados no tipo de computador em que foram compilados, uma vez que consistem, na verdade, em instruções de linguagem de máquina, entendidas e executadas pelo processador.

Já nas linguagens de programação interpretadas, o código fonte é interpretado por um programa chamado de interpretador, no caso do Java este programa se chama JVM (Java Virtual Machine), que percorre o código fonte e executa as ações indicadas pelas instruções no arquivo. Uma das vantagens das linguagens interpretadas está no fato dos programas poderem rodar em uma variedade de plataformas diferentes, pois estes só existem em código fonte.

2.2.1.3 Linguagem de Programação Multiplataforma

Essa característica da linguagem Java indica que um programa desenvolvido nela pode ser executado em computadores e sistemas operacionais diferentes daqueles em que foram projetados. Em algumas linguagens de programação como o Pascal e o C, o código fonte é compilado para uma plataforma específica, o que faz com que o código binário do programa possa ser executado somente por computadores que possuam esse sistema operacional. Segundo Horstmann; Cornell (2010 p. 21) “(...) escrever um programa que seja bom no Windows, Macintosh e nas dez versões do Unix é um esforço de proporções heroicas”.

Para que o mesmo programa possa ser executado em plataformas diferentes é necessário compilá-lo novamente, mas dessa vez utilizando a plataforma na qual se deseja que ele seja executado, porém, na maioria das vezes o programa utiliza recursos que são dependentes de um determinado sistema operacional.

O Java evita que isso aconteça, pois possui um interpretador chamado de Java Virtual Machine (JVM) capaz de entender o que o programa precisa fazer e traduzir para a linguagem do sistema operacional no qual ela está rodando no momento.

A JVM executa as instruções que estão no *bytecode*. Esse *bytecode* gerado pelo processo de compilação pode ser interpretado pela máquina virtual instalada em sistemas operacionais e computadores diferentes, o que caracteriza o *slogan* muito utilizado pela Sun: *Write once, run anywhere*, ou seja, escreva o código uma vez e o execute em qualquer lugar.

O programa em Java é desenvolvido e compilado uma vez e a JVM se encarrega de interpretá-lo e traduzi-lo para a plataforma onde estão instaladas.

Optou-se pela utilização da plataforma Java neste projeto pelo fato que, além das características e benefícios citados anteriormente, é uma linguagem difundida, estável e madura, que possui vasta documentação, *frameworks* e ferramentas para auxiliar no desenvolvimento de aplicações voltadas para a web.

2.2.2 Java Enterprise Edition (JEE)

Java EE é uma plataforma Java que permite a criação de aplicações em larga escala, dividida em camadas, sólida e com segurança. Tem como objetivo fornecer aos desenvolvedores um conjunto de API (*Application Programming Interface*) que torne mais rápido o desenvolvimento dos sistemas corporativos, menos complexo e com melhor desempenho. Suas especificações são desenvolvidas pela JCP (*Java Community Process*), que é a responsável por todas as tecnologias Java e que ajuda a garantir o padrão, a estabilidade e a compatibilidade com os vários sistemas operacionais.

A Figura 2 ilustra a divisão em camadas de uma aplicação JEE. Na camada do cliente é onde ficam as aplicações que acessam os recursos disponíveis no servidor. Dentro da camada web estão as páginas com conteúdo dinâmico gerado pela camada de negócios, tais páginas podem ser tanto JSP (JavaServer Pages) quanto JSF (JavaServer Faces). Na camada de negócios estão todas as regras e lógica do sistema. Essa camada também é responsável por acessar os dados, estes que por sua vez fazem parte da última camada da aplicação.

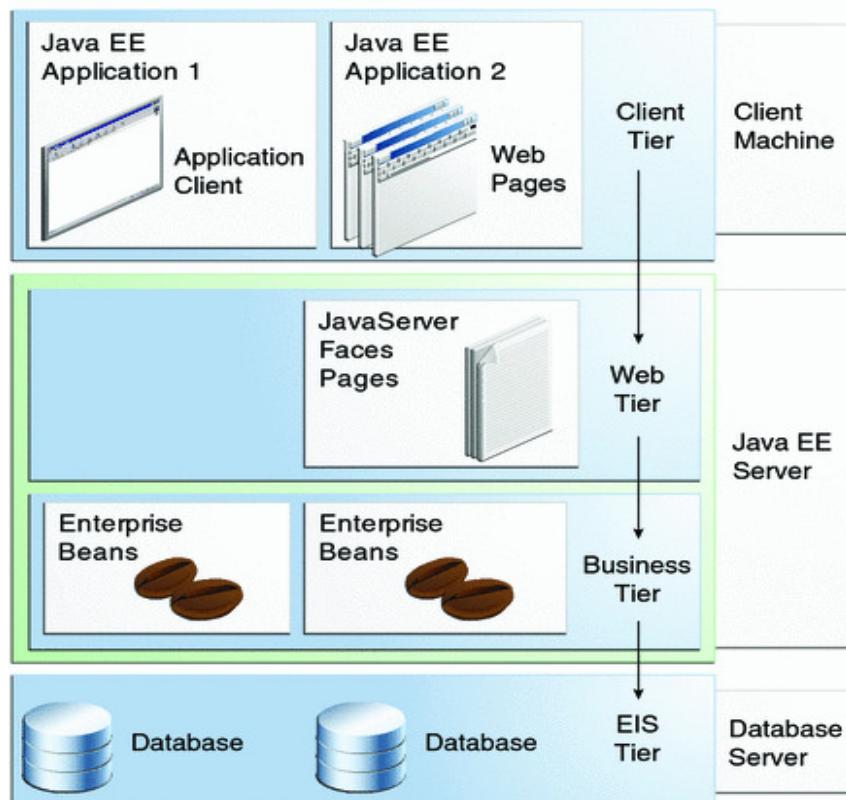


Fig. 2 - Divisão de camadas em uma aplicação Java EE.
Fonte: <http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html>

As soluções JEE são compostas por componentes de softwares que podem ser desenvolvidos separadamente e depois montados e executados em um servidor capaz de gerenciá-los, provendo uma maior escalabilidade e acessibilidade, conforme pode ser observado na Figura 3.

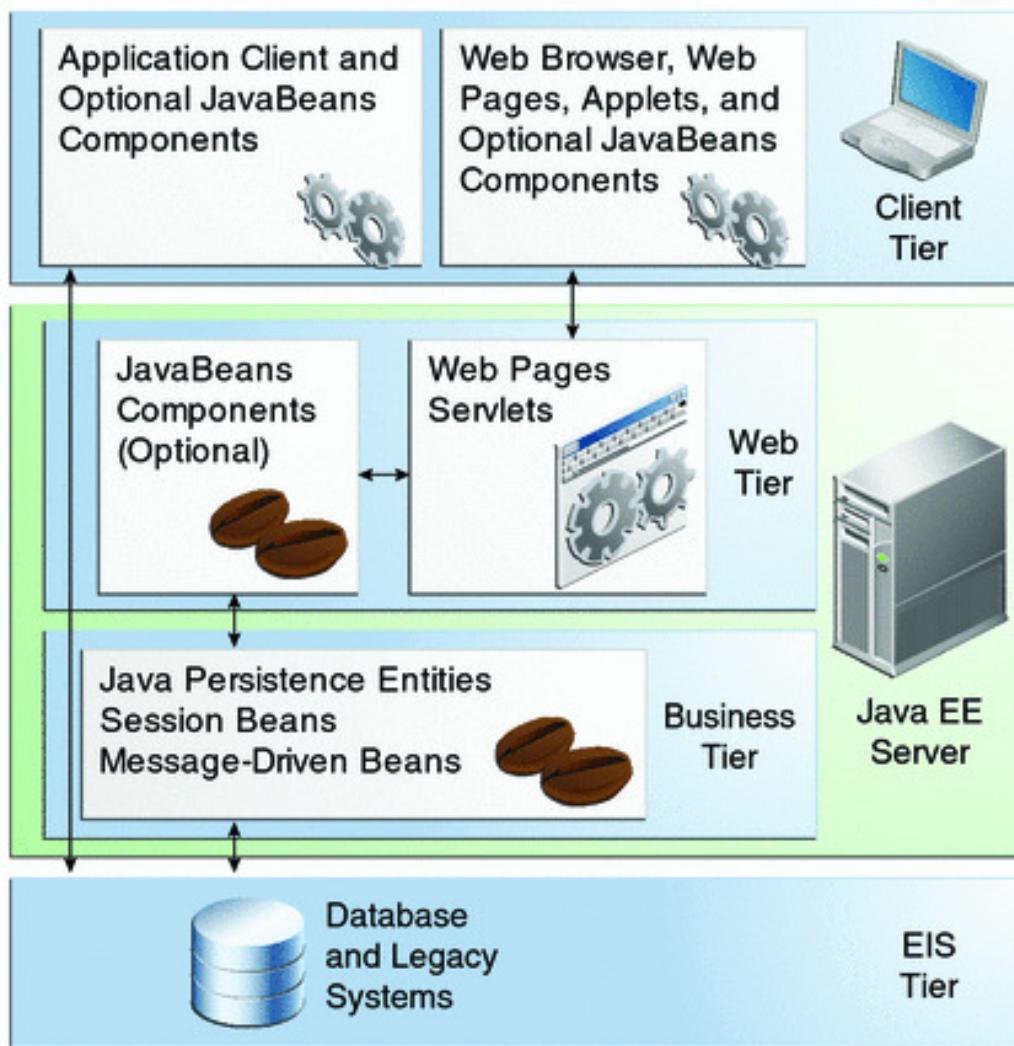


Fig. 3 - Componentes de uma aplicação Java EE.

Fonte: <http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html>

Alguns desses componentes são:

- Aplicações clientes, navegadores e Applets que são executados na máquina do cliente.
- Servlet, JavaServer Pages e JavaServer Faces que são responsáveis por gerar conteúdo dinâmico, estes componentes ficam no servidor web.

- Enterprise JavaBeans que são componentes responsáveis por guardar todas as regras de negócio e lógica do sistema. Conforme definido pela Oracle (2009, s.p.) “Enterprise JavaBeans é uma arquitetura para aplicação distribuída baseada em componentes”.

Por possuir tais componentes, uma das principais adições providas pela especificação JEE é o servidor de aplicação, ou *Container*. Esse *Container* é uma interface entre os componentes e a plataforma específica que suporta esse componente.

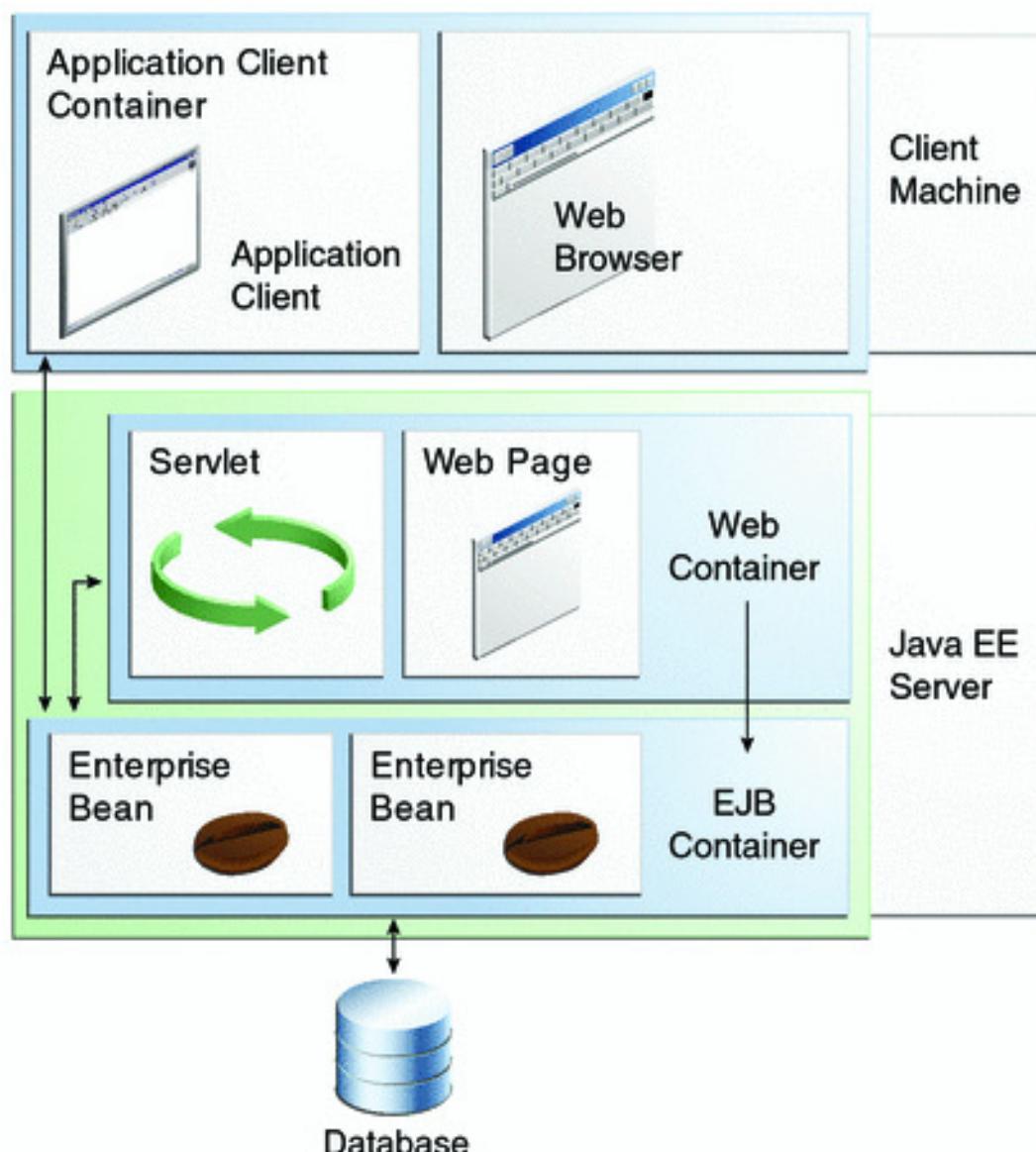


Fig. 4 - Containers de aplicações Java EE.

Fonte: <http://docs.oracle.com/javaee/6/tutorial/doc/bnabo.html>.

A Figura 4 ilustra os tipos de *container*, que são:

- EJB *container*: é um dos *containers* mais conhecidos da arquitetura e especifica um ambiente de execução que tem como objetivo fornecer um padrão rápido e simplificado de desenvolvimento baseado em componentes distribuídos, transacionais, seguros e portáveis.
- WEB *container*: trata-se da implementação do contrato de componentes da web com a arquitetura JEE, fornecendo um ambiente de execução para os componentes do tipo web que incluem desde segurança até gerenciamento do ciclo de vida e transações. Inclui Servlets (classes Java que podem ser carregadas dinamicamente ao serem executadas em um servidor web) disponibilizadas por serviços de rede baseados em *request* e *response* e que utilizam obrigatoriamente o protocolo de transferência de hipertexto (HTTP).
- Applet *container*: é um *container* web que gerencia a execução de applets.

A seguir é apresentado o *framework* Java Server Faces que faz parte da especificação JEE e que foi utilizado na interface com o usuário deste sistema. Esse *framework* foi escolhido por se tratar da tecnologia mais estável e atual para se construir sistemas web em Java, além de apresentar componentes visuais prontos que facilitam o desenvolvimento.

2.2.3 JavaServer Faces

JavaServer Faces ou simplesmente JSF, é um *framework* que permite a elaboração de interfaces de usuário web por meio da ligação entre componentes de um formulário e objetos Java, separando assim a lógica das regras de negócio, das conexões com serviços externos, da navegação e do gerenciamento das configurações. Seus pontos fortes são o grande número de componentes e o *design* bastante flexível, que permitiu a esse *framework* crescer e abranger outras tecnologias.

O JSF está dividido nas seguintes partes: um conjunto de componentes pré-fabricados de interface de usuário, um modelo de programação orientado a eventos e um modelo de elementos que permite que desenvolvedores independentes forneçam componentes adicionais.

Desenvolvido pela *Java Community Process* (JCP), a especificação do JSF possui um amplo conjunto de funcionalidades para o cenário de desenvolvimento onde é aplicada, possibilitando ao desenvolvedor direcionar seu foco para a lógica de negócios e deixar que o *framework* o auxilie nas tarefas comuns na maioria dos sistemas.

De acordo com Geary; Horstmann (2010), todas estas características citadas anteriormente fizeram com que o JSF viesse a ser a tecnologia para desenvolvimento web em Java mais adotada atualmente e tornou-se praticamente uma regra a ser seguida. Sua especificação foi construída para ser um padrão web poderoso e tem se mostrado muito confiável, o que explica essa maciça adoção.

Outro aspecto importante desta tecnologia são os componentes gráficos que simplificam o desenvolvimento de interfaces de usuário, aumentando a produtividade do desenvolvedor e o reaproveitamento do código fonte. O JSF possui uma série de componentes básicos que ajudam na criação das páginas e na geração de conteúdo dinâmico e que em boa parte dos casos é suficiente.

Além destes componentes simples, existem também bibliotecas adicionais mais sofisticadas como, por exemplo, a biblioteca PrimeFaces, a biblioteca ICEfaces e a biblioteca RichFaces mantida pela Red Hat que é utilizada neste projeto e será discutida no próximo tópico.

2.2.4 RichFaces

RichFaces é uma biblioteca de componentes JSF que possui código fonte aberto e que além das ferramentas visuais, também adiciona recursos de AJAX às aplicações sem que o desenvolvedor tenha que recorrer ao JavaScript.

Foi desenvolvido em 2005 por Alexander Smirnov e pela empresa Exadel. Em março de 2007 se uniu ao projeto JBoss, uma divisão da Red Hat, e também a outro projeto, o Ajax4jsf, transformando-se então no JBoss RichFaces.

Uma das grandes vantagens desta tecnologia em relação às concorrentes é a maneira como sua arquitetura AJAX é integrada ao sistema. Ao invés do modelo tradicional orientado aos componentes, o RichFaces permite um suporte à toda a página fazendo com que sua árvore de componentes possa ser completamente atualizada por meio de uma solicitação AJAX ocasionada por um evento.

Atualmente o RichFaces se encontra na versão 4. Essa versão melhorou o suporte a AJAX que está presente desde a versão 3, melhorou também outros recursos para se adaptar ao JSF 2.0, incluindo usabilidade, performance, recursos dinâmicos e seus componentes.

A Figura 5 a seguir ilustra o ciclo realizado pelas requisições AJAX do RichFaces:

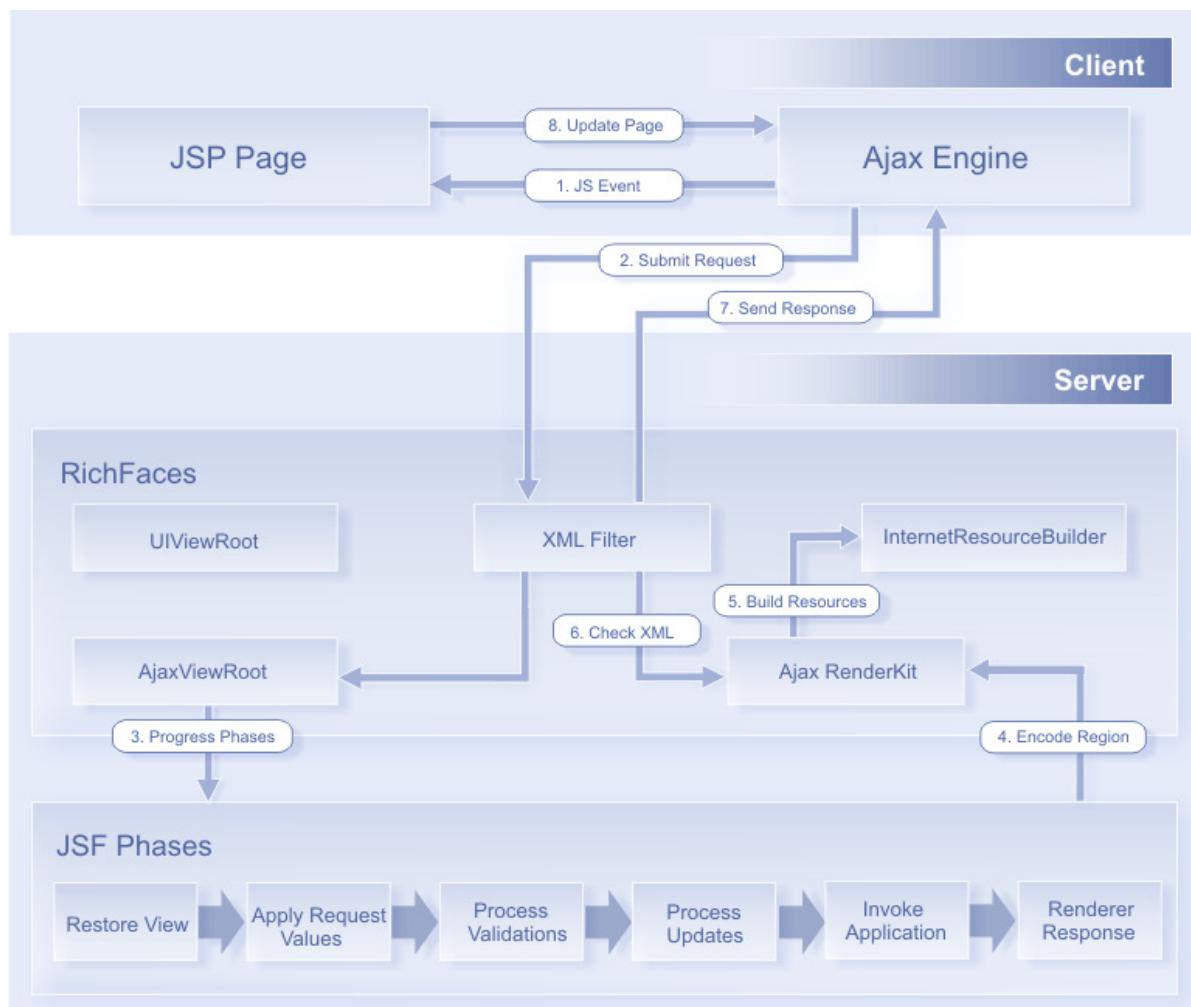


Fig. 5 – Ciclo de vida das requisições AJAX do RichFaces.

Fonte: http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/ArchitectureOverview.html.

A arquitetura do RichFaces é constituída das seguintes ferramentas:

- Filtro AJAX: Utilizado na obtenção de todos os recursos oferecidos pelo RichFaces e deve ser registrado no web.xml. Esse filtro serve para o reconhecimento de múltiplos tipos de requisições HTTP;
- Componentes AJAX: Utilizados para enviar requisições ao servidor do lado do cliente;
- *Container* AJAX: Trata-se de uma interface que descreve uma área em uma página JSF que deve ser decodificada durante uma requisição AJAX;
- Motor JavaScript: o motor JavaScript do RichFaces é executado no lado do cliente. Ele atualiza diferentes áreas da página JSF de acordo com as informações contidas na resposta do AJAX. Este motor oferece uma API para que os desenvolvedores não precisem escrever código JavaScript.

Ao pesquisar por uma biblioteca de componentes para ser utilizada neste projeto, foi necessário encontrar alguma que possuísse as seguintes características: aceitação pela comunidade Java, acabamento visual, documentação disponível e o número de componentes.

O RichFaces atende a todos os requisitos anteriormente citados e se comporta bem tanto no servidor de aplicação utilizado – JBoss Application Server – quanto nos navegadores atuais, garantindo assim a portabilidade da aplicação.

2.2.5 Enterprise JavaBeans

Enterprise JavaBeans (EJB) é uma plataforma para o desenvolvimento de aplicações mais robustas, portáveis, reutilizáveis e escaláveis, utilizando a linguagem de programação Java. Encontra-se atualmente na versão 3.0, mas desde a sua primeira edição, o EJB tem sido apontado como um componente ou *framework* que permite construir soluções corporativas sem ter de reinventar serviços como transações, segurança, persistência automatizada, entre outros que são muito comuns em qualquer sistema (Panda *et al*, 2007), possibilitando ao programador se concentrar melhor na lógica de negócios sem ter que gastar muito tempo na produção de código de infraestrutura.

Segundo Panda *et al* (2007), um EJB é um bloco de código escrito em Java que é executado em um ambiente especializado, chamado de *container EJB*, que fornece inúmeros serviços para os componentes. Estes componentes estão divididos em três tipos: Session Beans, Message-Driven Beans e Entities. A Figura 6 ilustra os tipos de EJB:

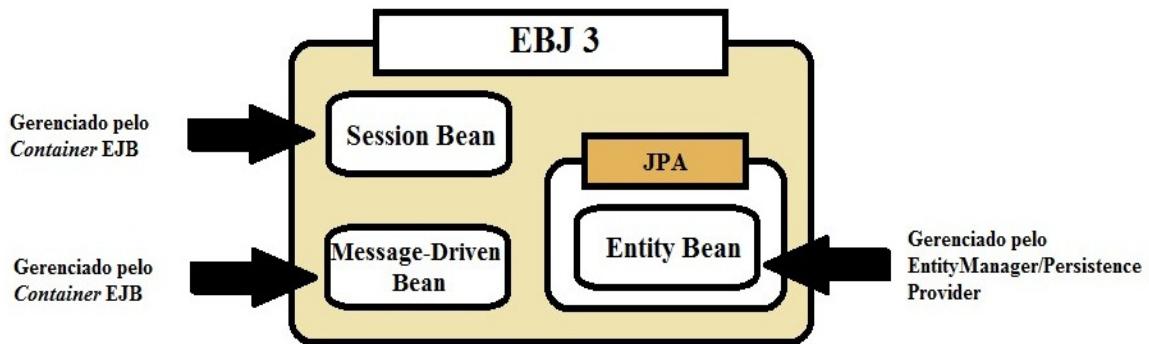


Fig. 6 - Tipos de EJB. *Fonte:* Panda *et al* (2007).

O primeiro tipo, chamado de Session Bean, são componentes que interagem diretamente com os usuários e contém toda a lógica de negócios da aplicação. Um Session Bean representa um único cliente acessando o sistema implantado no servidor e chamando algum de seus métodos. Este sistema pode conter várias sessões, dependendo do número de usuários acessando a aplicação. Cada Session Bean mantém uma sessão interativa apenas para um único cliente e o protege da complexidade da lógica do sistema, pois executa as regras de negócio no lado do servidor. Os Session Beans estão divididos em duas partes: Stateless Session Beans e Stateful Session Beans.

Um Stateless Session Bean não mantém o estado de conversação com o cliente. Quando um cliente invoca os métodos de um Bean Stateless, as variáveis de instância mantêm seu estado apenas durante a chamada do método. Uma vez que o método em questão é concluído, o *container EJB* destrói o Stateless Session Bean.

Os Stateful Session Beans diferentemente dos Stateless, utilizam variáveis de instância que permitem que os dados persistam entre uma e outra chamada de método. O cliente define e altera o estado destas variáveis em uma requisição, e estes dados ficam disponíveis enquanto a sessão do usuário se mantiver ativa. Como o cliente interage frequentemente com seu Bean, esse estado é chamado de estado de conversação.

O segundo tipo é conhecido como Message-Driven Beans, e assim como os Session Beans, são utilizados para processar a lógica de negócio, entretanto, interagem com o usuário de outra maneira. Diferentemente dos Session Beans, os Message-Driven Beans nunca são invocados diretamente. Em vez disso, os MDB são acessados por meio de mensagens enviadas a um servidor que permite a troca assíncrona de informações entre os componentes de um sistema. Os Message-Driven Beans são normalmente utilizados para a integração de aplicações robustas ou processamento assíncrono. Alguns exemplos típicos de servidores de mensagens são IBM WebSphere MQ, Sonic MQ, Oracle Queueing e TIBCO.

O terceiro e último tipo são as Entities ou Entidades, que nada mais são do que objetos Java que podem ser persistidos no banco de dados. Em outras palavras, uma entidade define uma tabela do banco de dados, enquanto cada instância dessa entidade representa uma linha dessa tabela.

Com todas essas características e ferramentas o EJB elevou os padrões de desenvolvimento do lado do servidor provendo muitas vantagens como, por exemplo: simplicidade, integração de soluções, padrão aberto de desenvolvimento, grande aceitação da comunidade incluindo suporte de diversos fabricantes, estabilidade, suporte a falhas, *clustering* e balanceamento de carga.

Este projeto utiliza a versão 3.0 dos EJB. Essa versão se aplica a este projeto por ser mais estável que as anteriores e suportar a utilização de *annotations*, o que simplifica muito o trabalho do desenvolvedor e evita a configuração por meio de arquivos XML, que muitas vezes acaba sendo trabalhosa.

Na próxima seção será explicado o uso do *framework* JBoss Seam, responsável por integrar as tecnologias JavaServer Faces (JSF) e Entreprise JavaBeans (EJB).

2.2.6 JBoss Seam

Como já citado anteriormente, o JSF e o EJB 3 são dois dos *frameworks* utilizados neste trabalho. O EJB se trata de um modelo de componentes para a lógica de negócios e para a persistência de dados, onde ambas ocorrem no lado do servidor. Já o JSF é um excelente modelo para a camada de apresentação, que acontece no lado do cliente.

Embora essas duas tecnologias sejam muito poderosas, nenhuma delas é capaz de solucionar os problemas do paradigma do desenvolvimento web sozinha, e certamente, trabalham melhor em conjunto.

A especificação Java EE não define uma maneira para integrar estes dois modelos de componentes, e prevendo essa situação, seus criadores proveram pontos de extensão padrão para se integrarem com outras soluções.

De acordo com a Seam Reference (2009), o *framework* JBoss Seam unifica o modelo de componentes do JSF e do EJB, eliminando o código de integração e permitindo que o desenvolvedor se dedique exclusivamente à lógica de negócios. Essa unificação trata-se basicamente de contextos e componentes. Os componentes são criados e associados a um contexto. Os contextos são gerenciados pelo *container* de acordo com as transações executadas pelos usuários. A Figura 7 a seguir ilustra como o Seam se encaixa na arquitetura do padrão JEE:

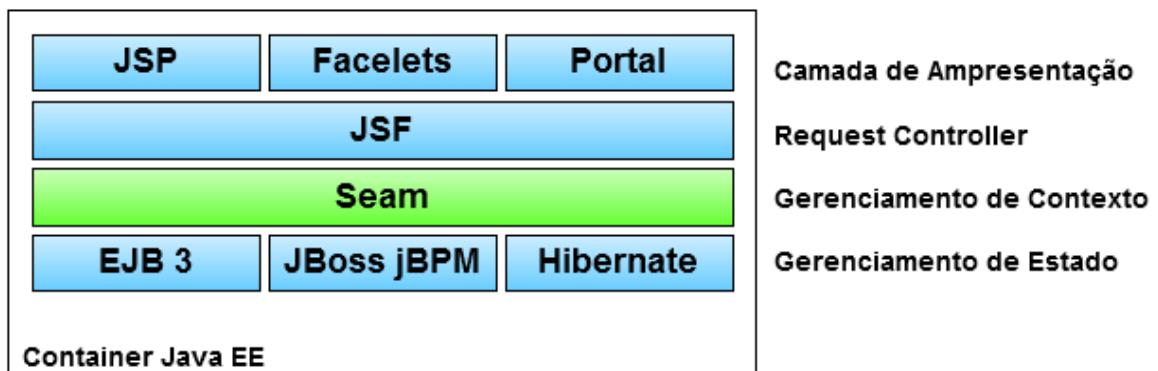


Fig. 7 – Integração do JBoss Seam com a arquitetura Java EE. Fonte: Seam Reference (2009).

Como pode ser observado, o *framewrok* JBoss Seam esta localizado entre as principais tecnologias web baseadas em Java, o que o torna tão poderoso.

2.2.6.1 Contextos

O JBoss Seam possuí 7 contextos diferentes, cada um com suas funcionalidades bem definidas. São eles: Stateless, Event, Page, Conversation, Session, Business Process e Application.

Dentre os contextos mais utilizados estão o Event e o Conversation. O ciclo de vida do contexto Event dura até o fim de uma requisição. No caso desta ser uma requisição JSF, o contexto Event é finalizado após a renderização da página. Já o contexto Conversation é um dos mais úteis do *framework*. Uma conversação é caracterizada por uma unidade de trabalho ou um processo de negócio. O estado de seus componentes é mantido pelo *container* enquanto o cliente realizar requisições no sistema. O Seam permite a utilização de várias conversações em paralelo.

Uma das principais finalidades destes contextos é limitar o uso desnecessário dos recursos computacionais e do sistema, uma vez que o desenvolvedor não precisa se preocupar com a gerência dos componentes. O *framework* garante, por exemplo, a exclusão dos componentes de uma conversação no caso de uma sessão terminar ou expirar.

2.2.6.2 Componentes

Os componentes do Seam são o mesmo componentes do EJB: Session Beans, Message-Driven Beans e Entities. Cada um deles é encapsulado em um dos contextos citados anteriormente.

Todos os componentes do Seam possuem um nome. Ele é definido pela anotação @Name e não está relacionado a nenhum outro nome presente no Java EE por ser específico do JBoss Seam. Nenhuma das outras anotações do *framework* funcionará caso o componente que será utilizado por ele não possuir um nome.

O contexto dos componentes pode ser definido pela anotação @Scope. Caso ele não seja explicitamente definido, o *framework* se encarregará de defini-lo de acordo com o tipo de componente. Os Stateful Session Beans, por exemplo, são associados automaticamente ao escopo Conversation. Enquanto os Stateless Session Beans são armazenados no contexto Stateless. A seguir será discutido o uso do *framework* Lucene, responsável pela indexação e procura dos documentos neste projeto.

2.2.7 Apache Lucene

O Apache Lucene foi criado em 2000 por Doug Cutting. É uma das mais famosas bibliotecas para indexação e consulta de texto. O Lucene é um projeto escrito em Java pela Fundação Apache, que tem seu código fonte aberto e é distribuído sob os termos da *Apache License*. O Lucene funciona com aplicativos JEE e JSE, e também possui integração com outras linguagens de programação como *Perl*, *Python*, *C#*, *.NET*, *PHP*, etc. (Hatcher *et al*, 2009). Para a Fundação Apache (2010), dentre as características do Lucene se destacam:

- Pesquisa ranqueada (os resultados mais relevantes são retornados primeiro).
- Consulta por campos como autor, título e conteúdo.
- Ordenar os resultados por qualquer campo como ano, autor e curso.
- Permite atualizar os índices de consulta e pesquisar simultaneamente.

Sonawane (2009) afirma que as API's do Lucene focam principalmente na indexação e procura de texto, mas também podem ser utilizadas para desenvolver mecanismos de busca para aplicações como clientes de e-mail, busca em banco de dados e sites como TheServerSide, jGuru e LinkedIn, além de empresas como IBM, AOL e HP, que também utilizam o Lucene. A Figura 8 mostra como o *framework* trabalha junto da aplicação e na interação com o usuário.

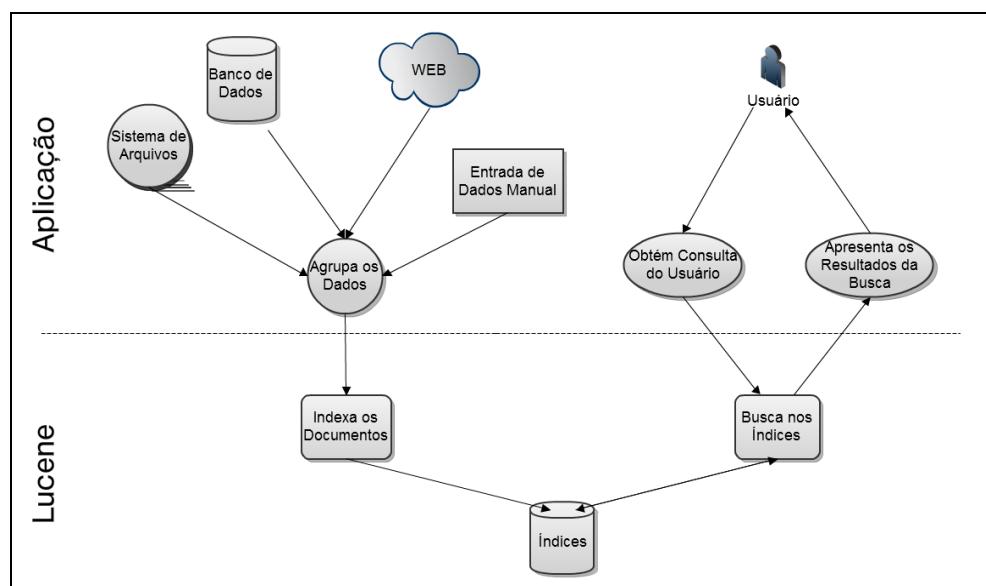


Fig. 8 – Trabalho do framework Apache Lucene junto da aplicação.

Fonte: Elaborado pelos autores baseado em Hatcher *et al* (2009).

Para Hatcher *et al* (2009) o Lucene é a biblioteca de busca e indexação mais utilizada, devido ao fato de possuir sofisticados recursos e um ótimo desempenho. Neste projeto ele atua como um motor de indexação e recuperação de documentos, o que possibilitou, por meio de uma API relativamente simples e eficiente, indexar e persistir a informação de modo que fosse possível recuperá-la posteriormente pela aplicação.

Entretanto, para a realização desta indexação, foi necessário integrar uma ferramenta que convertesse os documentos em PDF para texto puro, já que o Lucene não se responsabiliza por essa tarefa e a deixa a cargo do desenvolvedor. Na seção seguinte discutiremos sobre o *framework* PDFBox responsável pela extração do texto contido nos arquivos em PDF.

2.2.8 Apache PDFBox

Segundo a Apache *Software Foundation* (2012), o PDFBox é uma ferramenta que permite a manipulação de documentos no formato PDF, inclusive a extração do conteúdo textual destes arquivos e se destaca pelas seguintes funcionalidades:

- Extração de texto dos documentos em PDF;
- União de documentos;
- Encriptação das informações nos arquivos;
- Bloqueio dos arquivos;
- Criação de documentos em PDF a partir de arquivos de texto;

A utilização desta biblioteca foi imprescindível neste trabalho, pois, sem ela, não seria possível a extração do texto dos arquivos para indexação. A maneira como esta funcionalidade foi desenvolvida é brevemente apresentada no Capítulo 3 na seção onde os aspectos de implementação são discutidos. Apresenta-se também um tutorial completo da integração entre estas duas tecnologias, Lucene e PDFBox, no Apêndice III. Este tutorial contém exemplos retirados do código fonte do próprio sistema FindPro, com todas as etapas da transformação de um documento em arquivo de texto para ser indexado pela ferramenta de indexação.

Na próxima seção será discutido sobre o PostgreSQL, um sistema gerenciador de banco de dados utilizado para a persistência das informações no sistema de recuperação.

2.2.9 PostgreSQL

Originalmente criado por um centro de pesquisa da Universidade de Berkeley, na Califórnia, o PostgreSQL é um servidor de banco de dados SQL avançado e de código fonte aberto desenvolvido sob a licença Berkeley Software Distribution ou simplesmente BSD.

Atualmente é mantido por um enorme grupo de programadores e colaboradores localizados ao redor do mundo. Está disponível em um grande número de plataformas e possuí mais de 20 anos de desenvolvimento contínuo e aprimorado, fato que agregou ao sistema muitas características avançadas. Dentre elas se destacam:

- Excelente conformidade com os padrões SQL mais atuais;
- Arquitetura cliente-servidor;
- Controle de concorrência;
- Altamente configurável e extensível para muitos tipos de aplicação.
- Grande número de recursos de escalabilidade e extensibilidade.

Para Riggs; Krosing (2010), o PostgreSQL é um sistema gerenciador de banco de dados de propósito geral e dispõe de muitas ferramentas para exercer tal função. É possível utilizar bancos de dados simples e normalizados, ou então, utilizar extensões que permitem ao usuário criar seus próprios tipos de dados.

A aplicação deste gerenciador de banco de dados neste projeto se deve ao fato desta tecnologia se integrar com outras ferramentas também utilizadas neste projeto como o Hibernate – presente no JBoss Seam, além de abranger os recursos necessários à criação de um sistema multiusuário, como por exemplo, controle de transações e de concorrência. Sua licença, a BSD, permite a utilização gratuita em softwares proprietários sem nenhuma restrição inclusive quanto limite para a quantidade de informações que podem ser salvas no banco de dados.

2.3 Engenharia de Software

A engenharia de *software* é uma área da computação que tem como objetivo melhorar a qualidade das aplicações e aumentar a produtividade no processo de desenvolvimento. Ela trata de aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento.

Diante disso, engenharia de software pode ser caracterizada pela utilização de métodos, ferramentas e procedimentos. Segundo Pressman (2002), os métodos proporcionam os detalhes de como construir os softwares. As ferramentas fornecem o apoio tecnológico aos métodos, enquanto os procedimentos fazem a ligação entre os métodos e as ferramentas.

Em uma abordagem de engenharia de *software*, inicialmente deve-se analisar e decompor o problema a ser tratado em partes menores. Para cada uma dessas partes, uma solução deve ser apresentada. Assim que os subproblemas são solucionados um a um, é necessário integrá-los. Para tal, um processo de desenvolvimento de *software* deve ser estabelecido, bem como as ferramentas para o início do trabalho.

2.3.1 Processo de Desenvolvimento de Software

A partir das definições de engenharia de *software* citadas anteriormente, pode-se verificar que as diferentes abordagens quanto às combinações destas características, somadas as particularidades de cada projeto, resultam em um novo conjunto de atividades chamado Processo de Desenvolvimento de Software.

Dentre estes processos, destacam-se dois grupos bem definidos: um deles caracterizado por conter os processos formais e disciplinados, conhecidos como processos pesados, onde o Rational Unified Process (RUP) é um bom exemplo. No outro grupo se encontram os processos mais leves e rápidos, surgidos mais recentemente, dentre os quais se encontra o Extreme Programming (XP).

Mesmo abrangendo uma grande parcela de projetos, os processos expostos anteriormente não se adequam perfeitamente aos projetos de médio porte. Para cobrir essa fatia, um processo que pode ser considerado um meio termo entre o RUP e o XP surgiu há pouco tempo, o ICONIX. Este é, segundo Rosenberg *et al* (2005), um processo intermediário e equilibrado que embora possua muitas características do RUP, evita a sobrecarga imposta pelo mesmo. Além disso, ele é relativamente simples e compacto, como o XP, mas sem descartar a ênfase necessária à análise e ao projeto, fato que ocorre no XP.

2.3.2 ICONIX

Na teoria, cada fase dos processos de desenvolvimento de *software* é extremamente importante, enquanto na prática, nunca se parece ter tempo para a modelagem e análise do sistema. Há sempre uma pressão para que prematuramente se inicie a codificação, uma vez que o trabalho de desenvolvimento de *softwares* tende a ser medido pela quantidade de código produzido.

Para Rosemberg; Stephens (2007), o ICONIX é um processo de desenvolvimento de *software* minimalista e simplificado que recai sobre esta área que está entre a modelagem e a codificação. Sua ênfase incide sobre quando e o que precisa ser feito em determinado momento no ciclo de vida. Ainda de acordo com estes autores, o ICONIX é um processo que está adaptado ao padrão UML, e possui uma característica particular chamada *Traceability of Requirements*, onde os requisitos da aplicação são revistos periodicamente de forma a ficarem sempre alinhados com as necessidades do cliente.

Esse processo pode ser dividido em dois modelos de visão, um dinâmico e outro estático, que se interagem ao longo de quatro fases que serão detalhadas mais adiante.

A Figura 9 apresenta uma ilustração do ICONIX:

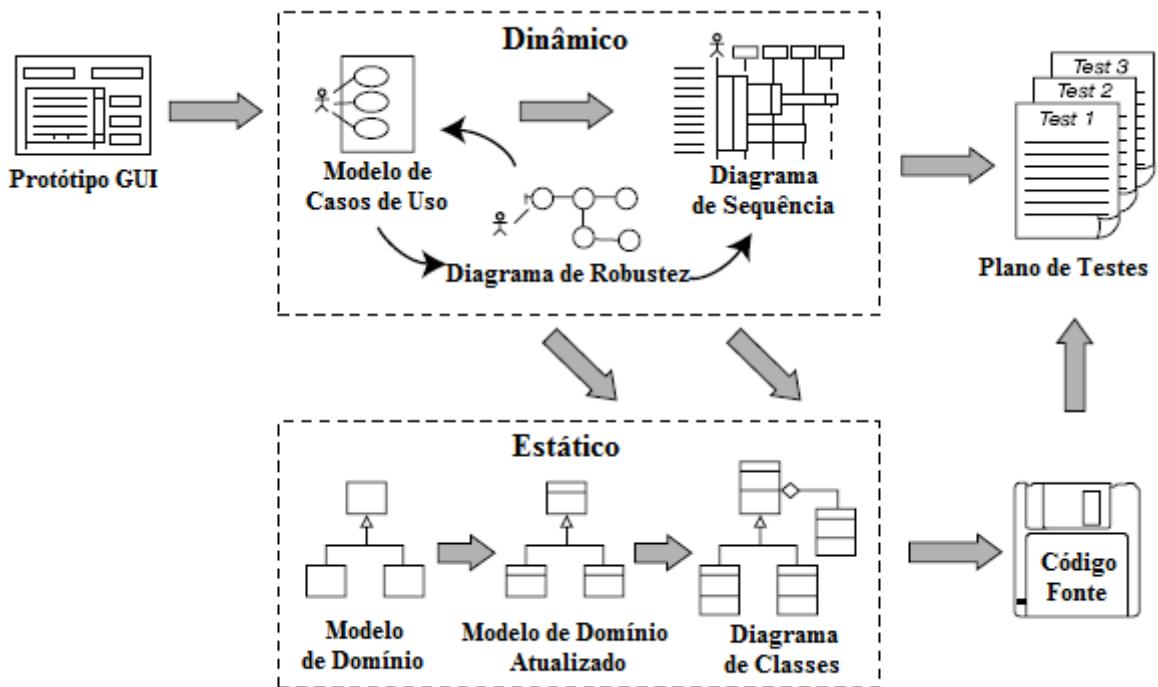


Fig. 9 – Modelo de Visão Dinâmico e Estático. **Fonte:** Rosenberg; Stephens, 2007.

O modelo dinâmico de visão demonstra o processo de desenvolvimento dos sistemas onde não há participação do usuário, como por exemplo, as interações realizadas para se desenvolver um modelo de domínio ou um diagrama de classes. Já no modelo estático, ocorre a modelagem de componentes que representam a utilização da aplicação pelo usuário. Essa utilização é representada por meio de diagramas de casos de uso, diagramas de sequência e diagramas de robustez.

Rosemberg; Stephens (2007) afirmam também que, além da divisão em dois modelos de visão, ocorre também a divisão do processo ICONIX em quatro fases, onde os requisitos do sistema são periodicamente revisados.

A primeira fase é chamada de Análise de Requisitos e tem como objetivo geral reconhecer os objetos do problema real e as relações que possam existir entre eles. Essas relações são geralmente representadas por um diagrama de classes de alto nível e conhecidas como modelo de domínio.

Nesta fase é importante apresentar para o cliente uma forma sucinta de visualização das funcionalidades propostas para o sistema, e também desenvolver um diagrama de casos de uso para identificar todos os atores envolvidos na aplicação.

Na segunda fase, conhecida como Análise e Projeto Preliminar, devem-se descrever os casos de uso com os fluxos principais das ações, podendo ou não conter os fluxos alternativos e as exceções. Aborda-se também a análise de robustez, onde as entidades de cada caso de uso são descobertas e atualizadas no diagrama de classes e modelo de domínio.

Na fase seguinte, Projeto Detalhado, atribui-se comportamento aos casos de uso por meio de diagramas de sequência, identificando a comunicação entre os diversos objetos.

A última fase, ou Implementação, não possuí muito suporte pelo ICONIX, pois, segundo alguns autores, esta etapa trata de aspectos particulares de cada desenvolvedor, como por exemplo, a linguagem de programação utilizada ou a plataforma. Entretanto ele aborda algumas orientações quanto aos procedimentos do desenvolvimento como:

- Produzir diagramas que ajudem na programação;
- Gerar código da aplicação;
- Realizar testes de integração e testes unitários;
- Realizar testes de sistema e testes de aceitação.

A adoção da engenharia de *software* e suas boas práticas garantem a prevenção de uma série de problemas que só seriam percebidos durante o uso de uma aplicação. No mundo empresarial moderno, as metodologias de desenvolvimento pouparam a equipe de programação de contratemplos, evitando recodificação, multas por prazos mal estipulados e desentendimento entre clientes e desenvolvedores, situações muito ruins para os negócios e principalmente para o usuário final.

Esses motivos levaram a utilização do processo de desenvolvimento ICONIX neste trabalho. Reconhecido como um processo prático, ele especifica de forma clara e objetiva o que deve ser feito e quando deve ser feito, permitindo que o foco seja direcionado para a solução e amenizando problemas que ocorrem em projetos com prazos curtos e com poucos desenvolvedores.

3 QUADRO METODOLÓGICO

Neste capítulo serão discutidas as etapas do desenvolvimento deste trabalho, bem como o papel dos participantes e a metodologia de desenvolvimento do sistema de recuperação textual. Esses procedimentos foram realizados com base na metodologia de pesquisa aplicada e nas etapas do processo de desenvolvimento de *software* ICONIX.

Ao longo desta discussão, os passos de cada fase serão representados por meio de alguns diagramas, contudo, foge do escopo deste capítulo a inclusão de todos os artefatos gerados durante a modelagem do sistema, entretanto, podem ser encontrados no apêndice 1. A seguir será apresentado o tipo de pesquisa no qual esse trabalho se baseia.

3.1 Tipo de Pesquisa

Para Salomon (1999), “pesquisa é um trabalho empreendido metodologicamente, quando surge um problema, para o qual se procura uma solução adequada de natureza científica”. Já a pesquisa aplicada, segundo Barros; Lehfeld (2000), tem como motivação a necessidade de produzir conhecimento, produtos e processos com finalidades imediatas, ou seja, o fruto de seu desenvolvimento é empregado prontamente no problema para qual foi designado.

Com relação a sua estrutura, as pesquisas aplicadas apresentam-se da seguinte maneira: fundamentação teórica, metodologia de pesquisa e análise e discussão de dados. O referencial teórico tem como objetivo servir de base para a análise e discussão dos dados. Estes dados por sua vez são coletados e aplicados a partir de uma metodologia de acordo com os objetivos, características, objeto de estudo, e do contexto de investigação da pesquisa Marconi; Lakatos (2002).

A partir das informações anteriormente discutidas, e observando que este projeto foi dirigido à solução de um problema real, que neste caso é a procura nos trabalhos de conclusão de curso, e que para a resolução deste problema foram empregados instrumentos científicos como fundamentação teórica e metodologias de pesquisa, constatou-se que este projeto é uma pesquisa aplicada. Sendo assim, este trabalho apresenta uma alternativa à busca dos documentos gerados na universidade, podendo inclusive ser aplicada a outros tipos de procura em documentos do tipo textual. Na próxima seção será abordado o contexto desta pesquisa.

3.2 Contexto de Pesquisa

Esta pesquisa foi realizada observando as atividades de uma biblioteca de uma instituição de ensino superior localizada no sul de Minas Gerais. Essa biblioteca conta com um grande acervo de trabalhos acadêmicos das mais variadas áreas do conhecimento e que podem ser emprestados pelos usuários. Quanto ao acesso aos livros e projetos, é possível realizar uma procura de duas maneiras: presencialmente ou por um sistema disponibilizado no portal do aluno.

A pesquisa realizada pelo portal do aluno possibilita a procura pelos documentos utilizando o seu título, autores ou assunto. Além da pesquisa o sistema oferece a possibilidade do usuário fazer uma reserva do trabalho desejado, mas no caso da existência de uma fila de espera, não é possível saber quantas pessoas já aguardam pela obra.

Na pesquisa presencial existem bibliotecárias que ajudam na procura pelos trabalhos. A biblioteca também conta com um sistema que possui informações sobre a fila de espera e a localização dos livros e projetos nas prateleiras fazendo com que essa pesquisa seja mais viável quando se tem interesse em pesquisar no conteúdo dos documentos.

A biblioteca permite o acesso da comunidade em geral ao material bibliográfico, mas somente alunos, professores e colaboradores devidamente cadastrados no sistema da instituição podem realizar um empréstimo. O número de trabalhos a serem emprestados por cada usuário é de no máximo três e o prazo de devolução não pode ultrapassar sete dias corridos, caso isso ocorra há incidência de multa. Também não é permitida a retirada de obras que possuam somente um exemplar ou que estejam sendo muito utilizadas, como obras de referência, encyclopédias, dicionários, atlas e terminologias. A seguir serão apresentados os sujeitos que participarão deste projeto como também as funções que eles desempenharam.

3.3 Metodologia de Desenvolvimento

Neste capítulo serão descritas todas as etapas do desenvolvimento do sistema de recuperação textual FindPro. Esses procedimentos foram realizados de acordo com as fases do processo de desenvolvimento de software ICONIX que está dividido em quatro etapas: análise de requisitos, análise do projeto preliminar, análise do projeto detalhado e desenvolvimento.

3.3.1 Análise de Requisitos

Nesta fase os dados coletados por meio dos instrumentos de pesquisa foram avaliados para a identificação dos casos de uso do projeto. Estes casos de uso definiram funcionalidades do sistema, onde cada um deles foi representado por fluxos de eventos. Os nomes dos casos de uso descrevem o que ocorre na aplicação quando os mesmos são executados pelo usuário, e sua união constitui todas as maneiras possíveis de utilizar este sistema. O diagrama da Figura 10 representa os casos de uso descobertos para o sistema de recuperação FindPro, na qual é possível ter uma ideia de cada funcionalidade da aplicação apenas observando o nome de cada caso.

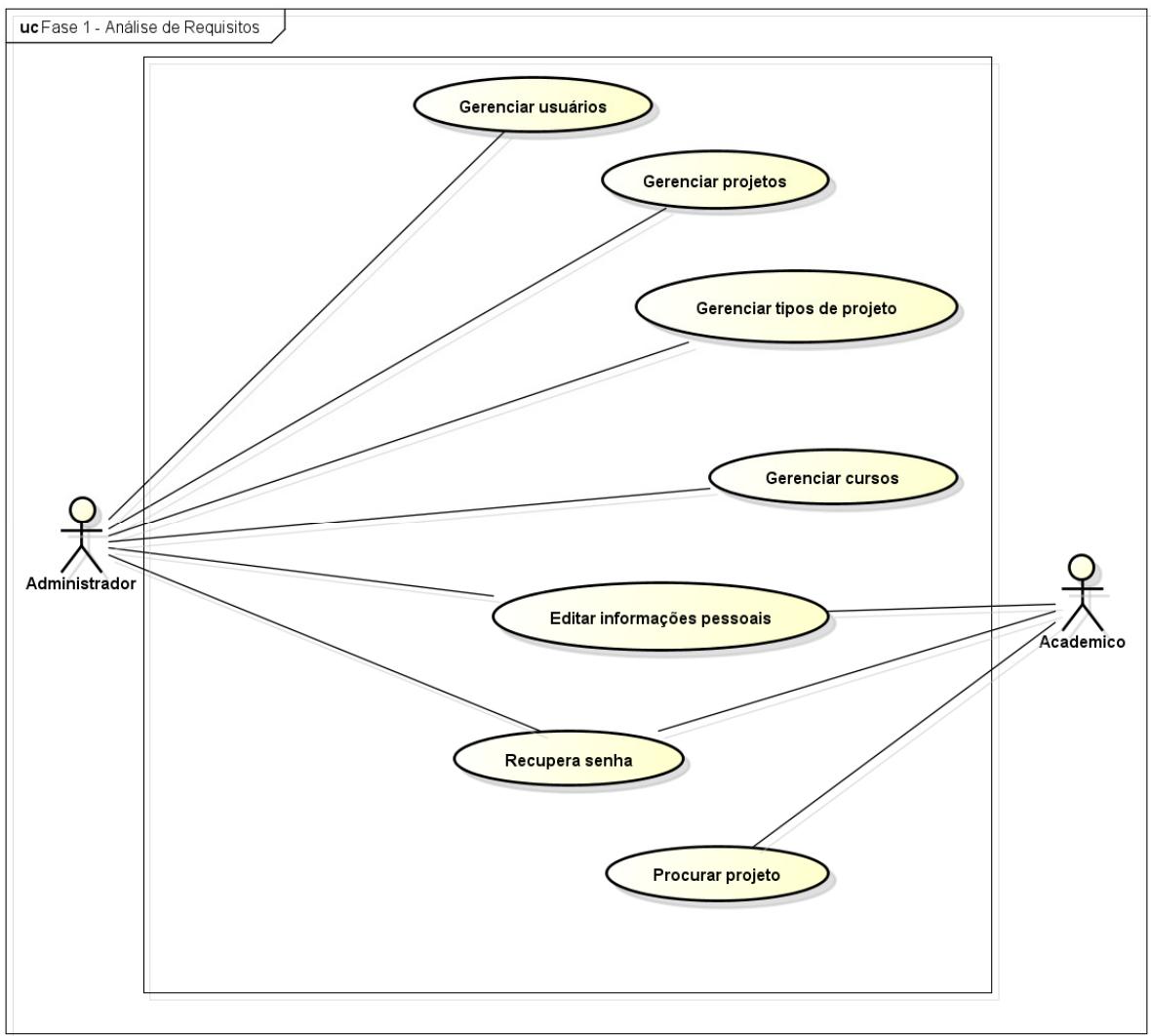


Fig. 10 – Diagrama de Casos de Uso do sistema de recuperação textual FindPro.

Fonte: Elaborado pelos autores.

A figura anterior permite verificar que os seguintes requisitos funcionais foram definidos para o sistema:

- Gerenciar usuários: Este caso de uso abrange todas as tarefas ligadas à manutenção de usuários da aplicação, como por exemplo, inserção, edição dos dados e inativação de conta.
- Gerenciar projetos: Representa as atividades relacionadas com o cadastro e o preenchimento dos elementos pré-textuais dos projetos acadêmicos, bem como uma possível remoção ou inativação dos mesmos. Este caso de uso é um dos pontos centrais do sistema de recuperação textual, pois desencadeia o processo de indexação dos arquivos realizado pelo *framework* Apache Lucene, para que posteriormente, as informações possam ser recuperadas pelos usuários.

- Gerenciar tipos de projeto: Este caso de uso representa a manutenção dos tipos de projetos que podem ser cadastrados no banco de dados, como teses, artigos ou monografias.
- Gerenciar cursos: Abrange as tarefas relacionadas com a inserção, edição ou inativação das coleções dos projetos acadêmicos. Essas coleções representam os cursos de onde os trabalhos se originaram.
- Editar informações pessoais: Refere-se à edição dos dados do próprio usuário, como por exemplo, mudança de endereço de e-mail ou *username*.
- Recuperar senha: Permite aos usuários definirem uma nova senha, caso tenham perdido ou esquecido sua senha antiga.
- Procurar projeto: Este é um dos casos de uso mais importantes da aplicação, sua principal atividade constitui na busca e recuperação dos trabalhos acadêmicos presentes na base de dados do sistema.

O diagrama de casos de uso também permitiu a descoberta dos atores, que constituem outra característica importante da aplicação. Os atores especificam elementos externos ao sistema, interagindo com determinado caso de uso, como por exemplo, uma secretária realizando o cadastro de um aluno. Nesta etapa, dois tipos de atores foram identificados:

- Administrador: a este tipo de ator, foram atribuídas funcionalidades como o cadastro de projetos, a inserção de novos usuários e também o gerenciamento dos cursos e tipos de trabalho.
- Acadêmico: as atividades realizadas por este tipo de usuário incluem a alteração dos dados cadastrais, a recuperação de sua senha e principalmente a pesquisa por projetos.

A partir da identificação dos casos de uso e dos atores que interagem com o sistema, iniciou-se a elaboração dos fluxos de eventos, que são uma descrição textual das maneiras pelas quais os *use cases* podem ser realizados. Cada um destes fluxos de eventos foi composto por um conjunto de cenários que podem ocorrer de diversas maneiras.

Estes cenários por sua vez, descrevem modos alternativos de comportamento do sistema, suas falhas ou casos excepcionais, como também as decisões do usuário e a resposta da aplicação. O fluxo de eventos representado pela Figura 11 a seguir, detalha o caso de uso Gerenciar Projetos.

Caso de uso relacionado: Gerenciar projetos	
Descrição:	Cadastrar um novo projeto no sistema.
Autor(es):	Administrador.
Pré-condições:	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Projetos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar projeto.</p> <p>4 – Ator clica em “Cadastrar projeto”.</p> <p>5 – Sistema exibe janela para cadastrar projeto.</p> <p>6 – Ator informa título, autores, orientadores, local, data, palavras chave, tipo, curso, adiciona o arquivo para upload e clica em gravar.</p> <p>7 – O sistema faz a validação das informações, persiste os dados e indexa o arquivo.</p> <p>8 – O sistema exibe uma mensagem de sucesso.</p>
Fluxo Alternativo:	<p>No Item 7 do fluxo principal caso a validação encontre algum erro nos dados:</p> <p>7.1 – O sistema exibe mensagem de erro informando os campos que foram preenchidos incorretamente.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 11 – Fluxo de Eventos – Cadastrar Projeto. **Fonte:** Elaborado pelos autores.

No modelo de programação orientada a objetos, as interações realizadas pelos atores em cada um dos fluxos de eventos, resultam na manipulação de entidades pelo sistema. Estas entidades, nada mais são do que representações de objetos do mundo real através de linguagem de programação, como os arquivos dos projetos acadêmicos representados pela classe Projeto, ou os acadêmicos, representados pela classe Usuario. A figura 12 ilustra o conjunto de entidades definidas para a aplicação.

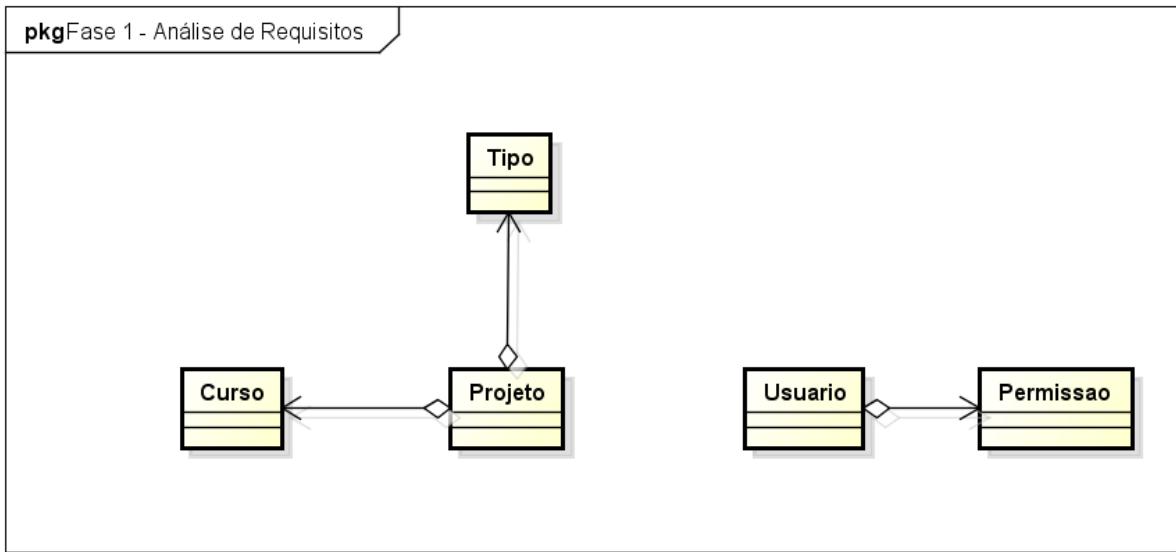


Fig. 12 – Modelo de Domínio do sistema de recuperação FindPro. **Fonte:** Elaborado pelos autores.

Como pode ser observado na figura 12, o sistema trabalha com as seguintes entidades:

- **Tipo:** Esta entidade representa uma categoria entre uma série de tipos de projetos possíveis, como por exemplo, monografias, artigos ou teses.
- **Curso:** Representa o curso dos autores do projeto, como Sistemas de Informação ou Letras.
- **Projeto:** Esta é a principal entidade manipulada pelo sistema. Tem como objetivo representar os projetos acadêmicos que serão recuperados pela aplicação.
- **Usuario:** Representa um usuário do sistema, que pode ser tanto um Administrador, quanto um Acadêmico, de acordo com as permissões a ele atribuídas.
- **Permissao:** Define os privilégios dos usuários que utilizam o sistema.

Este conjunto de entidades citado anteriormente é chamado de **Modelo de Domínio**, cuja principal função foi fornecer uma visão de alto nível das principais classes da aplicação. Assim que estas classes foram definidas, os artefatos foram revisados e finalizou-se a primeira fase do processo ICONIX, porém, a documentação gerada nesta etapa serviu de insumo para a **Análise e Projeto Preliminar**, que será abordada na próxima seção.

3.3.2 Análise e Projeto Preliminar

Nesta etapa do desenvolvimento, iniciou-se a criação dos diagramas de robustez, tendo como base os casos de uso e fluxos de eventos construídos na fase anterior. Os diagramas de robustez auxiliaram na transformação da descrição textual das funcionalidades do sistema, descobertas pelos casos de uso, em estereótipos que permitiram identificar os objetos de fronteira, de controle e de entidade. A Figura 13 ilustra o diagrama de robustez gerado a partir do fluxo de eventos Gerenciar projetos (Novo Projeto), já exibido anteriormente na Figura 11.

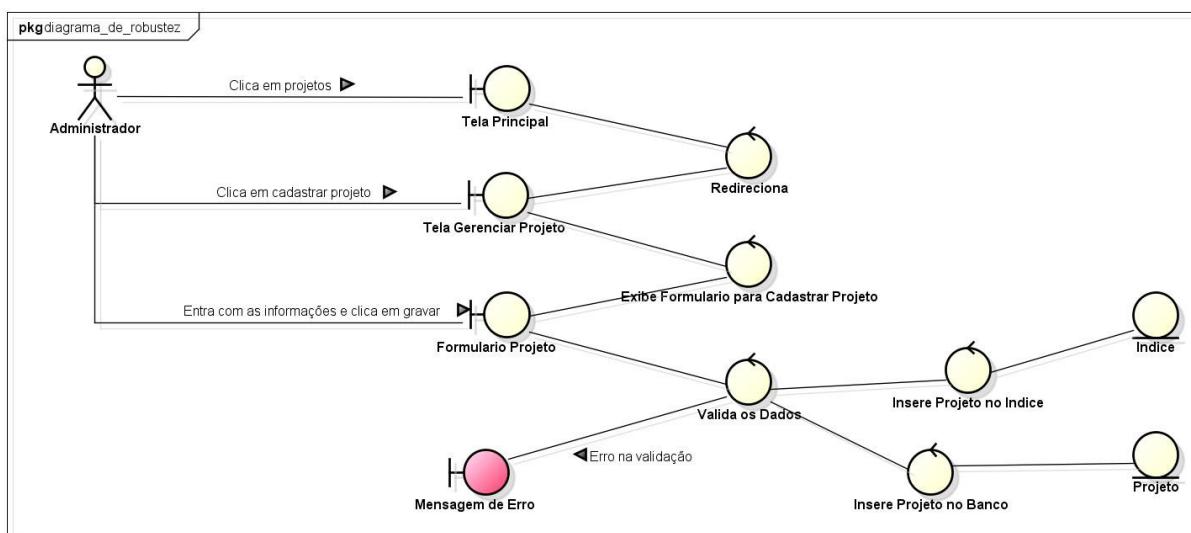


Fig. 13 – Diagrama de Robustez Gerenciar Projeto (Cadastrar Projeto). **Fonte:** Elaborado pelos autores.

Com a identificação das classes de fronteira, partiu-se para a prototipação da interface gráfica da aplicação, onde foi construído um esboço das janelas com as quais o usuário irá interagir, além de outros aspectos como usabilidade e navegação. A Figura 14 ilustra a classe de fronteira Novo Projeto, que conforme descrito anteriormente, possibilita que o administrador cadastre um novo projeto e ao clicar em salvar, persista-o no banco de dados e inicie o processo de indexação do documento.

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)
[Home](#) > [Projeto](#) > Cadastrar Projeto

Projeto _____

* campos obrigatórios

*Título:

*Autor(es):

*Orientador(es):

*Palavras Chave:

*Local:

*Data:

*Tipo:

*Curso:

*Arquivo:

Fig. 14 – Formulário de cadastro de novos projetos.

Fonte: Elaborado pelos autores.

A revisão do modelo de domínio foi outra atividade desta etapa, que passou a contar com os atributos de cada entidade do sistema. Este modelo de domínio, agora atualizado, pode ser visto mais adiante na Figura 15.

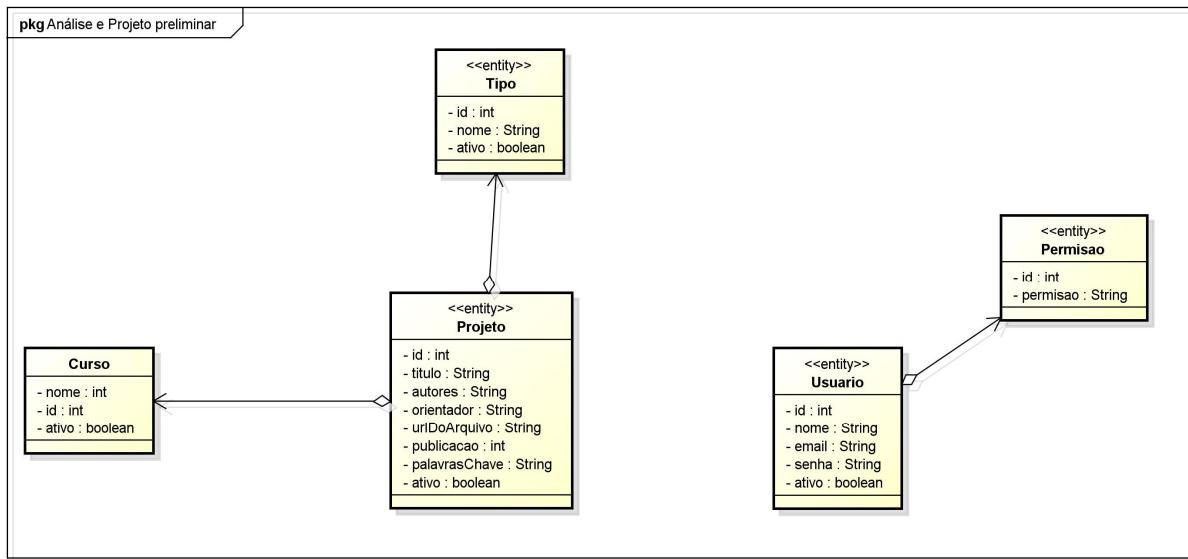


Fig. 15 – Modelo de Domínio atualizado. **Fonte:** Elaborado pelos autores.

Agora com o modelo de domínio atualizado, o próximo passo foi gerar o banco de dados da aplicação. O *script* para a geração do banco, pode ser encontrado no Apêndice 1 junto aos outros artefatos desenvolvidos no quadro metodológico. A Figura 16 representa as tabelas e relacionamentos existentes na base de dados do sistema, chamada findpro.

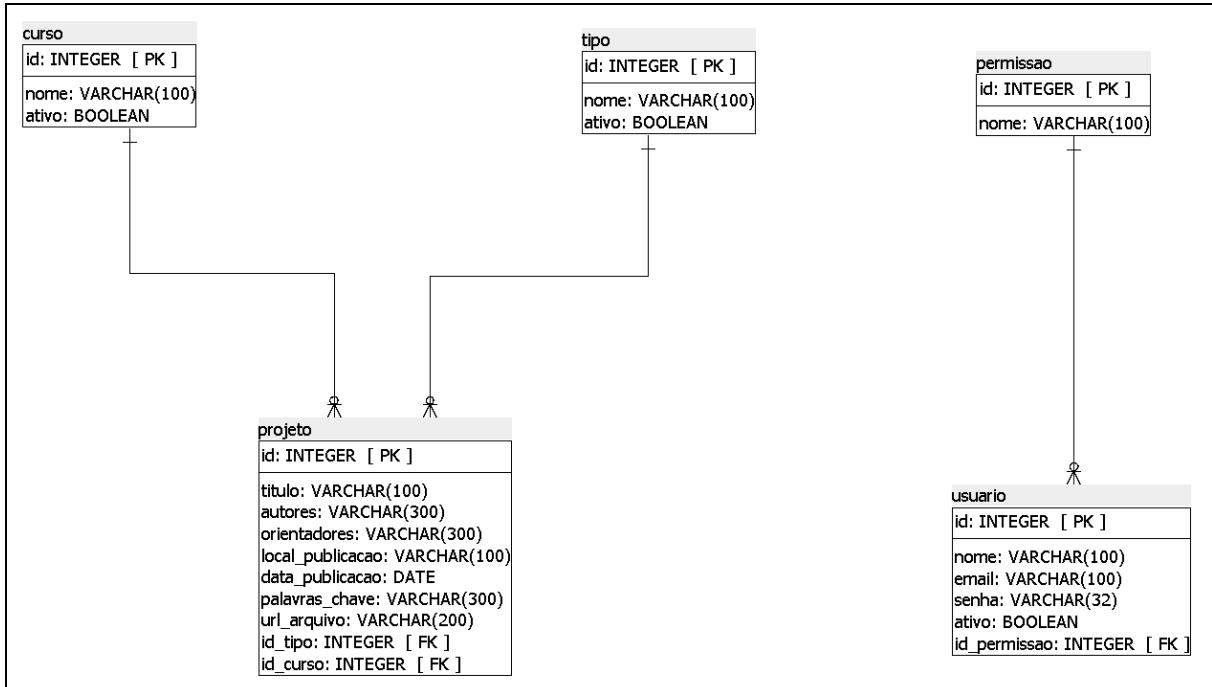


Fig. 16 – Tabelas do banco de dados findpro. **Fonte:** Elaborado pelos autores.

Terminada a fase de análise e projeto, iniciou-se a fase do projeto detalhado, que será discutida na próxima seção.

3.3.3 Projeto Detalhado

Nessa fase, foram desenvolvidos os diagramas de sequência, cuja função foi detalhar como as mensagens entre os objetos são trocadas no decorrer do tempo da execução de um caso de uso. Estas mensagens também serviram para atualizar o modelo de domínio com as operações encontradas, transformando-o então no diagrama de classes completo. A Figura 17 mostra o Fluxo de Eventos Cadastrar Projeto e logo em seguida seu respectivo Diagrama de Sequência, onde o ator faz o cadastro de um novo projeto.

Caso de uso relacionado: Gerenciar projetos	
Descrição:	Cadastrar um novo projeto no sistema.
Ator(es):	Administrador.
Pré-condições:	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Projetos no menu principal. 2 – Sistema exibe tela para gerenciar projeto. 4 – Ator clica em “Cadastrar projeto”. 5 – Sistema exibe janela para cadastrar projeto. 6 – Ator informa título, autores, orientadores, local, data, palavras chave, tipo, curso, adiciona o arquivo para upload e clica em gravar. 7 – O sistema faz a validação das informações, persiste os dados e indexa o arquivo. 8 – O sistema exibe uma mensagem de sucesso.</p>
Fluxo Alternativo:	<p>No Item 7 do fluxo principal caso a validação encontre algum erro nos dados: 7.1 – O sistema exibe mensagem de erro informando os campos que foram preenchidos incorretamente.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 17 – Fluxo de Eventos Cadastrar Projeto. **Fonte:** Elaborado pelos autores.

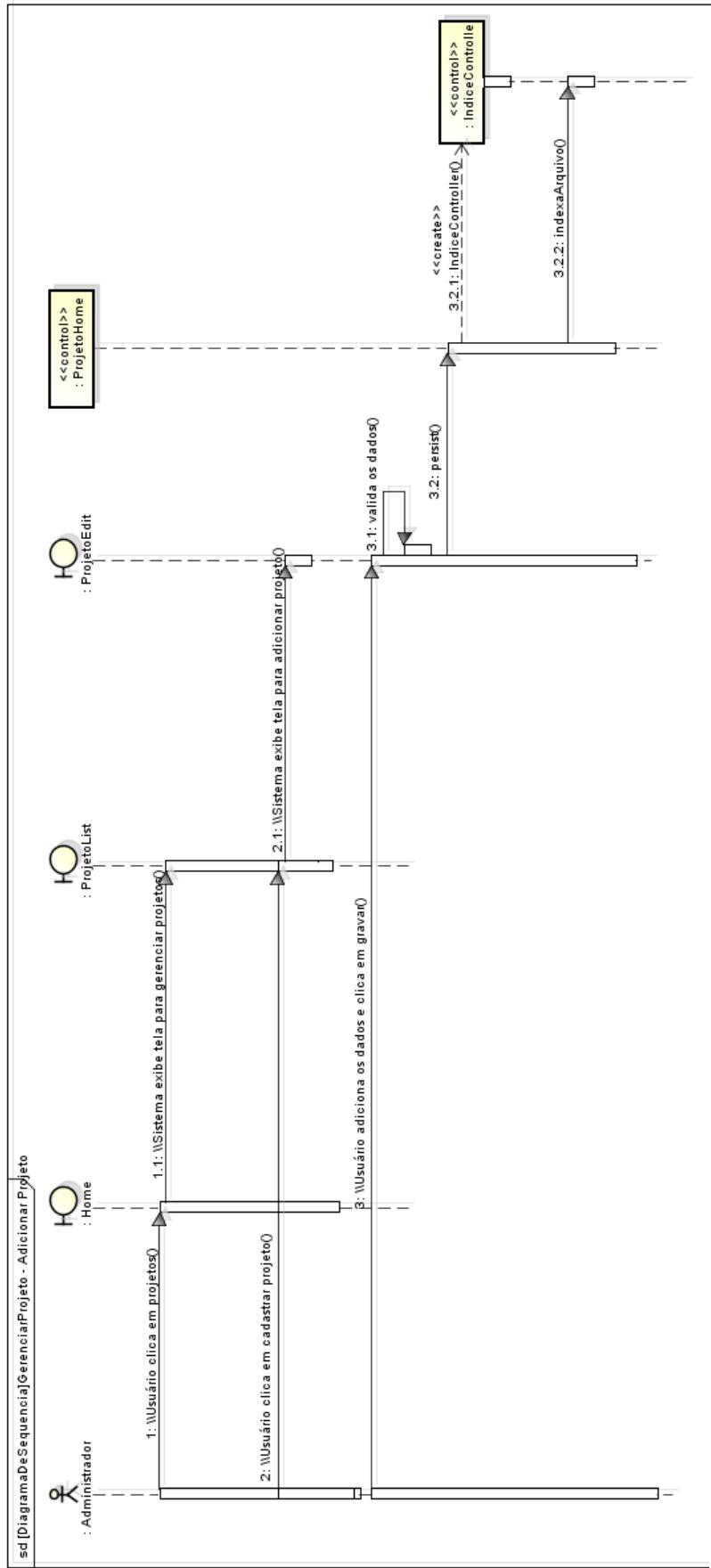


Fig. 18 – Diagrama de Sequência Cadastrar Projeto. Fonte: Elaborado pelos autores.

E então com as operações bem definidas, o modelo de domínio foi atualizado para o diagrama de classes. A Figura 19 exibe um fragmento do diagrama de classes da aplicação.

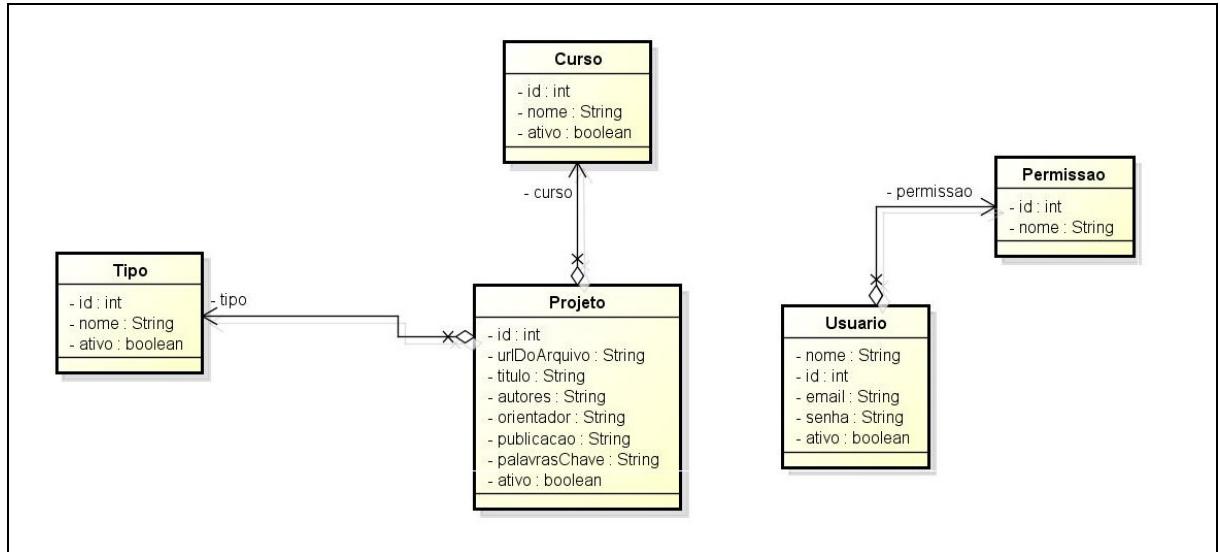


Fig. 19 – Fragmento do Diagrama de Classes. **Fonte:** Elaborado pelos autores.

A Figura acima ilustra as entidades manipuladas pelo sistema de recuperação FindPro. Com o diagrama de classes finalizado, partiu-se para o desenvolvimento do sistema utilizando uma linguagem de programação, neste caso Java. Esta fase é chamada de Implementação e será discutida na próxima seção.

3.3.4 Implementação

Esta fase envolveu as atividades de codificação, compilação e integração da aplicação. Na codificação os artefatos foram traduzidos em num programa, utilizando a linguagem de programação Java e os outros *frameworks* já mencionados no capítulo 2.

Outro aspecto muito importante desta fase foi a transformação dos protótipos em páginas web. A Figura 20 seguir apresenta a página de cadastro de projetos, que é vista somente por administradores.

The screenshot shows a web-based application interface for adding a new project. At the top, there's a navigation bar with links for 'Pagina Inicial', 'Usuarios', 'Cursos', 'Tipo de Projetos', 'Projetos', 'signed in as: cesar@findpro.com', and 'Logout'. The main area is titled 'Add Projeto' and contains the following fields:

- Titulo ***: FindPro - Sistema de Recuperação Textual em Publicações Acadêmicas
- Tipo: ***: Trabalho de Conclusão de Curso
- Autores ***: Cesar Augusto Couto Santps; João Paulo Pereira
- Orientadores ***: Roberto Ribeiro Rocha
- Local publicacao ***: Pouso Alegre - MG
- Data publicacao ***: 01/12/2012
- Palavras chave ***: Recuperação Textual. JBoss Seam. Apache Lucene.
- Url arquivo**: A file upload field with a 'Escolher Arquivo' button.

Fig. 20 – Página de cadastro de novos projetos. **Fonte:** Elaborado pelos autores.

Na página de cadastro anteriormente apresentada, outro aspecto importantíssimo deve ser destacado: a conversão dos documentos em PDF para texto puro. A responsável por essa tarefa é a ferramenta PDFBox, já discutida no capítulo 2. Assim que o documento é submetido pelo usuário no sistema, entra em cena uma das funcionalidades deste *framework* que retira todo o conteúdo textual por meio da classe PDFTextStripper e de seu método getText(). Todas as etapas do dessa conversão são explicadas com detalhes no Apêndice III.

O código fonte deste sistema pode ser encontrado na mídia que acompanha este trabalho, juntamente com um tutorial do *framework* Lucene. Outra opção para a obtenção do sistema é por meio do servidor de controle de versão hospedado no Google Code, que pode ser acessado através do endereço <http://code.google.com/p/findpro/>.

Vale lembrar que para colocar a aplicação em funcionamento, deve-se usar um servidor de aplicações, que no caso deste projeto foi o JBoss Application Server 5.1.

4 DISCUSSÃO DE RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos pela pesquisa e desenvolvimento do sistema de recuperação textual FindPro. Esta discussão será realizada de diferentes perspectivas com o objetivo de demonstrar os pontos fortes e fracos de cada uma delas.

Antes mesmo da escolha do tema deste projeto, nosso objetivo era desenvolver um sistema com ferramentas e tecnologias que não foram abordadas pela grade do curso de Sistemas de Informação, embora, os pré-requisitos para que aprendêssemos a utilizá-las tenham sido nos ensinado ao longo deste tempo.

Decidimos então, desenvolver um sistema que pudesse ajudar, de alguma maneira, os acadêmicos que necessitavam de um acesso mais fácil, rápido e dinâmico às publicações desenvolvidas na universidade e, a partir daí, iniciamos a pesquisa sobre teorias e tecnologias que atendessem aos conceitos de acessibilidade e disponibilidade de informações.

Constatamos que somente um sistema web atenderia às questões da comodidade e facilidade no acesso aos projetos acadêmicos. Os sistemas baseados nesta plataforma usufruem de uma série de protocolos e métodos que possibilitam a troca de informações entre dispositivos conectados à internet e combinam perfeitamente com o conceito de disponibilidade dos documentos, permitindo que qualquer pessoa com acesso a internet possa encontrar o trabalho que procura.

Desenvolver um sistema utilizando tecnologias com suporte a plataforma *web* como os EJB, JSF entre outras foi, então, definido como um dos objetivos específicos deste projeto e, para alcançá-lo, utilizamos o *framework* JBoss Seam, que nos ajudou na criação de toda infraestrutura da aplicação, facilitando em muitos pontos, como na geração das entidades, na criação das páginas XHTML e na integração do código de apresentação com o código da lógica dos negócios.

Uma das maiores dificuldades encontradas nesta etapa, foi personalizar o conteúdo previamente gerado pela ferramenta seam-gen, pois não estávamos familiarizados com a maneira com que ele trabalha com seus componentes e os integra com as páginas XHTML.

Outro fator com que nos preocupamos, nesta pesquisa, foi implantar um recurso capaz de recuperar as informações com relevância para o usuário e não somente selecionar os dados de um banco de dados, sem nenhum tipo de processamento, tornando-se apenas um *front-end* para os projetos mantidos.

O *framework* Apache Lucene foi utilizado no desenvolvimento deste recurso, responsável pela indexação, registro e recuperação das informações. Ele permitiu que as publicações pudessem ser exploradas e organizadas pelo sistema, fazendo com que tivéssemos o domínio sobre o conteúdo presente em seus arquivos, e, de posse destes dados, dirigir os usuários aos projetos que melhor atendessem sua necessidade de informação.

O Lucene nos forneceu uma grande abstração de um conjunto poderoso de técnicas, rotinas e algoritmos de indexação, mas, apesar de toda essa tecnologia, ele não implementa filtros para a extração de informações de nenhum tipo de arquivo, o que acabou se tornando uma de nossas tarefas em relação ao *framework*. Um tutorial de configuração com os primeiros passos para se realizar o processo de indexação de arquivos e a recuperação de suas informações, pode ser encontrado nos apêndices deste trabalho.

Assim que o tipo do sistema, as ferramentas e o *framework* para o desenvolvimento do motor de busca foram definidos, foi preciso escolher um processo de desenvolvimento de *software* que melhor atendesse o escopo deste trabalho, e se adequasse também às limitações de tempo e a uma equipe de desenvolvimento reduzida.

Optamos pelo processo de desenvolvimento ICONIX e sua utilização se tornou outro objetivo específico desta pesquisa. Os processos do ICONIX nos guiaram desde o levantamento de requisitos até a implementação e, seguindo suas diretrizes, foi possível modelar apenas os artefatos essenciais ao desenvolvimento da aplicação de uma forma direta e simplificada. Seu ciclo de vida incremental possibilitou a revisão e a modificação dos modelos gerados anteriormente a um novo cenário ou necessidade.

Neste ponto, os dois objetivos específicos da pesquisa estavam bem definidos, a modelagem da aplicação estava concluída, e as ferramentas e prazos estipulados. Dividimos as tarefas de desenvolvimento entre os autores, e mantivemos contato por meio de vídeo conferências e reuniões para acertarmos alguns detalhes.

O resultado de todas estas etapas foi o sistema de recuperação FindPro, que foi desenvolvido para ser executado em um servidor de aplicações JEE e que pode ser acessado pela internet, além de ser modelado com o processo de desenvolvimento ICONIX e escrito na linguagem de programação Java.

Alcançamos, assim, aos objetivos específicos e temos um produto capaz de armazenar as publicações acadêmicas e disponibilizá-las para futuras pesquisas. A Figura 21 ilustra a página inicial do sistema por meio do qual é possível realizar a procura pelos projetos desejados.



Fig. 21 – Página de busca do sistema de recuperação. **Fonte:** Elaborado pelos autores.

Entretanto, a aplicação finalizada ainda não se caracterizava como resultado desta pesquisa, pois seria necessário que ela cumprisse com a tarefa de dinamizar a procura, e tornar as publicações mais acessíveis, ou seja, atingir o objetivo geral deste trabalho. Para tanto, algumas tarefas tiveram de ser executadas.

Em primeiro lugar iniciamos o cadastro de alguns documentos no formato PDF para testar as funcionalidades do sistema. O principal objetivo destes testes foi procurar por possíveis erros de validação na entrada de dados. Testamos também, questões de desempenho e configurações em diferentes *hardwares* com o intuito de observar o desempenho da aplicação. A Figura 22 a seguir ilustra o cadastro de uma publicação para que, posteriormente, possa ser indexada.

findpro: Página Inicial Usuários Cursos Tipo de Projetos Projetos signed in as: cesar@findpro.com Logout

Novo Projeto

Projeto*
Sistema Colaborativo Para Requisição De Software Acadêmico

Tipo do Projeto* Trabalho de Conclusão de Curso

Autor(es)*
Bruno Pereira De Andrade; Rodrigo Luis De Faria

Orientador(es)*
Márcio Emilio Cruz Vono de Azevedo

Local de Publicação*
Pouso Alegre - MG

Ano de Publicação* 01/12/2011

Palavras-chave*
Sistema Colaborativo. Software acadêmico. ICONIX. Java.

Arquivo em PDF*

<input type="button" value="Escolher Arquivo"/>	<input type="button" value="Limpar Todos"/>
Score-TCC.pdf Done	

* campos obrigatórios

Curso *

Código	Nome
50	Sistemas de Informação

Fig. 22 – Teste de cadastro de publicações. **Fonte:** Elaborado pelos autores.

Foram cadastradas inicialmente trinta publicações, quando, três usuários utilizaram simultaneamente o sistema de indexação sem nenhum tipo de problema quanto ao acesso e a gravação concorrente nos índices. A Figura 23 ilustra a página com os projetos já cadastrados.

Resultados da procura por Projetos (14)						
Código	Tipo do Projeto	Curso	Titulo	Autor(es)	Orientador(es)	Local de Publicação
200	Trabalho de Conclusão de Curso	Sistemas de Informação	DESENVOLVIMENTO DE APLICATIVO PARA O LABORATÓRIO DO POSTO MÉDICO DO 14º GRUPO DE ARTILHARIA DE CAMPANHA	Manassés Brito do Carmo	Artur Luis Ribas Barbosa	Pouso Alegre - MG
201	Trabalho de Conclusão de Curso	Sistemas de Informação	EXPRESSÕES REGULARES	Lucio de Lima	Ednaldo David Segura	Pouso Alegre - MG
202	Trabalho de Conclusão de Curso	Sistemas de Informação	IMPLEMENTANDO E-COMMERCE COM JAVA WEB	Éder Luiz Gonçalves; Emilio Scodeler	Ednaldo David Segura	Pouso Alegre - MG
203	Trabalho de Conclusão de Curso	Sistemas de Informação	JAVA SPEECH EM AUXILIO AOS PORTADORES DE NECESSIDADES ESPECIAIS VISUAIS	Paulo Sergio de Carvalho	Artur Luis Ribas Barbosa	Pouso Alegre - MG
204	Trabalho de Conclusão de Curso	Sistemas de Informação	JPORTUGOL: UMA FERRAMENTA DE AUXÍLIO À APRENDIZAGEM DE ALGORITMOS	Andréia Cristina Dos Santos Gusmão	Artur Luis Ribas Barbosa	Pouso Alegre - MG

Fig. 23 – Publicações cadastradas. **Fonte:** Elaborado pelos autores.

Assim que todas as publicações foram cadastradas, os testes de busca começaram a ser realizados. Iniciamos estes testes utilizando termos simples como “iconix”, “java” e “software” e a aplicação se comportou como o previsto. Mas, foi na pesquisa por termos compostos e frases complexas que o sistema de recuperação se mostrou mais eficaz. Além das várias maneiras que o Lucene oferece para que a procura possa ser realizada, utilizamos também como índices propriedades particulares dos arquivos como autor, orientador e título. Isso representa um campo a mais de procura, sem sequer utilizar o motor de busca, o que torna o sistema mais rápido, já que estas propriedades se tornaram atributos dos documentos. A Figura 24 representa um busca realizada no sistema e respectivamente os resultados retornados.

The screenshot shows the Findpro search interface. At the top, there is a navigation bar with links: findpro, Página Inicial, Usuarios, Cursos, Tipo de Projetos, Projetos, and signed in as: cesar@findpro.com. Below the navigation bar, the Findpro logo is displayed. A search input field contains the query "metodologias ágeis". Below the input field is a "Buscar" button. Underneath the search form, there are two radio buttons: "Qualquer uma das palavras" (selected) and "Exatamente". A section titled "Resultados da procura por Projetos (3). Sua pesquisa levou 0 milisegundo(s)." displays a table of three search results. The table has columns: Título, Autor(es), Orientador(es), Curso, Índice de Relevância, and Visualizar. Each result includes a magnifying glass icon in the Visualizar column.

Título	Autor(es)	Orientador(es)	Curso	Índice de Relevância	Visualizar
Score-BR SISTEMA COLABORATIVO PARA REQUISIÇÃO DE SOFTWARE ACADÉMICO	Bruno Pereira De Andrade; Rodrigo Luís De Faria	Márcio Emílio Cruz Vono de Azevedo	Sistemas de Informação	0.054573905	
IMPLEMENTANDO E-COMMERCE COM JAVA WEB	Éder Luiz Gonçalves; Emílio Scodeler	Ednaldo David Segura	Sistemas de Informação	0.044559408	
SIGPOP SISTEMA GERENCIAL PARA PESQUISA DE OPINIÕES	Anderson Sofia Prado; Donizeti Costa De Moraes; Marcelo Flavio De Andrade; Renato Benedito Dos Santos	Crishna Irion	Sistemas de Informação	0.03150826	

Fig. 24 – Resultado de uma procura no sistema. **Fonte:** Elaborado pelos autores.

O sistema possibilita, além das buscas convencionais, mais oito tipos de busca: por autor, orientador, curso, local, ano, chave e tipo. Para utilizar uma destas buscas alternativas, basta que o usuário informe o tipo da busca, seguido de dois pontos mais o termo que se deseja procurar, conforme pode ser observado na Figura 25 a seguir:

The screenshot shows the Findpro application interface. At the top, there is a navigation bar with links: findpro, Página Inicial, Usuarios, Cursos, Tipo de Projetos, Projetos, signed in as: cesar@findpro.com, and Logout. Below the navigation bar, the word "Findpro" is displayed in blue. A search bar contains the query "chave:colaborativo". There are two radio buttons: one selected for "Qualquer uma das palavras" (Any of the words) and one for "Exatamente" (Exactly). A button labeled "Buscar" (Search) is to the right of the search bar. A message below the search bar says "Resultados da procura por Projetos (1). Sua pesquisa levou 0 milisegundo(s.)". A table displays the search results:

Título	Autor(es)	Orientador(es)	Curso	Índice de Relevância	Visualizar
Score-BR SISTEMA COLABORATIVO PARA REQUISIÇÃO DE SOFTWARE ACADÉMICO	Bruno Pereira De Andrade; Rodrigo Luis De Faria	Márcio Emílio Cruz Vono de Azevedo	Sistemas de Informação	1.1305887	

Fig. 25 – Resultado de uma procura utilizando o campo “palavra-chave”.

Fonte: Elaborado pelos autores.

Depois que os testes foram realizados, verificamos que a aplicação já se encontrava apta a disponibilizar as publicações aos usuários, permitindo que todo seu conteúdo pudesse ser explorado de maneira muito eficiente, e diante disso, o objetivo geral do desenvolvimento foi alcançado.

Foge do escopo desta pesquisa, o processo de implantação do sistema, embora a única dificuldade de se realizar esta tarefa seja a publicação de seu EAR. Vale ressaltar que no desenvolvimento deste projeto utilizamos o servidor de aplicações JBoss em sua versão 5.1.GA.

5 CONCLUSÃO

Os objetivos propostos neste projeto foram alcançados em sua totalidade por meio da utilização das ferramentas e tecnologias mencionadas no quadro teórico e das técnicas discutidas no quadro metodológico. O sistema de recuperação textual FindPro atendeu a todos os requisitos levantados e seu código fonte encontra-se disponível no endereço <http://code.google.com/p/findpro/>.

Deve-se destacar, também, a importância que a metodologia ICONIX teve neste trabalho. Suas diretrizes auxiliaram na divisão das quatro etapas do desenvolvimento da aplicação, como também o que deveria ser feito em cada etapa. Os diagramas UML possibilitaram que o sistema fosse documentado e estruturado, permitindo que os desenvolvedores trabalhassem, paralelamente, no desenvolvimento das funcionalidades do sistema.

As tecnologias baseadas na plataforma JavaEE também foram fundamentais para que os objetivos fossem atingidos, já que de nada adiantaria um sistema capaz de recuperar as informações dos trabalhos acadêmicos, mas que não pudesse ser acessado facilmente pela internet. Destaca-se aqui a tecnologia JBoss Seam, que pouparu bastante tempo de desenvolvimento e escrita de código de infraestrutura.

No que diz respeito à contribuição e relevância, este trabalho deixa como legado uma ferramenta capaz de difundir, de maneira interdisciplinar, o conhecimento obtido durante a produção das publicações acadêmicas, permitindo que acadêmicos de cursos distintos busquem ideias, citações e teorias nos projetos desenvolvidos na universidade.

Ainda no âmbito acadêmico, o sistema de recuperação FindPro possibilita a recuperação de informações de publicações mais antigas, bastando apenas que essas sejam cadastradas na aplicação, evitando assim que seus documentos físicos se deteriorem ou caiam no esquecimento. Estes documentos seriam então indexados e passíveis de serem recuperados com a mesma probabilidade de qualquer outro documento, tenham sido eles publicados há vinte ou há dois anos.

Aos acadêmicos do curso de Sistemas de Informação em especial, além de apresentar a utilização de tecnologias JavaEE, apresenta-se também o *framework* Apache Lucene, ou seja, estes alunos podem explorar o código fonte e os tutoriais deste trabalho a título de aprendizagem ou com a intenção de dar continuidade e agregar funcionalidades à aplicação.

Nas questões profissionais, o desenvolvimento do presente trabalho mostrou que utilizando as metodologias e as ferramentas corretas, é possível desenvolver aplicações para os mais variados problemas, ou minimizar e aperfeiçoar trabalhos repetitivos. Aliás, uma grande quantidade de funcionalidades ainda pode ser incorporada ao sistema como um farejador de plágios ou um sistema de gerenciamento de bibliotecas, uma vez que tais funcionalidades solidificariam ainda mais o sistema como um produto para o mercado de *softwares*.

Quanto ao desenvolvimento, constatou-se que maior atenção poderia ser dada às *stopwords* e aos sinônimos, com o objetivo de melhorar ainda mais a relevância das busca. Outra funcionalidade que poderia ser melhorada está relacionada ao cadastro dos projetos, no qual uma alternativa seria automatizar a entrada de dados no sistema por meio da leitura e preenchimento dos elementos pré-textuais no formulário, ficando isso como lição para trabalhos futuros.

Espera-se que tanto este documento quanto o sistema desenvolvido sejam de grande serventia aqueles que os utilizarem como ferramenta de procura, ou mesmo como referencial prático para futuros projetos, visto que, o desenvolvimento de *softwares* e a recuperação de informações podem ser muito mais explorados de forma conjunta. Este trabalho demonstrou que o acesso às publicações da universidade pode ser aperfeiçoado com uma solução simples e objetiva, fornecendo aos usuários uma maneira alternativa de explorar o conhecimento gerado na universidade com muito mais praticidade de disponibilidade.

REFERÊNCIAS

- AMORIM, S.R.L.; CHERIAF, Malik; **Sistema de indexação e recuperação de Informação em construção baseado em ontologia.** Disponível em <<http://www6.ufrgs.br/norie/tic2007/artigos/A1115.pdf>> Acesso em: 04 mar. 2012.
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 12.676.** Referências bibliográficas. Rio de Janeiro: ABNT, 1992.
- BARROS, A. J. S.; LEHFELD, N. A. S. **Fundamentos de Metodologia: Um Guia para a Iniciação Científica.** 2. ed. São Paulo: Makron Books, 2000 p.87.
- BELKIN, N. J.; CROFT, W. B. **Information Filtering and Information Retrieval: Two sides of the same coin?** Disponível em: <http://www.ischool.utexas.edu/~i385d/readings/Belkin_Information_92.pdf>. Acesso em: 06 mai. 2012.
- BONA, C. **Avaliação de Processos de Software:** um estudo de caso em XP e ICONIX. Disponível em <<ftp://atenas.cpd.ufv.br/dpi/mestrado/Gerais/TeseIconix.pdf>> Acesso em: 01 mai. 2012.
- FISCHER, A. E.; GRODZINSKY, F. S. **The Anatomy of Programming Languages.** Prentice Hall, 1993.
- GEARY, D.; HORSTMANN, C. **Cora JavaServer Faces.** 3. ed. Boston: Prentice Hall, 2010.
- GOSLING, J.; JOY, B.; STEELE, G.; BRACHA, G.; BUCKLEY, A. **The Java™ Language Specification.** California: Oracle, 2011.
- HATCHER, E; GOSPODNETIC, O; MCCANDLESS, M. **Lucene In Action.** 2. ed. Greenwich: Manning, 2009.
- HORSTMANN, C. S.; CORNELL, G. **Core Java: Volume 1 – Fundamentos.** 8. ed. São Paulo: Pearson, 2008.
- ICONIXPROCESS. **Iconix Process.** Disponível em: <<http://iconixprocess.com/iconix-process/>>. Acesso em: 30 de abril de 2012.
- LANCASTER, F. W. **Indexação e Resumos:** Teoria e prática. 2. ed. Brasília: Briquet Lemos, 2004.
- MARCONI, M.A.; LAKATOS, E.M. **Técnicas de pesquisa.** 5. ed. São Paulo: Atlas, 2002.
- OLIVEIRA, M. M. **Como Fazer:** projetos, relatórios, monografias, dissertações e testes. 3. ed. Rio de Janeiro: Campus, 2005.
- ORACLE. **The Java EE 6 Tutorial.** Disponível em: <<http://docs.oracle.com/javaee/6/tutorial/doc/bnaaw.html>> Acesso em: 28 abr. 2012.
- PANDA, D.; RAHMAN, R.; LANE, D. **EJB 3 In Action.** Greenwich: Manning, 2007.
- PRESSMAN, R. S. **Engenharia de Software.** 5. ed. São Paulo: Makron Books, 2002.

- _____. **Engenharia de Software**. 6. ed. São Paulo: McGraw-Hill, 2006.
- RIGGS, S.; KROSING, H. **PostgreSQL 9 Administration Cookbook**. Birmingham: Packt Publishing, 2010.
- ROSENBERG, D.; STEPHENS, M.; COLLINS-COPE M. **Agile Development with ICONIX Process**: People, Process, and Pragmatism. New York: Apress, 2005.
- _____. **Use Case Driven Object Modeling with UML: Theory and Practice**. New York: Apress, 2007.
- SALOMON, D. V. **Como fazer uma monografia**. 9. ed. rev.- São Paulo: Martins Fontes, 1999.
- SAYÃO, L.F.; BARROS, A.C.T.M.; **Centro de informações nucleares: 25 anos de apoio da CNEN à área de C&T**. disponível em: <<http://capim.ibict.br/index.php/ciinf/article/view/558/507>> Acesso em 07, mar. 2012.
- SEAM REFERENCE. **Seam 2 Documentation**. Disponível em: <http://www.seamframework.org/Seam2/Documentation> Acesso em 10 ago. 2012.
- SILVA, A. M. R.; VIDEIRA, C. A. E. **UML, Metodologias e Ferramentas Case**. Lisboa: Centro Atlântico; 2001.
- SONAWANE, A. **Usando o Apache Lucene para Procura de Texto**. Disponível em: <<http://www.ibm.com/developerworks/br/java/library/os-apache-lucenesearch>> Acesso em 29 fev. 2012.
- THE APACHE SOFTWARE FOUNDATION. **Apache Lucene**. Disponível em <<http://lucene.apache.org/core/>> Acesso em 28 de abril 2012.
- VILAÇA, M. L. C. **Pesquisa e ensino**: considerações e reflexões. e-scrita: Revista do Curso de Letras da UNIABEU, Nilópolis: UNIABEU, v. 1, n. 2, p. 59-74, ago. 2010.

APÊNDICE I
ARTEFATOS DO PROJETO

CASOS DE USO

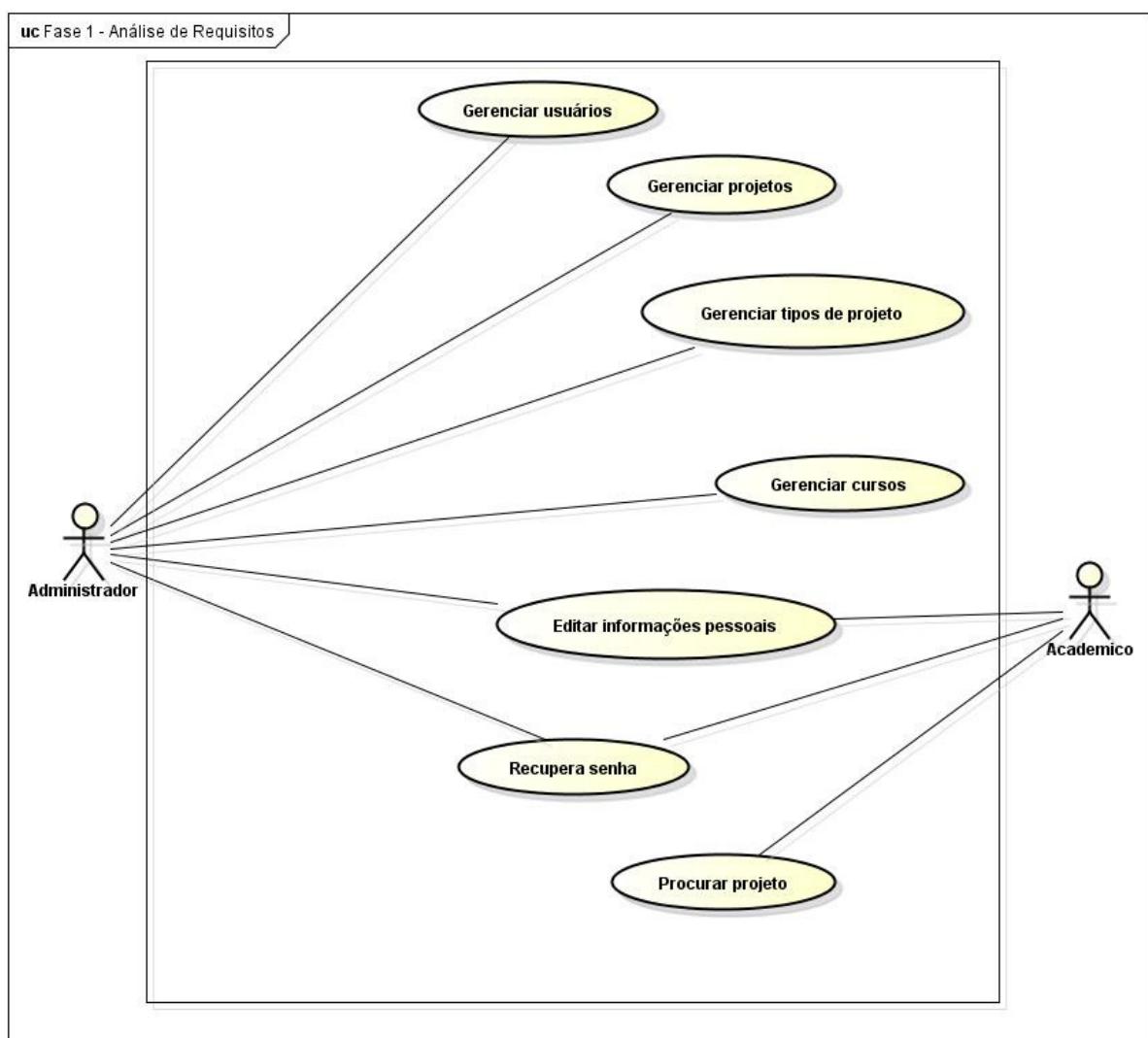


Fig. 1 – Diagrama de casos de uso. **Fonte:** Elaborado pelos autores.

1 ARTEFATOS DO CASO DE USO – GERENCIAR USUÁRIOS

1.1 CADASTRAR USUÁRIOS

Caso de uso relacionado: Gerenciar usuário	
Descrição:	Cadastrar um novo usuário no sistema.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Usuários no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar usuários.</p> <p>3 – Ator clica em “Cadastrar usuário”.</p> <p>5 – Sistema exibe formulário para cadastrar usuário.</p> <p>6 – Ator informa Nome, e-mail, permissão e clica em gravar.</p> <p>7 – Sistema valida os dados, gera uma senha e a envia por e-mail para o novo usuário.</p> <p>8 – Sistema grava os dados.</p>
Fluxo Alternativo:	No item 6 do fluxo principal caso o usuário deixe de preencher algum campo: 6.1 – Sistema exibe mensagem de erro.
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 2 – Fluxo de Eventos – Cadastrar Usuário. Fonte: Elaborado pelos autores.

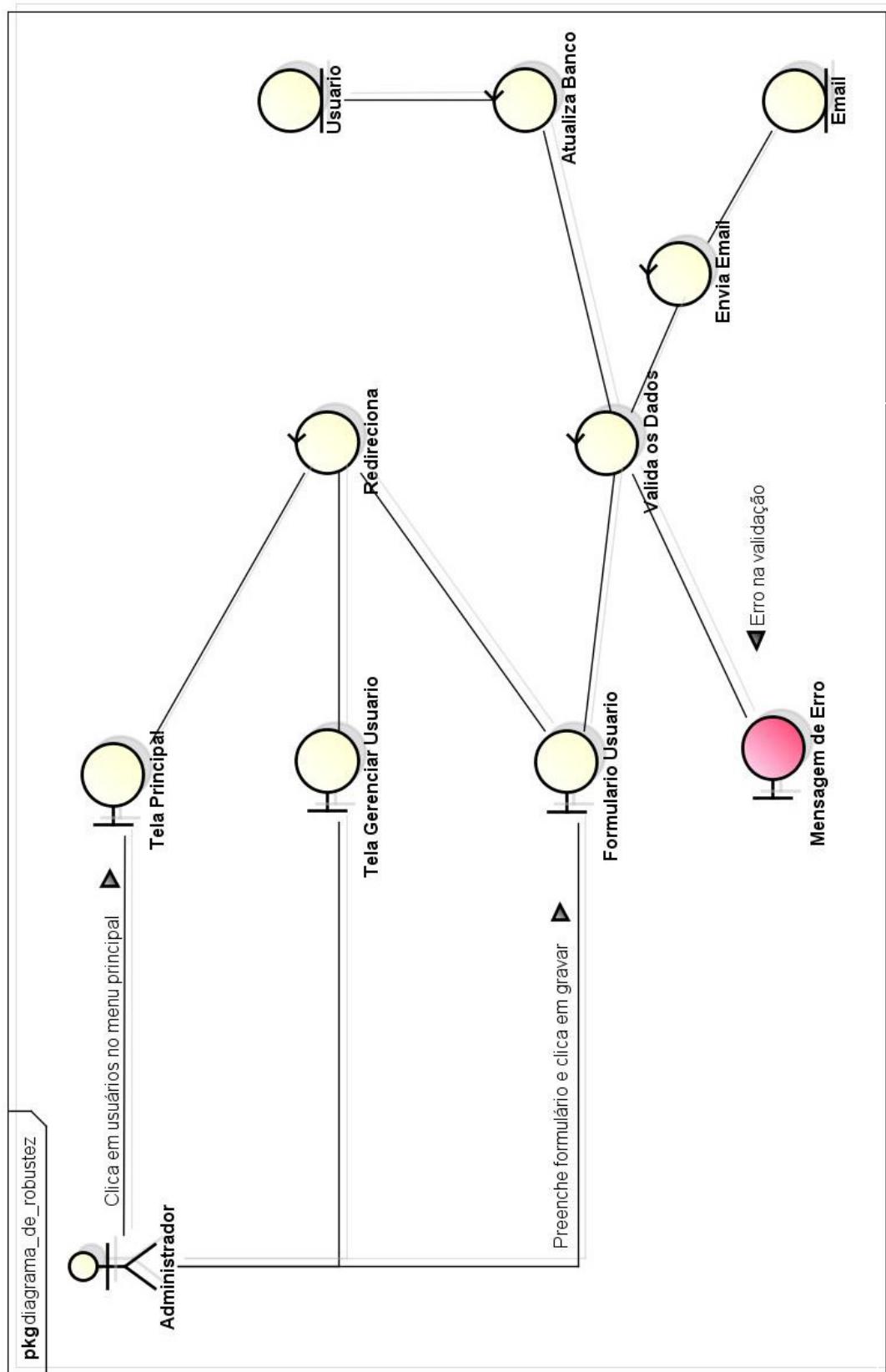


Fig. 3– Diagrama de Robustez – Cadastrar Usuário. **Fonte:** Elaborado pelos autores.

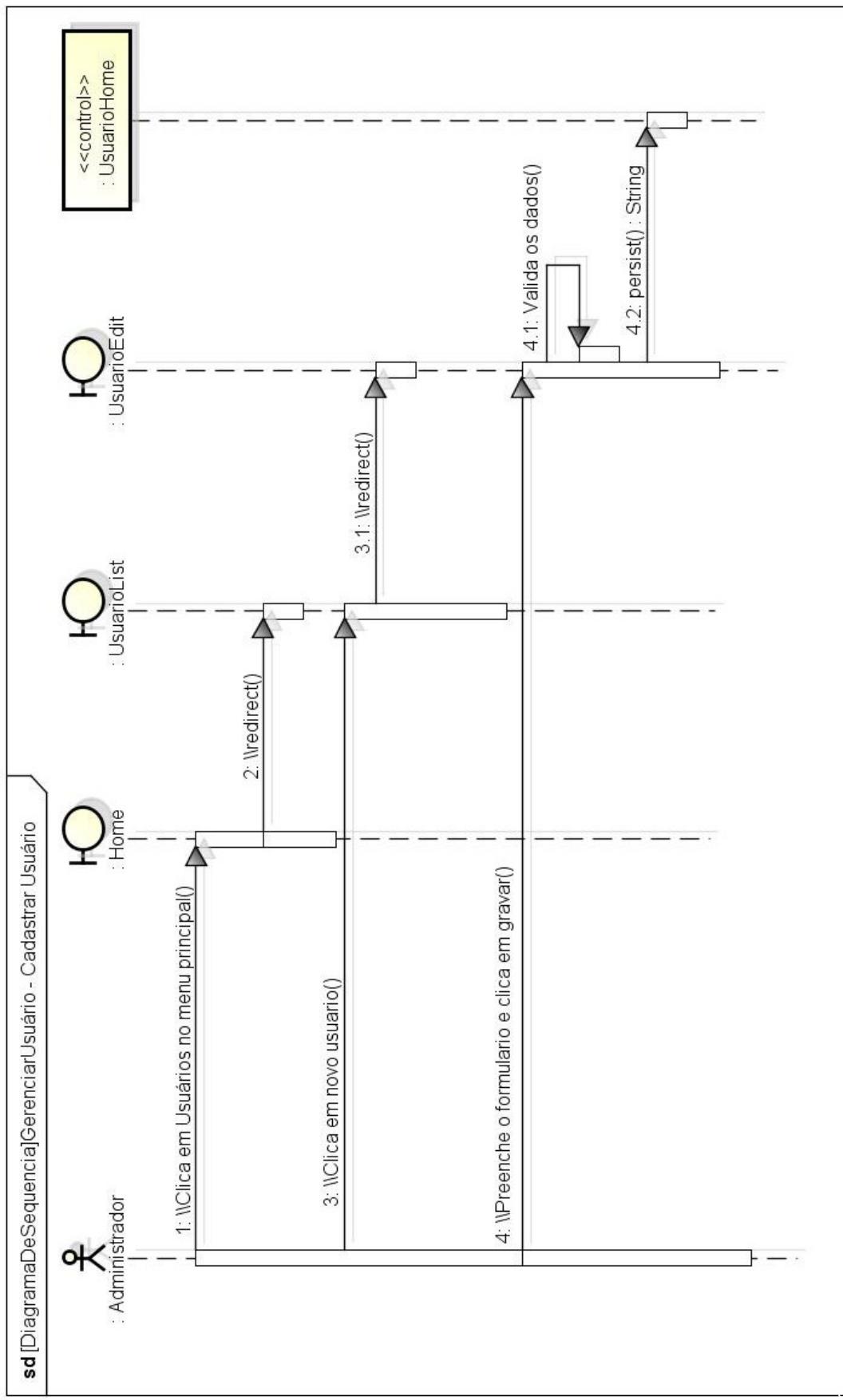


Fig. 4 – Diagrama de Seqüência – Cadastrar Usuário. **Fonte:** Elaborado pelos autores.

1.2 EDITAR USUÁRIOS

Caso de uso relacionado: Gerenciar usuário	
Descrição:	Editar um usuário.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Usuários no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar usuários.</p> <p>3 – Ator informa nome ou id ou e-mail e clica em buscar.</p> <p>4 – Sistema exibe lista com os resultados.</p> <p>5 – Ator seleciona usuário e clica em editar.</p> <p>6 – Sistema exibe formulário para editar usuário.</p> <p>7 – Ator altera as informações do usuário e clica em gravar.</p> <p>8 – Sistema valida os dados e grava o usuário no banco</p>
Fluxo Alternativo:	No item 8 do fluxo principal caso ocorra um erro de validação: 8.1 – Sistema exibe mensagem de erro.
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 5 – Fluxo de Eventos – Editar Usuário. **Fonte:** Elaborado pelos autores.

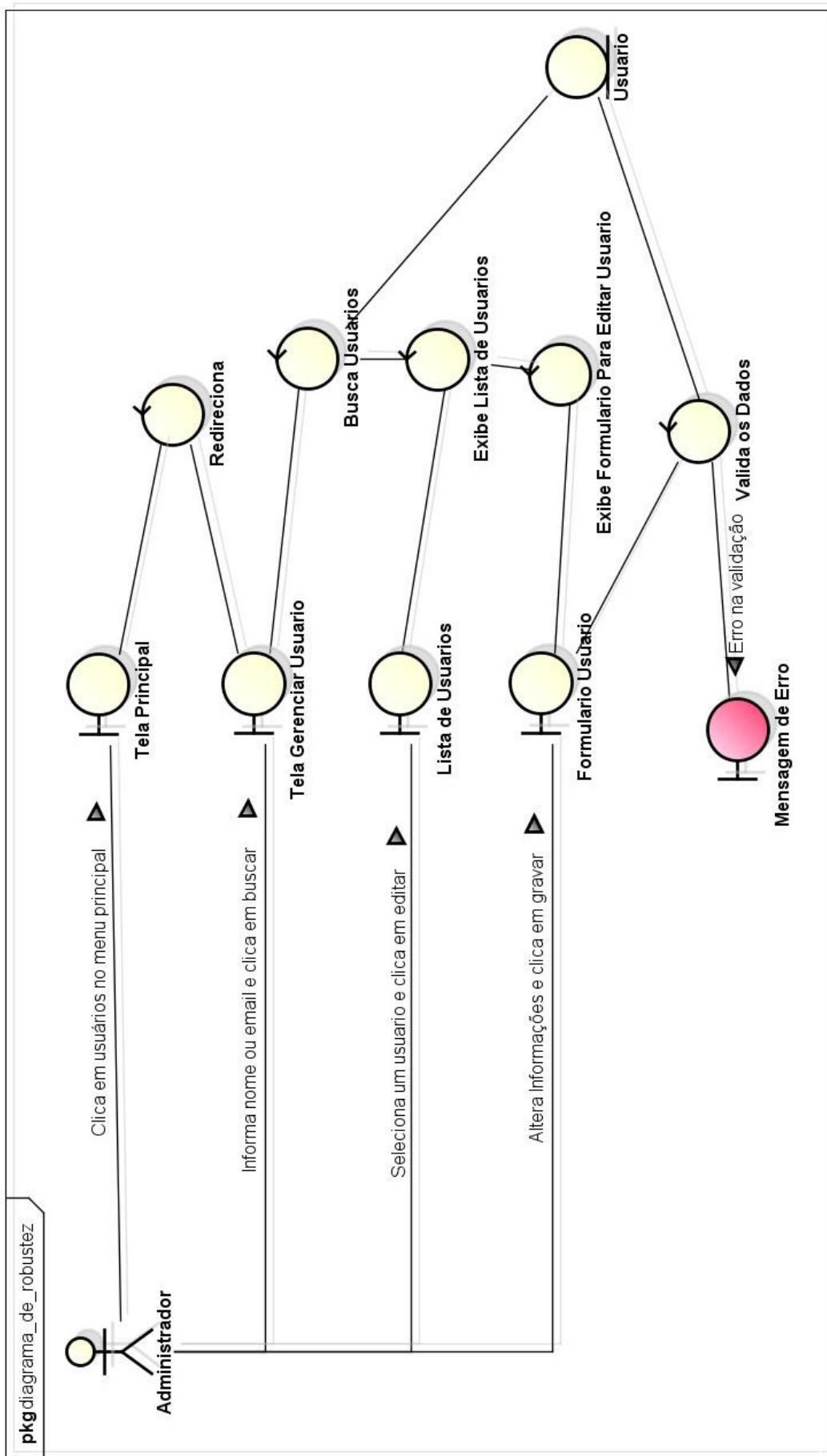


Fig. 6 – Diagrama de Robustez – Editar Usuário. **Fonte:** Elaborado pelos autores.

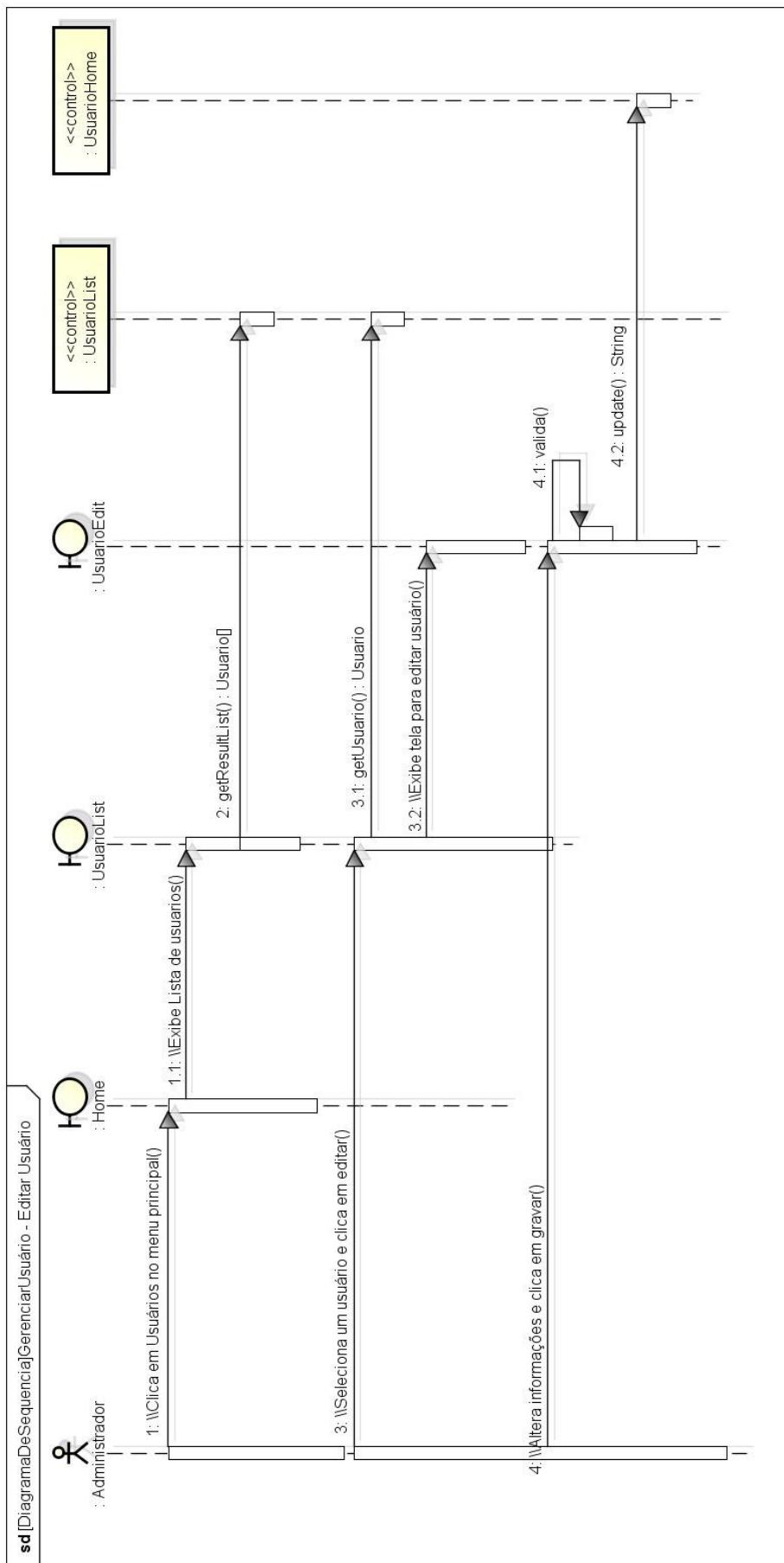


Fig. 7 – Diagrama de Sequência – Editar Usuário. **Fonte:** Elaborado pelos autores.

1.3 REMOVER USUÁRIOS

Caso de uso relacionado: Gerenciar usuário	
Descrição:	Remover um usuário do sistema.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Usuários no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar usuários.</p> <p>3 – Ator informa nome ou id ou e-mail e clica em buscar.</p> <p>4 – Sistema exibe lista com os resultados.</p> <p>5 – Ator seleciona um usuário e clica em excluir.</p> <p>6 – Sistema exibe mensagem de confirmação.</p> <p>7 – Ator clica em confirmar.</p> <p>8 – Sistema exclui o usuário.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 8 –Fluxo de Eventos – Remover Usuário. **Fonte:** Elaborado pelos autores.

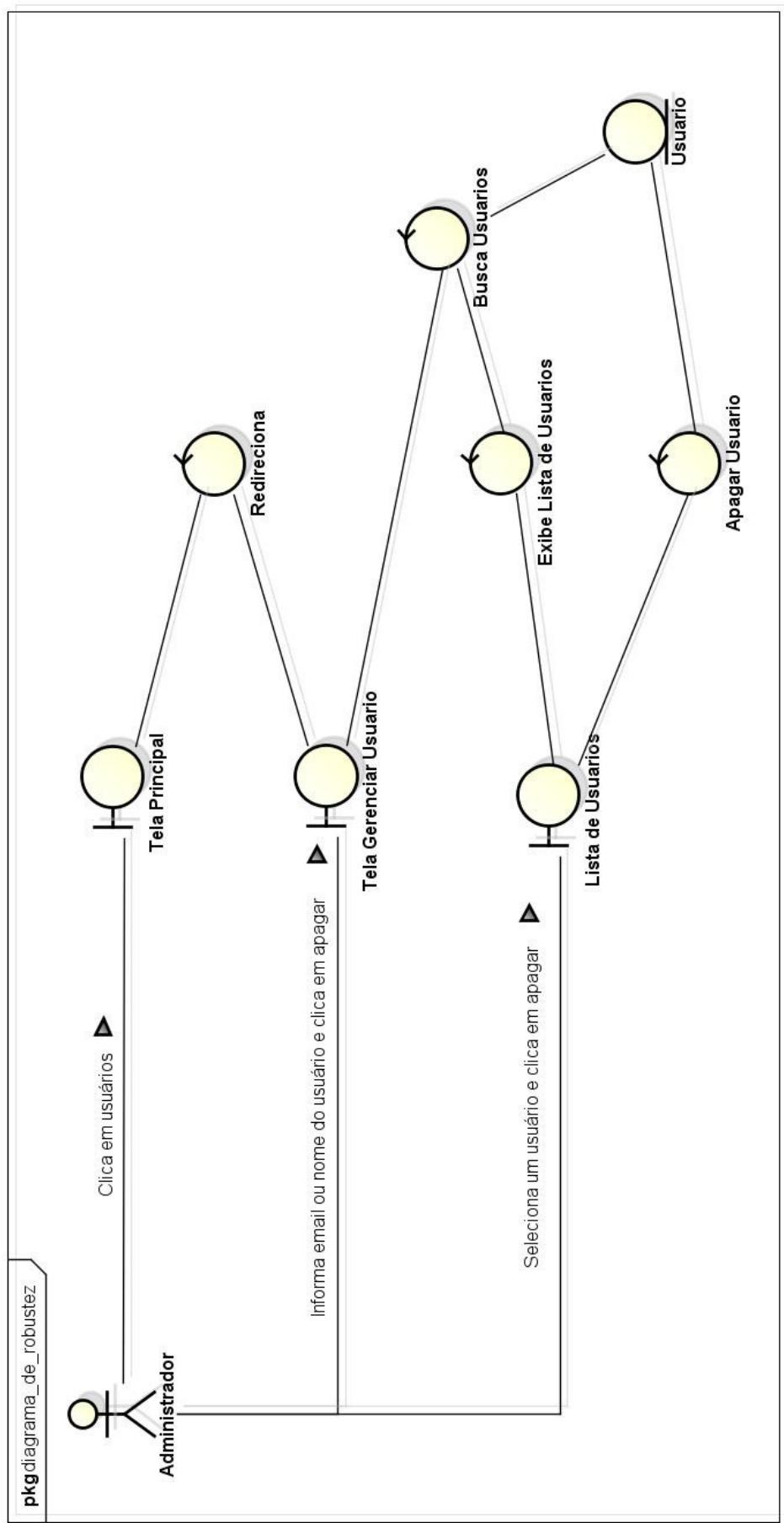


Fig. 9 –Diagrama de Robustez – Remover Usuário. **Fonte:** Elaborado pelos autores.

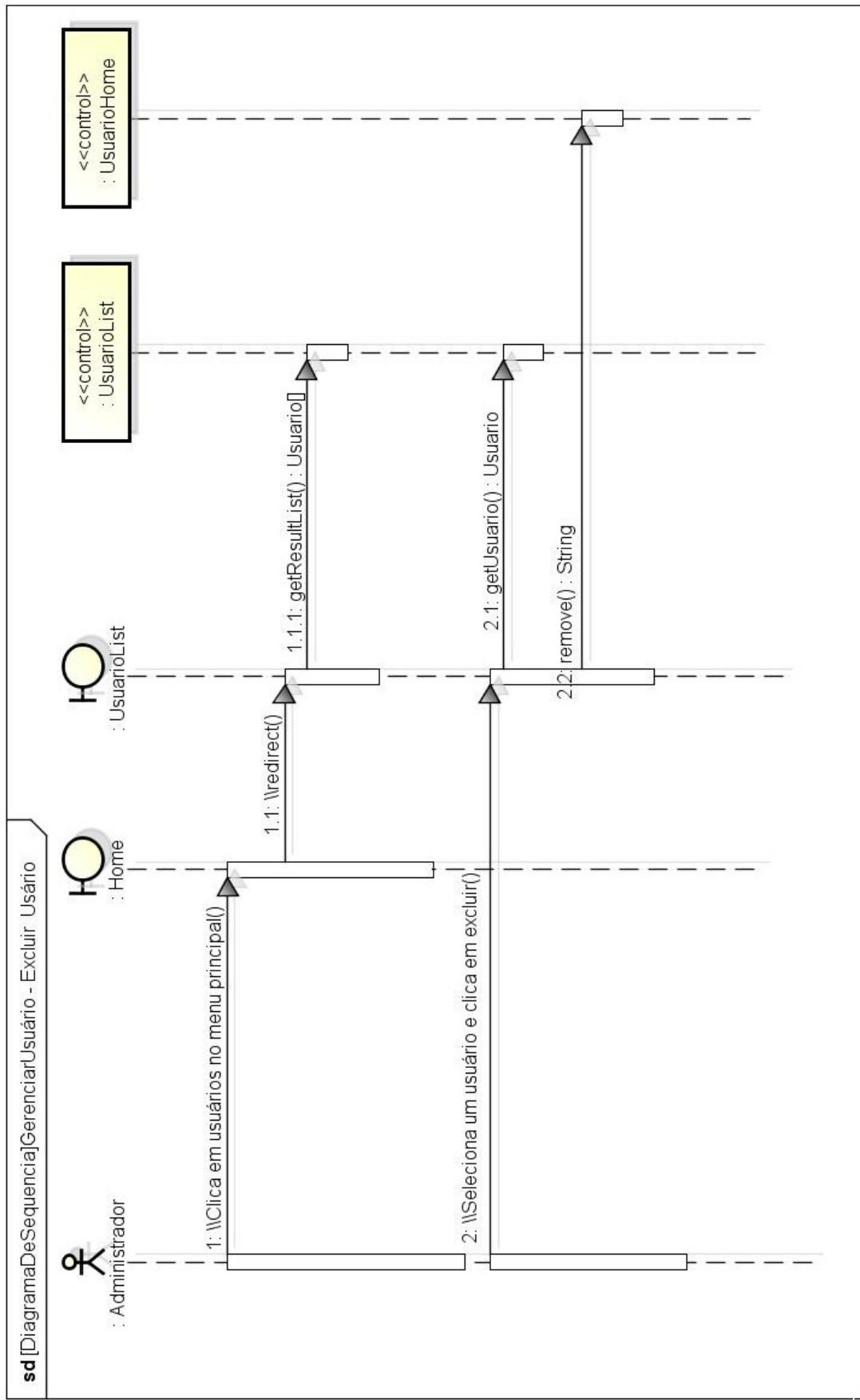


Fig. 10 –Diagrama de Seqüência – Remover Usuário. **Fonte:** Elaborado pelos autores.

2 ARTEFATOS DO CASO DE USO – GERENCIAR PROJETOS

2.1 CADASTRAR PROJETOS

Caso de uso relacionado: Gerenciar projetos	
Descrição:	Cadastrar um novo projeto no sistema.
Ator(es):	Administrador.
Pré-condições:	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Projetos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar projeto.</p> <p>4 – Ator clica em “Cadastrar projeto”.</p> <p>5 – Sistema exibe janela para cadastrar projeto.</p> <p>6 – Ator informa título, autores, orientadores, local, data, palavras chave, tipo, curso, adiciona o arquivo para upload e clica em gravar.</p> <p>7 – O sistema faz a validação das informações, persiste os dados e indexa o arquivo.</p> <p>8 – O sistema exibe uma mensagem de sucesso.</p>
Fluxo Alternativo:	<p>No item 6 do fluxo principal caso o ator clique no botão “Enviar Arquivo”:</p> <p>6.1 – O sistema exibe uma janela para o usuário fazer o upload do arquivo.</p> <p>No Item 7 do fluxo principal caso a validação encontre algum erro nos dados:</p> <p>7.1 – O sistema exibe mensagem de erro informando os campos que foram preenchidos incorretamente.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 11 – Fluxo de Eventos – Cadastrar Projeto. **Fonte:** Elaborado pelos autores.

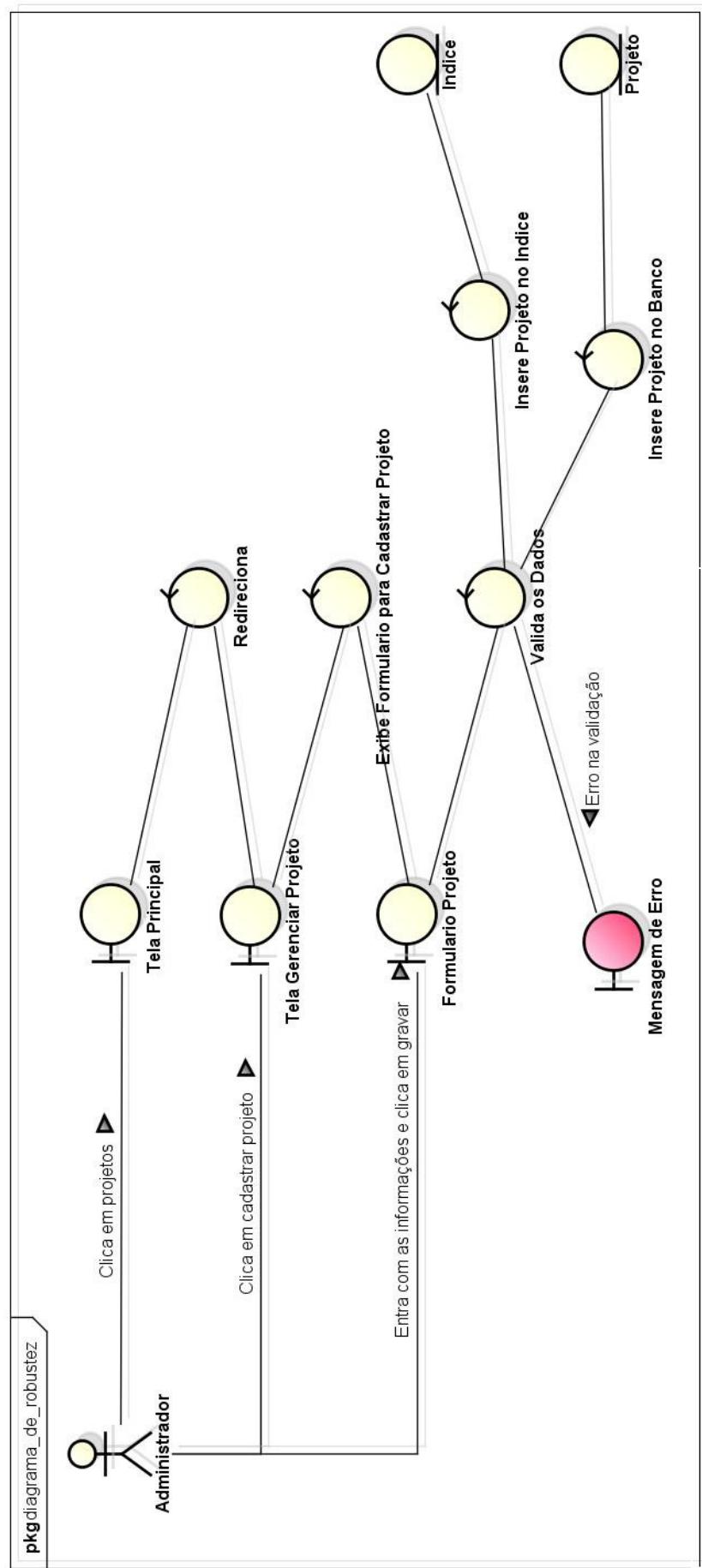


Fig. 12 – Diagrama de Robustez – Cadastrar Projeto. Fonte: Elaborado pelos autores.

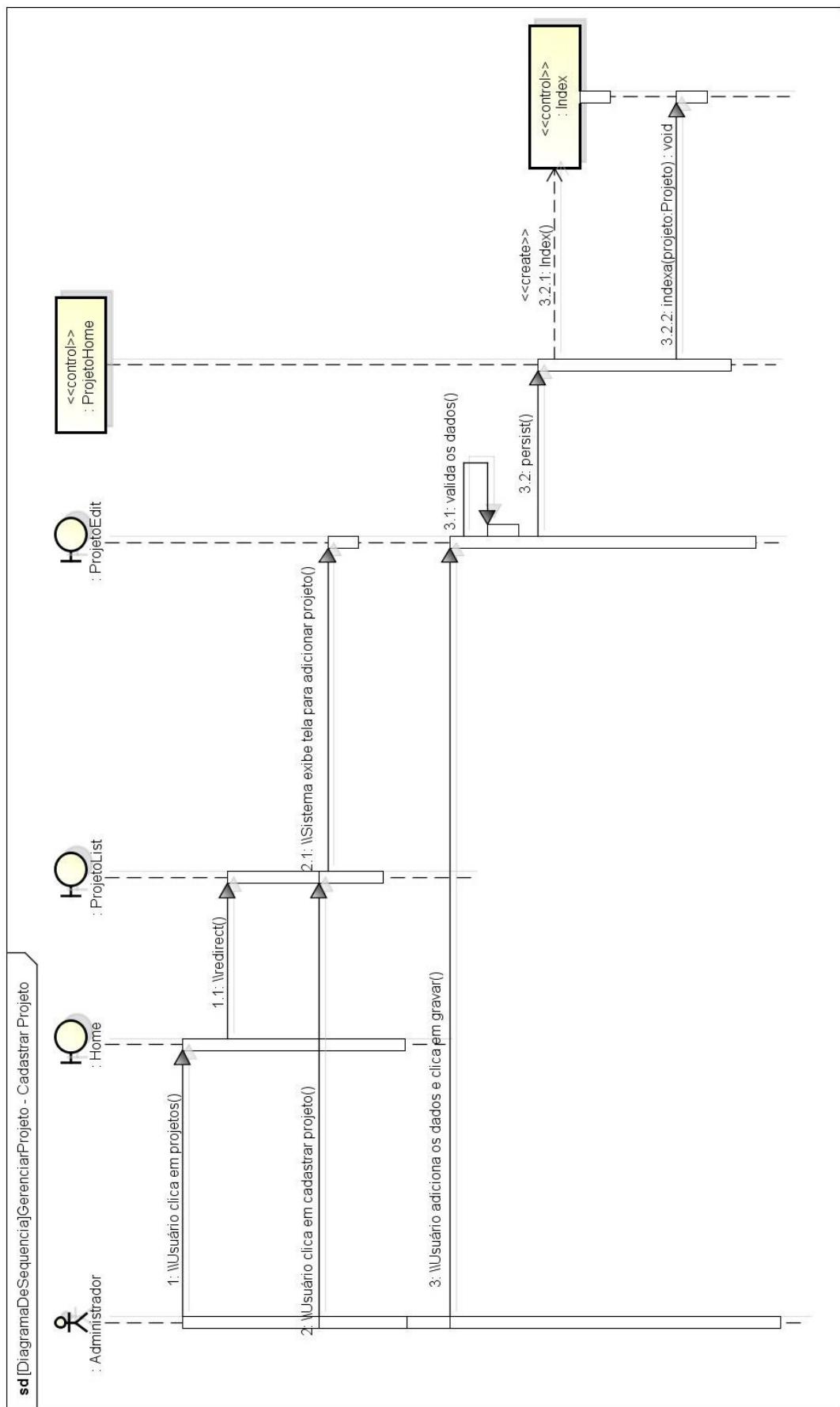


Fig. 13 – Diagrama de Sequência – Cadastrar Projeto. **Fonte:** Elaborado pelos autores.

2.2 EDITAR PROJETOS

Caso de uso relacionado: Gerenciar projetos	
Descrição:	Editar um projeto no sistema.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Gerenciar projetos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar projetos com campos para buscar por projetos.</p> <p>3 – Ator informa título ou id ou autor e clica em buscar.</p> <p>4 – Sistema exibe lista com os resultados.</p> <p>5 – Ator seleciona um projeto e clica em “editar”</p> <p>6 – Sistema exibe formulário com os dados do projeto para o ator editar.</p> <p>7 – Ator altera informações e clica em salvar.</p> <p>6 – Sistema valida os dados, persiste o projeto e atualiza os índices.</p>
Fluxo Alternativo:	No Item 6 do fluxo principal caso o ator preencha algum campo incorretamente: 6.1 – Sistema exibe mensagem de erro.
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 14 – Fluxo de Eventos – Editar Projeto. **Fonte:** Elaborado pelos autores.

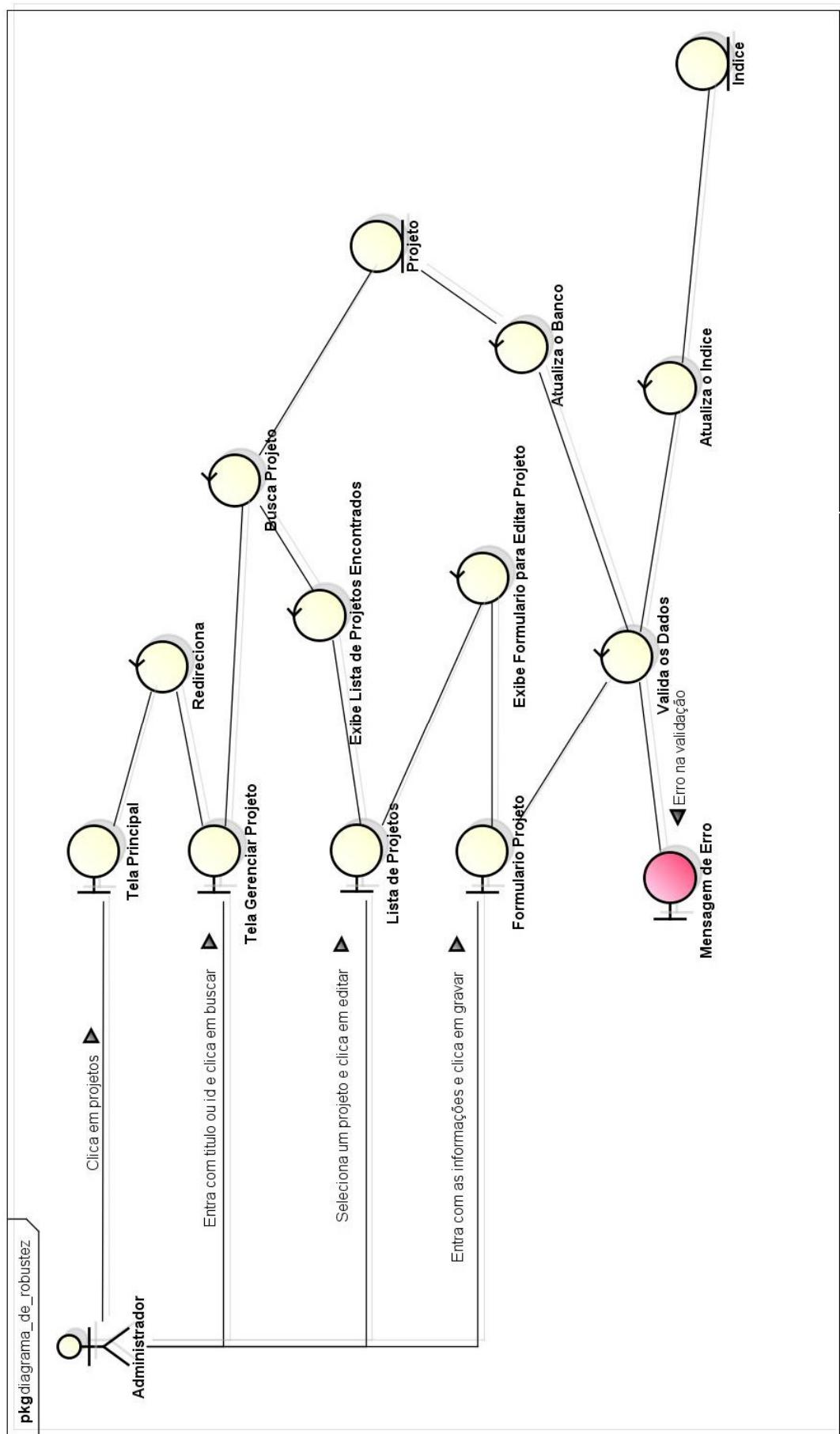


Fig. 15 – Diagrama de Robustez – Editar Projeto. **Fonte:** Elaborado pelos autores.

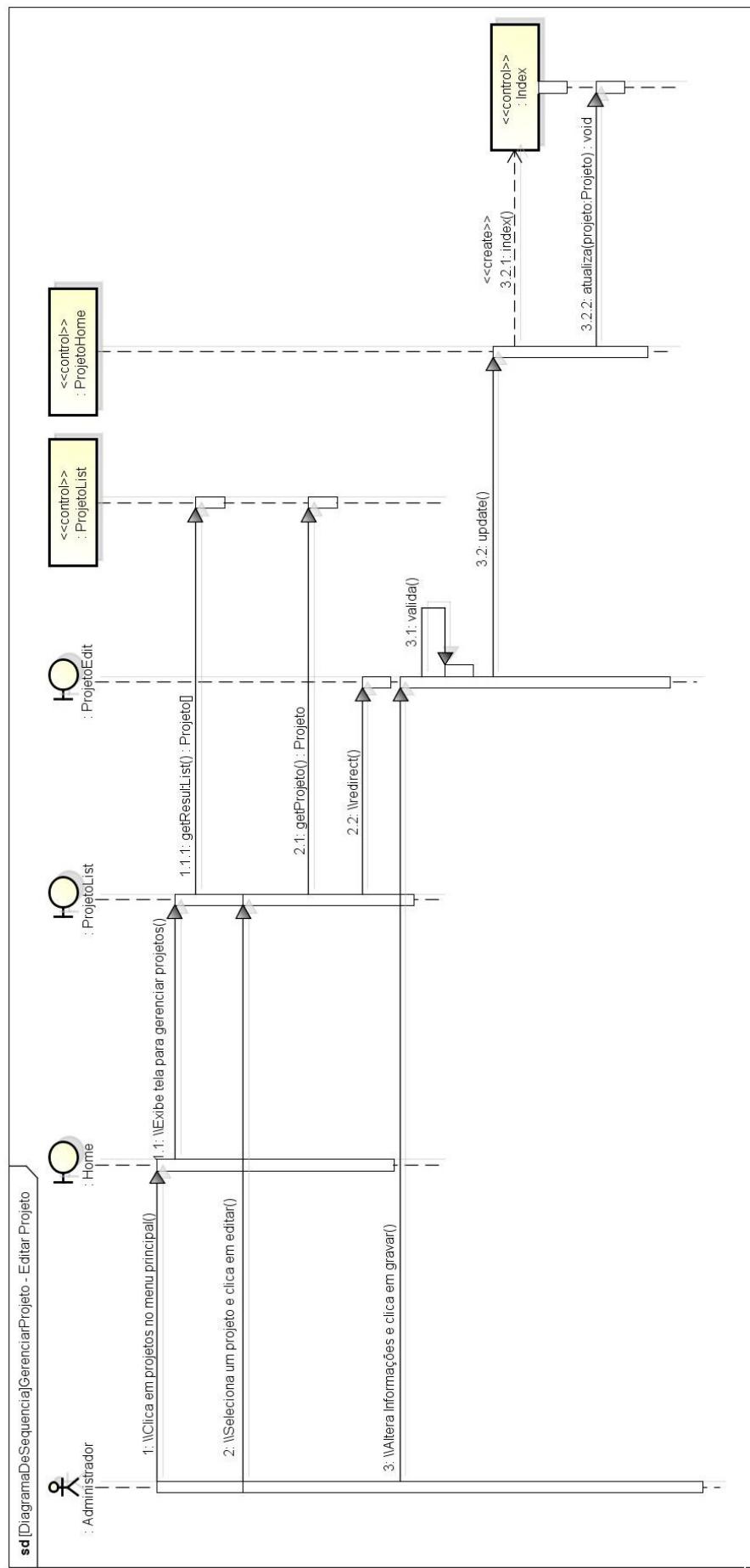


Fig. 16 – Diagrama de Sequência – Editar Projeto. **Fonte:** Elaborado pelos autores.

2.3 REMOVER PROJETOS

Caso de uso relacionado: Gerenciar projetos	
Descrição:	Remover um projeto do sistema.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Gerenciar Projetos no menu principal.</p> <p>2 – Sistema exibe tela gerenciar projetos com campo para buscar por projetos.</p> <p>3 – Ator informa titulo ou id ou autor do projeto e clica em buscar.</p> <p>4 – Sistema exibe lista com os resultados.</p> <p>5 – Ator seleciona um projeto e clica em “excluir”.</p> <p>6 – Sistema exibe mensagem de confirmação.</p> <p>7 – Ator clica em confirmar.</p> <p>8 – Sistema exclui o projeto e atualiza índice.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 17 – Fluxo de Eventos – Remover Projeto. **Fonte:** Elaborado pelos autores.

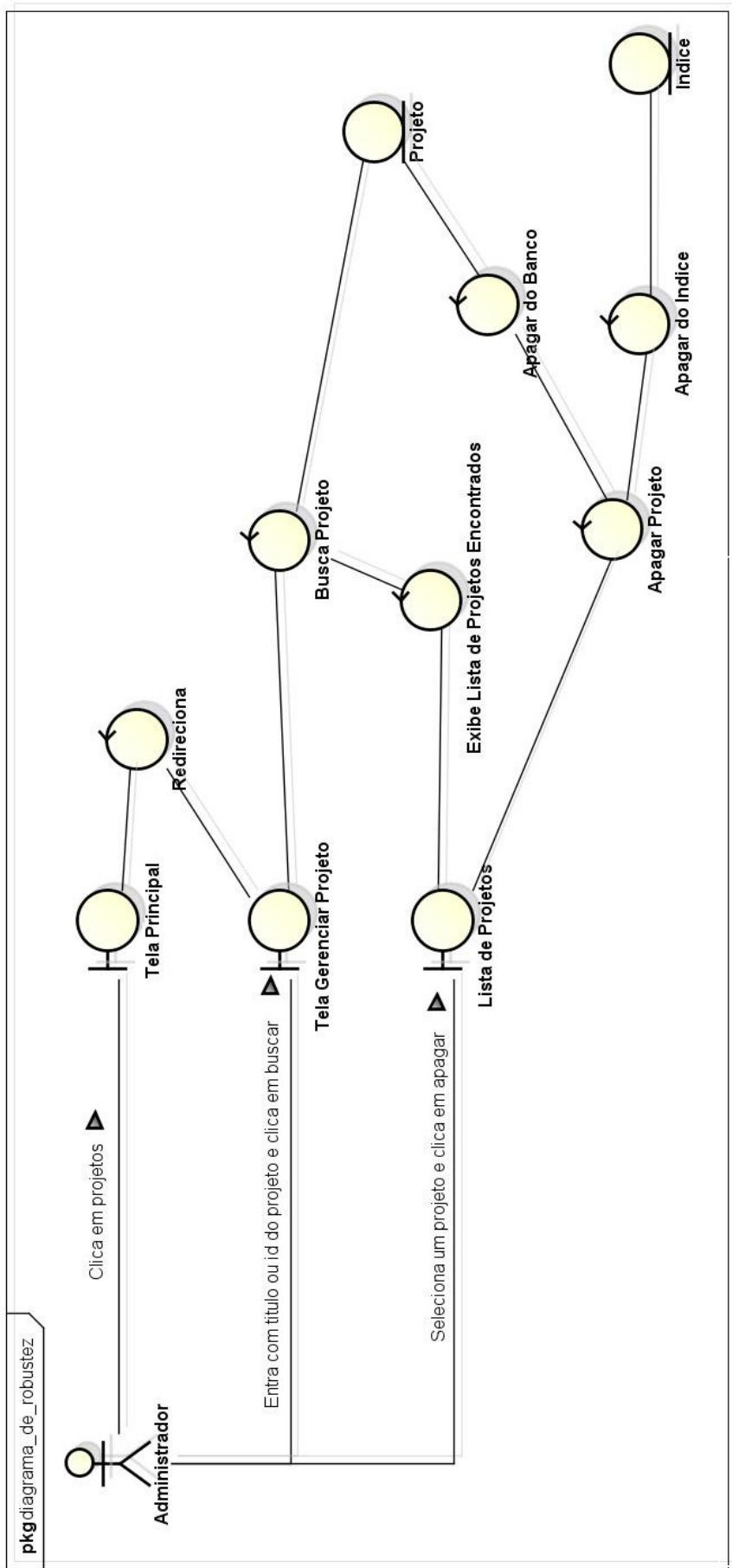


Fig. 18 – Diagrama de Robustez – Remover Projeto. **Fonte:** Elaborado pelos autores.

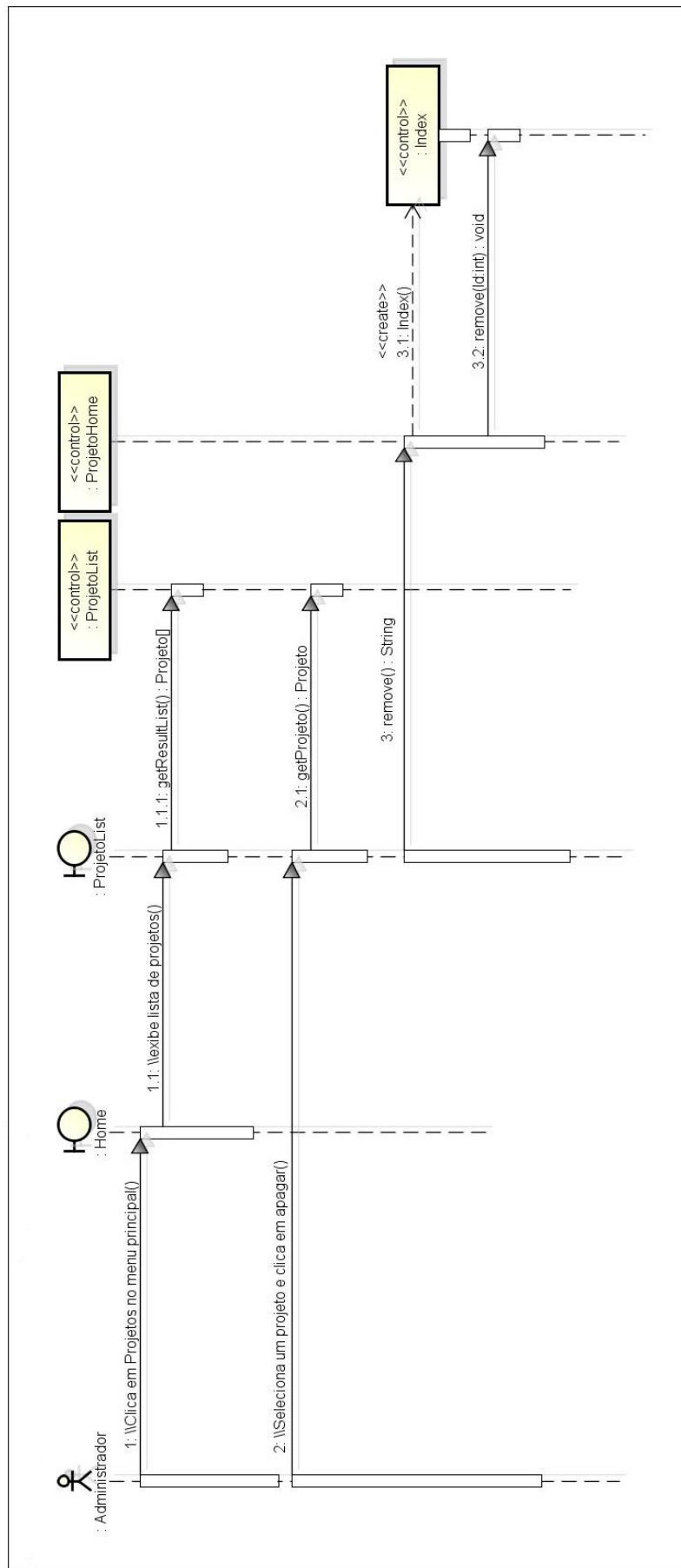


Fig. 19 – Diagrama de Sequência – Remover Projeto. **Fonte:** Elaborado pelos autores.

3 ARTEFATOS DO CASO DE USO – GERENCIAR TIPOS DE PROJETOS

3.1 CADASTRAR TIPO DE PROJETO

Caso de uso relacionado: Gerenciar Tipos de Projeto	
Descrição:	Cadastrar um novo tipo de projeto no sistema.
Ator(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Tipos de projeto no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar tipos de projeto.</p> <p>3 – Ator clica em “Cadastrar Tipo de Projeto”</p> <p>3 – Sistema exibe formulário para cadastrar tipo de projeto.</p> <p>5 – Ator preenche o formulário.</p> <p>6 – Sistema valida os dados, grava o novo tipo no banco e exibe mensagem de sucesso.</p>
Fluxo Alternativo:	No Item 6 do fluxo principal caso ocorra algum erro de validação: <p>6.1 – Sistema exibe mensagem de erro.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 20 – Fluxo de Eventos – Cadastrar Tipo Projeto. **Fonte:** Elaborado pelos autores.

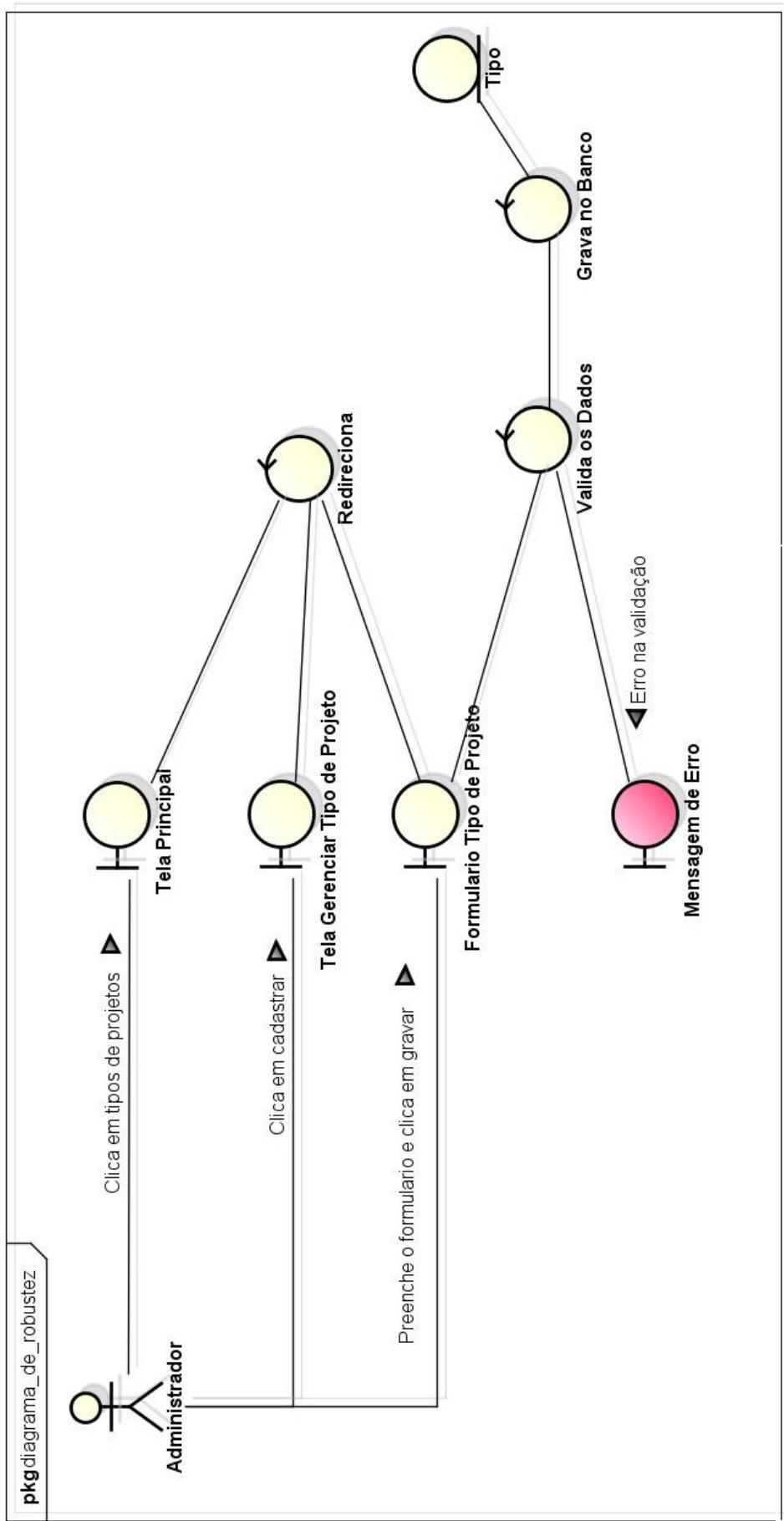


Fig. 21 – Diagrama de Robustez – Cadastrar Tipos Projeto. **Fonte:** Elaborado pelos autores.

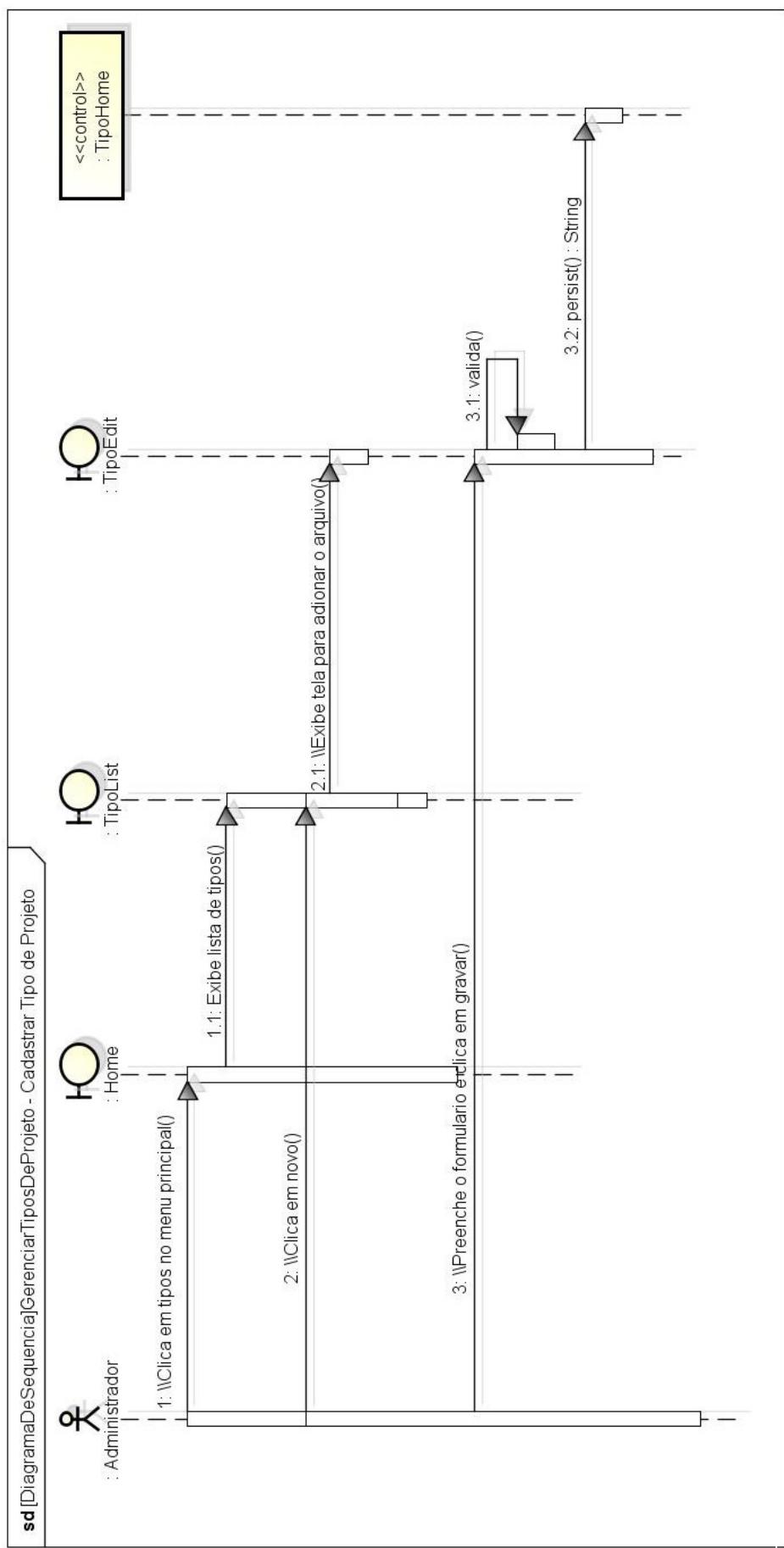


Fig. 22 – Diagrama de Sequência – Cadastrar Tipos Projeto. **Fonte:** Elaborado pelos autores.

3.2 EDITAR TIPO DE PROJETO

Caso de uso relacionado: Gerenciar Tipos de Projeto	
Descrição:	Editar tipos de projeto.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Tipos de Projeto no menu principal.</p> <p>2 – Sistema exibe janela para gerenciar tipos de projetos com campo para o ator buscar por um tipo.</p> <p>3 – Ator entra com o nome do tipo e clica em buscar.</p> <p>4 – Sistema exibe lista com os tipos de projeto encontrados.</p> <p>5 – Ator seleciona um tipo de projeto e clica em editar.</p> <p>6 – Sistema exibe formulário para editar tipo de projeto.</p> <p>7 – Ator entra com os dados e clica em gravar.</p> <p>8 – Sistema valida os dados, persiste e exibe mensagem de sucesso.</p>
Fluxo Alternativo:	<p>No item 8 caso ocorra algum erro de validação:</p> <p>8.1 - Sistema exibe mensagem de erro informando o campo que foi preenchido incorretamente.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 23 – Fluxo de Eventos – Editar Tipo de Projeto. **Fonte:** Elaborado pelos autores.

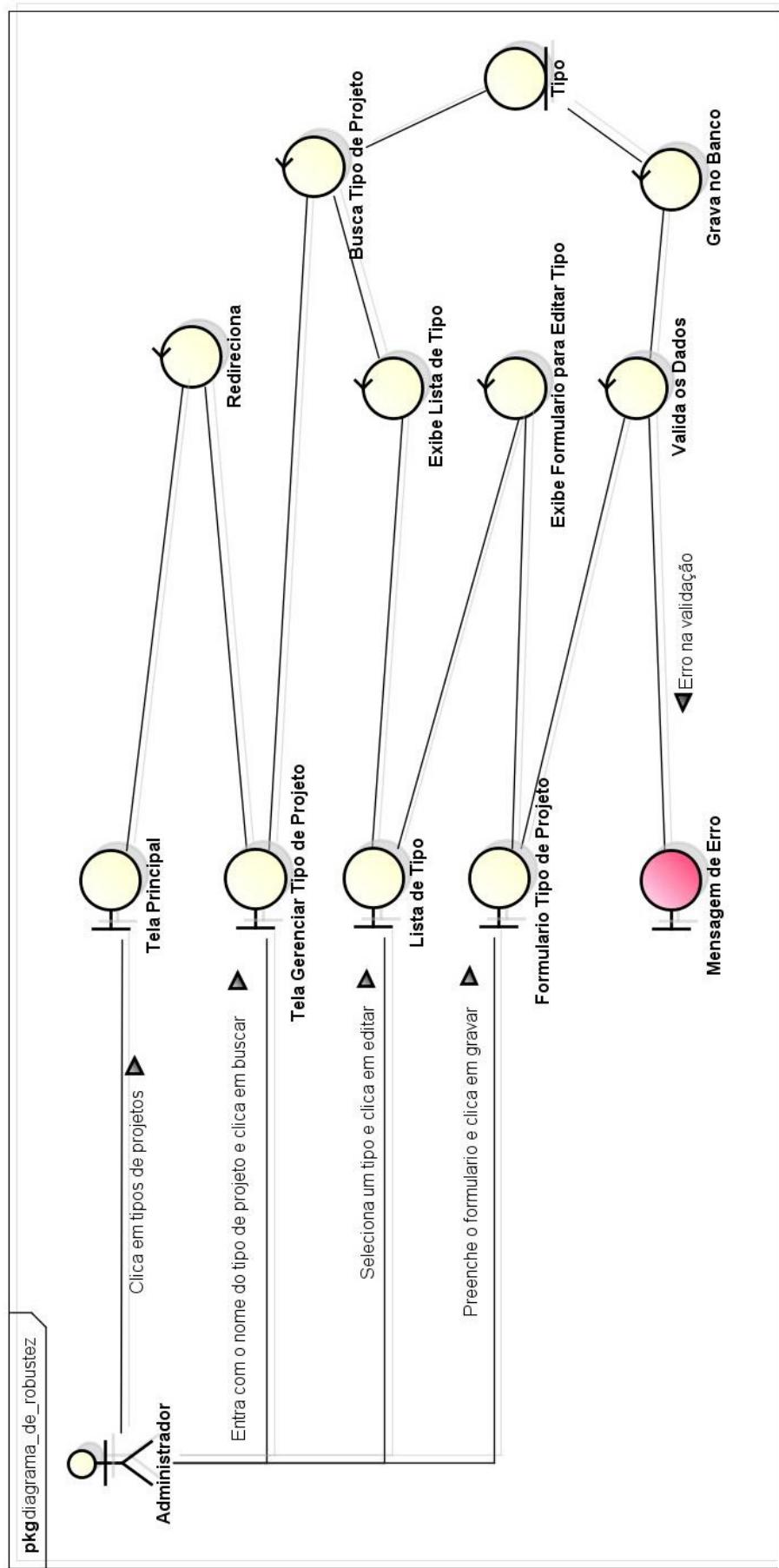


Fig. 24 – Diagrama de Robustez – Editar Tipo de Projeto. Fonte: Elaborado pelos autores.

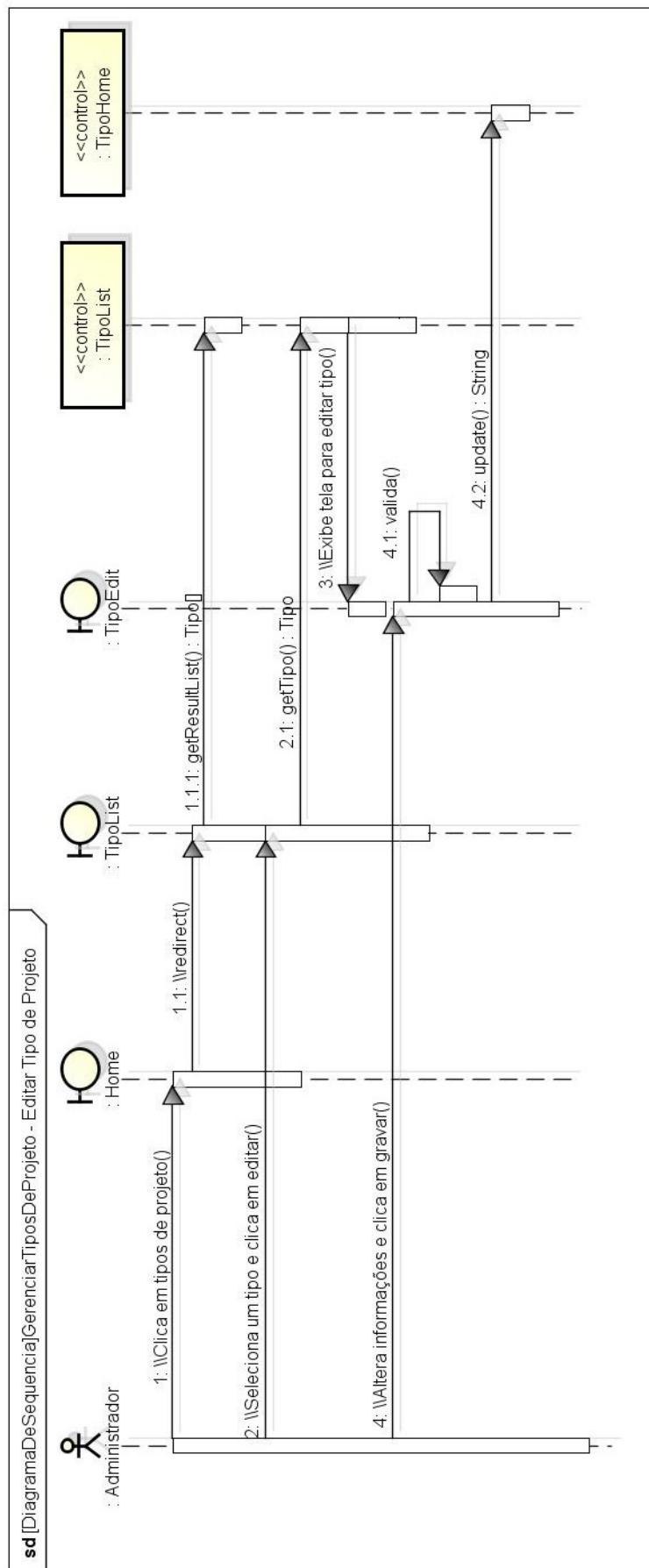


Fig. 25 – Diagrama de Seqüência – Editar Tipo de Projeto. **Fonte:** Elaborado pelos autores.

3.3 REMOVER TIPO DE PROJETO

Caso de uso relacionado: Gerenciar tipos de projeto	
Descrição:	Remover Tipo de projeto.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Tipos de Projeto no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar tipos de projetos com campo para o ator buscar por um tipo.</p> <p>3 – Ator entra com o nome do tipo e clica em buscar.</p> <p>4 – Sistema exibe lista com os tipos de projeto encontrados.</p> <p>5 – Ator seleciona um tipo de projeto e clica em excluir.</p> <p>6 – Sistema exibe mensagem de confirmação.</p> <p>7 – Ator clica em confirmar.</p> <p>8 – Sistema exclui tipo de projeto.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 26 – Fluxo de Eventos – Remover Tipo de Projeto. **Fonte:** Elaborado pelos autores.

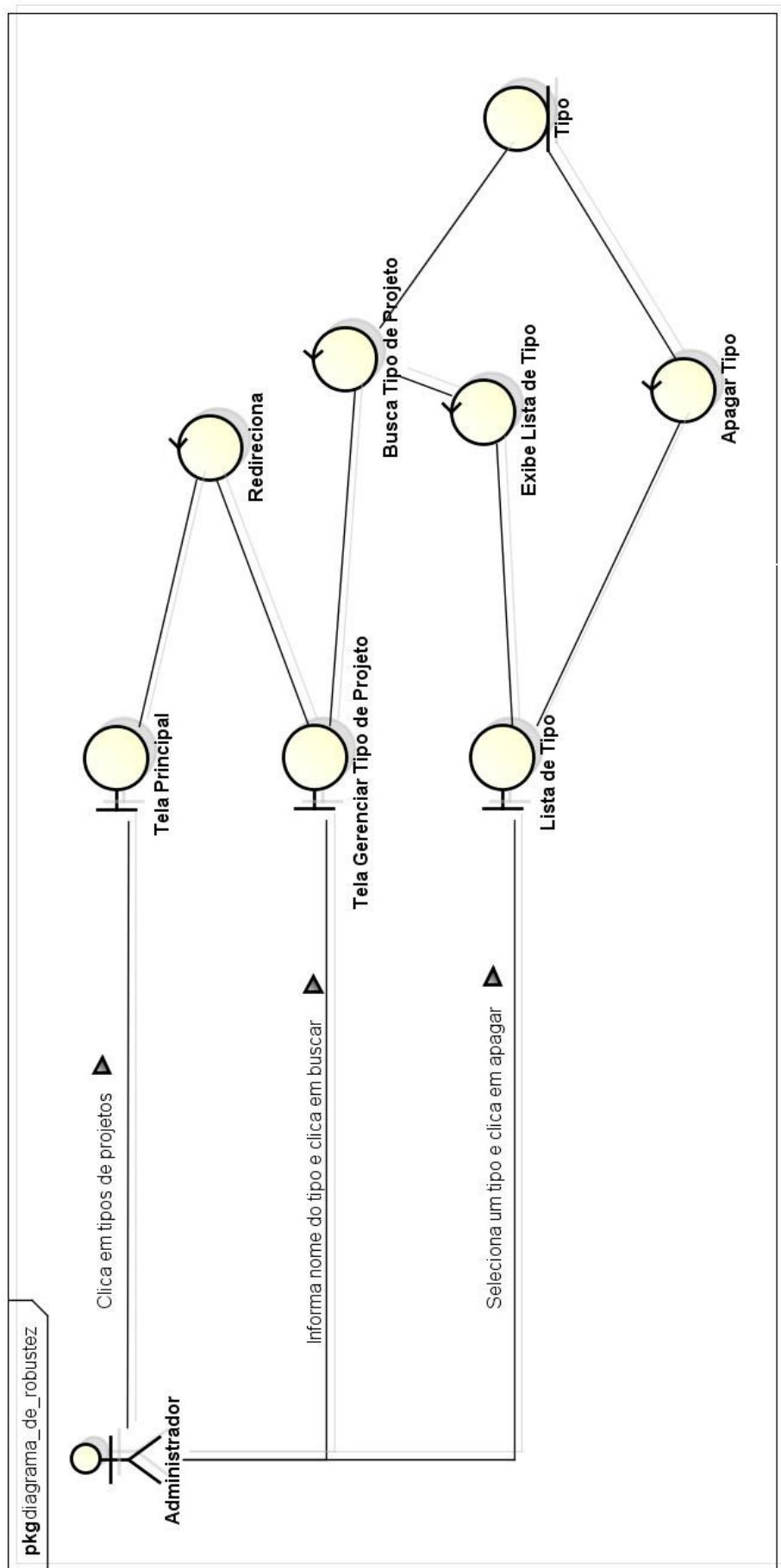


Fig. 27 – Diagrama de Robustez – Remover Tipo de Projeto. **Fonte:** Elaborado pelos autores.

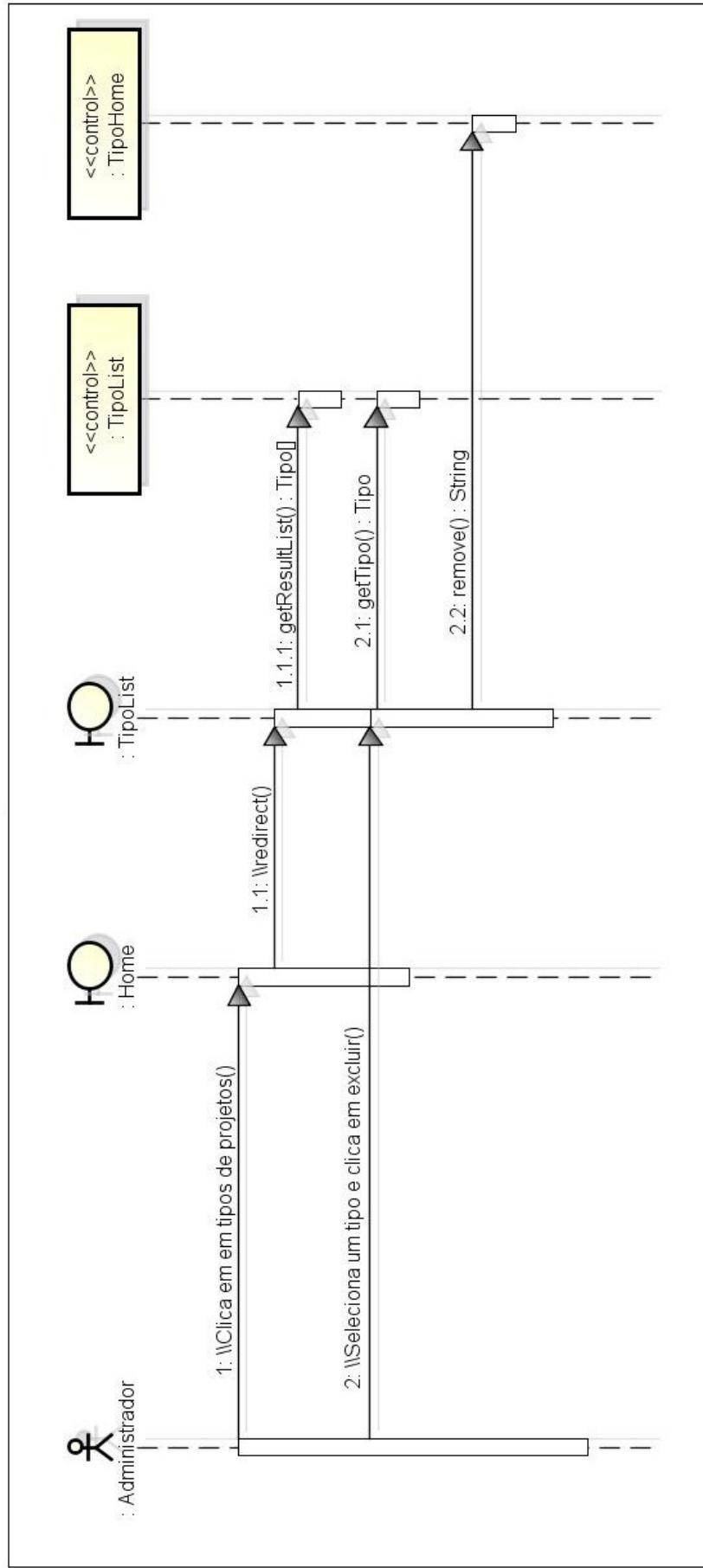


Fig. 28 – Diagrama de Sequência – Remover Tipo de Projeto. **Fonte:** Elaborado pelos autores.

4 ARTEFATOS DO CASO DE USO – GERENCIAR CURSOS

4.1 CADASTRAR CURSO

Caso de uso relacionado: Gerenciar Curso	
Descrição:	Cadastrar Curso.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Cursos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar cursos.</p> <p>3 – Ator clica em Cadastrar curso.</p> <p>4 – Sistema exibe tela para cadastrar curso.</p> <p>5 – Ator informa o nome do curso e clica em gravar.</p> <p>6 – Sistema valida os dados, persiste o curso e exibe mensagem de sucesso.</p>
Fluxo Alternativo:	No item 6 do fluxo principal caso ocorra algum erro de validação: <p>6.1 – Sistema exibe mensagem erro informando o campo que foi preenchido incorretamente.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 29 – Fluxo de Eventos – Cadastrar Curso. **Fonte:** Elaborado pelos autores.

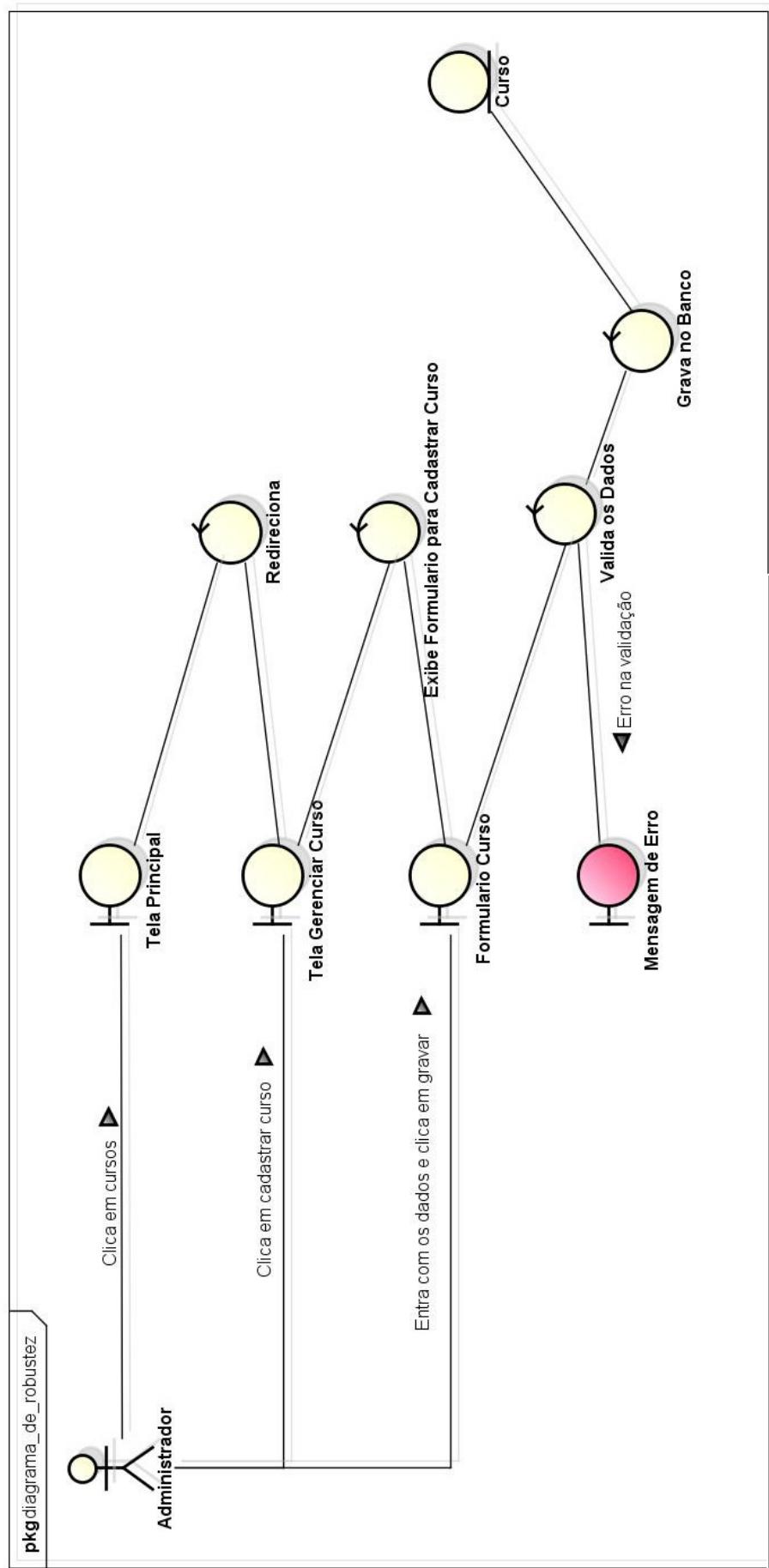


Fig. 30 – Diagrama de Robustez – Cadastrar Curso. Fonte: Elaborado pelos autores.

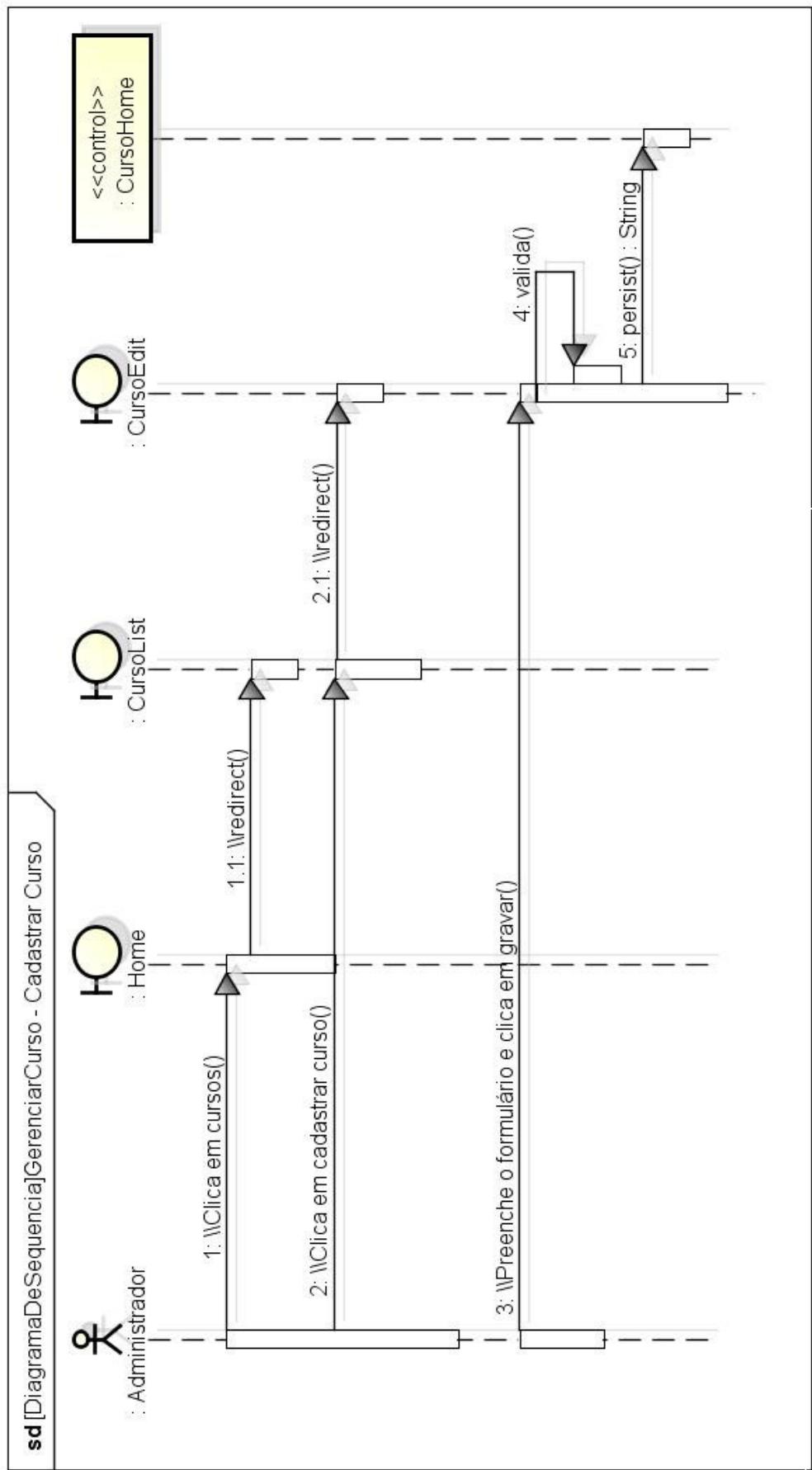


Fig. 31 – Diagrama de Sequência – Cadastrar Curso. Fonte: Elaborado pelos autores.

4.2 EDITAR CURSO

Caso de uso relacionado: Gerenciar Curso	
Descrição:	Editar curso.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Cursos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar cursos com campo para buscar cursos por nome.</p> <p>3 – Ator entra com o nome do curso e clica em Buscar.</p> <p>4 – Sistema exibe lista com os cursos encontrados.</p> <p>5 – Ator seleciona um curso e clica em editar.</p> <p>6 – Sistema exibe tela para editar curso.</p> <p>7 – Ator altera informações e clica em gravar.</p> <p>8 – Sistema valida os dados, persiste o curso e exibe mensagem de sucesso.</p>
Fluxo Alternativo:	No item 8 do fluxo principal caso ocorra algum erro de validação: 8.1 – Sistema exibe mensagem de erro.
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 32 – Fluxo de Eventos – Editar Curso. **Fonte:** Elaborado pelos autores.

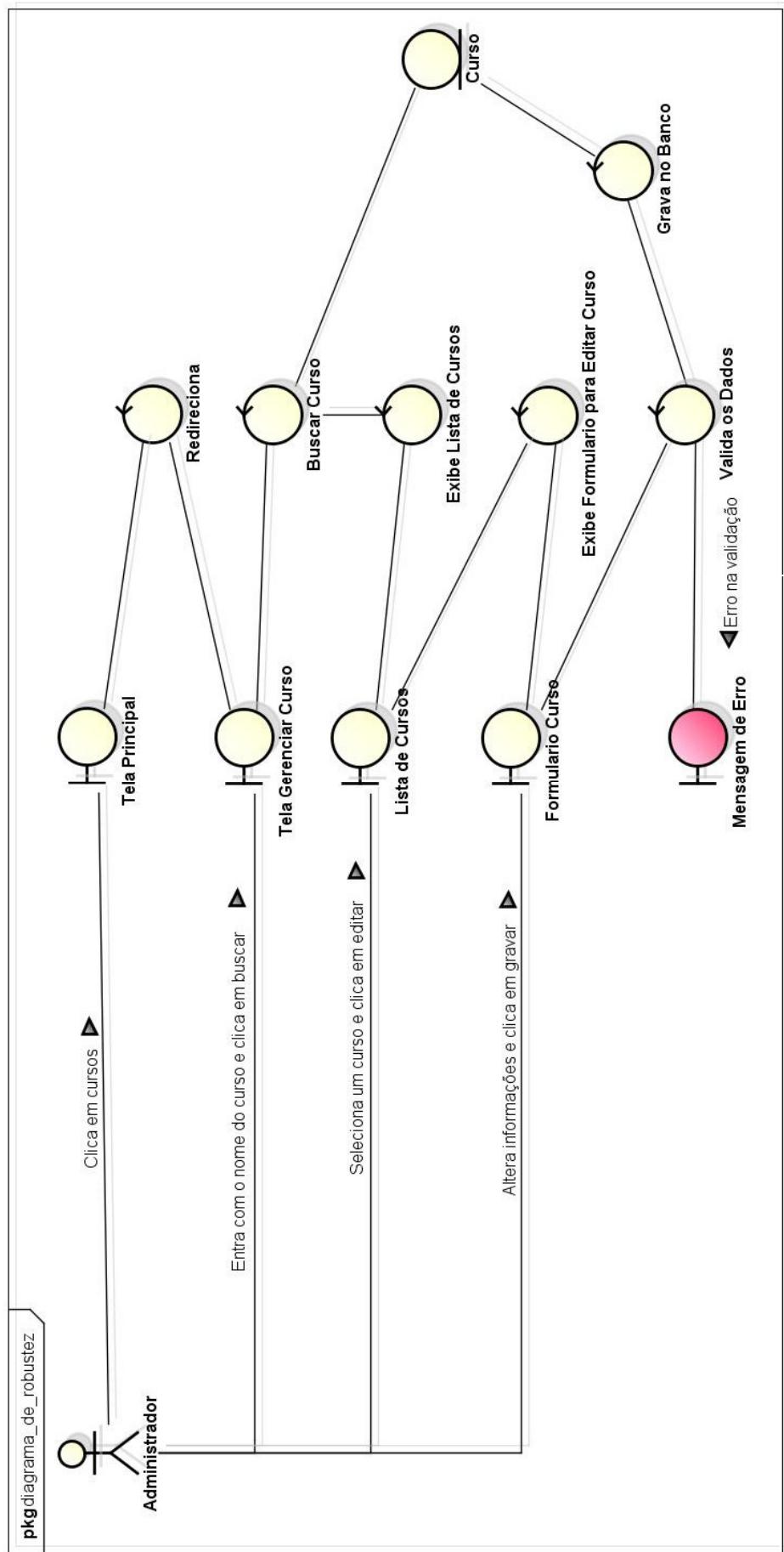


Fig. 33 – Diagrama de Robustez – Editar Curso. Fonte: Elaborado pelos autores.

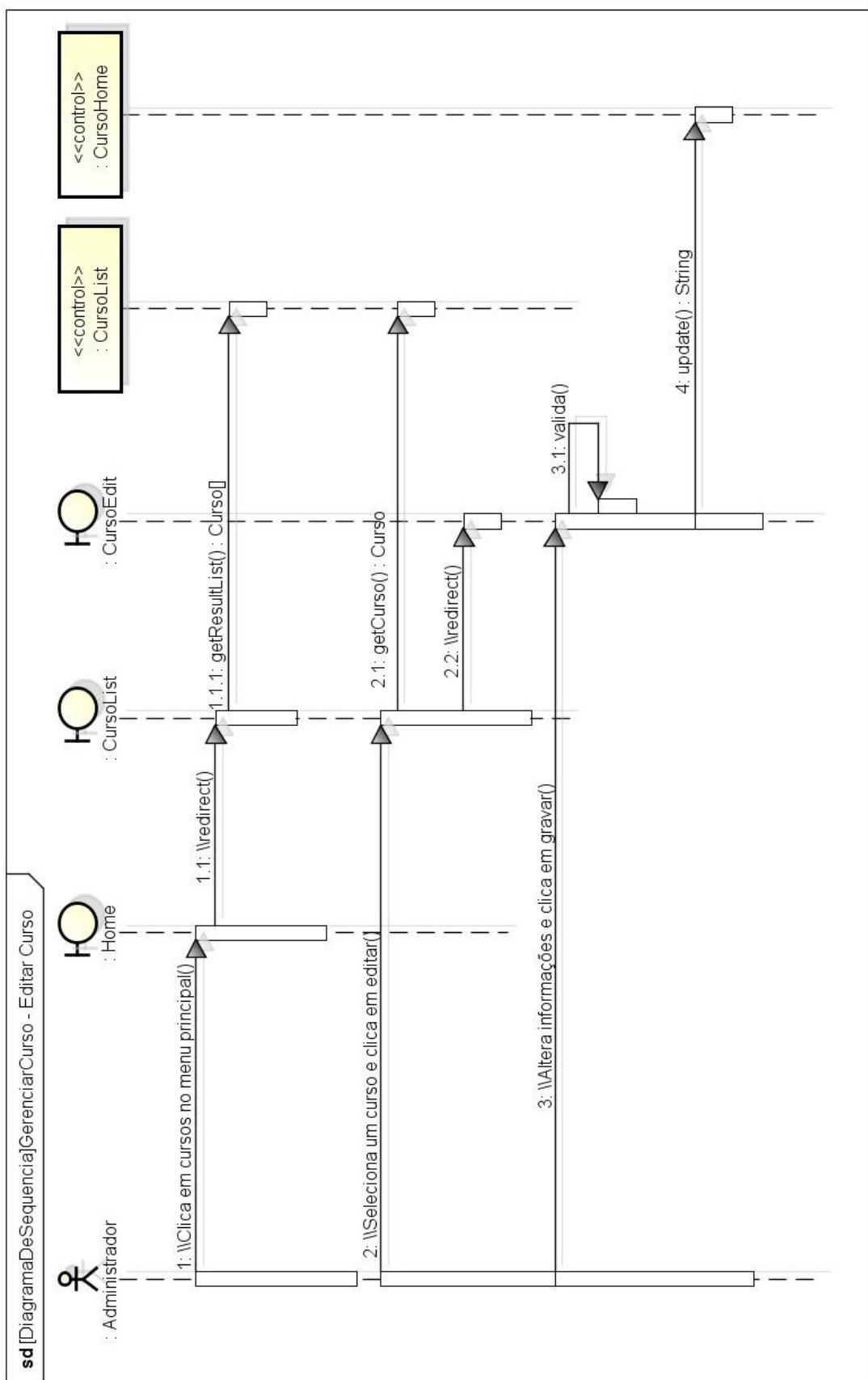


Fig. 34 – Diagrama de Seqüência – Editar Curso. **Fonte:** Elaborado pelos autores.

4.3 REMOVER CURSO

Caso de uso relacionado: Gerenciar Curso	
Descrição:	Remover um Curso.
Autor(es):	Administrador.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Cursos no menu principal.</p> <p>2 – Sistema exibe tela para gerenciar cursos com campo para buscar curso por nome.</p> <p>3 – Ator entra com o nome do curso e clica em buscar.</p> <p>4 – Sistema exibe lista com os cursos encontrados.</p> <p>5 – Ator seleciona um curso e clica em excluir.</p> <p>6 – Sistema exibe mensagem de confirmação.</p> <p>7 – Ator clica em confirmar.</p> <p>8 – Sistema exclui o curso e exibe mensagem de sucesso.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 35 – Fluxo de Eventos – Remover Curso. **Fonte:** Elaborado pelos autores.

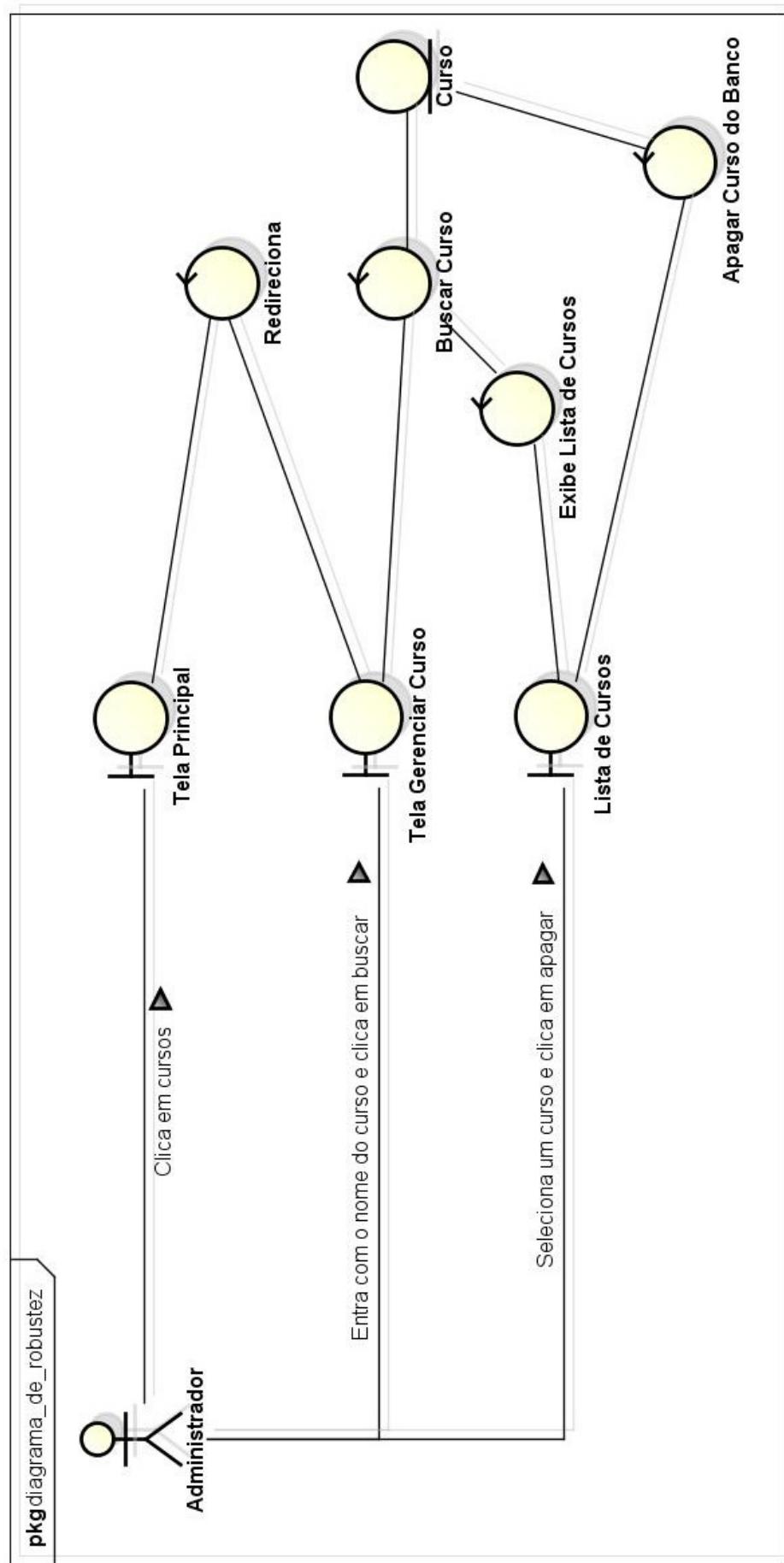


Fig. 36 – Diagrama de Robustez – Remover Curso. Fonte: Elaborado pelos autores.

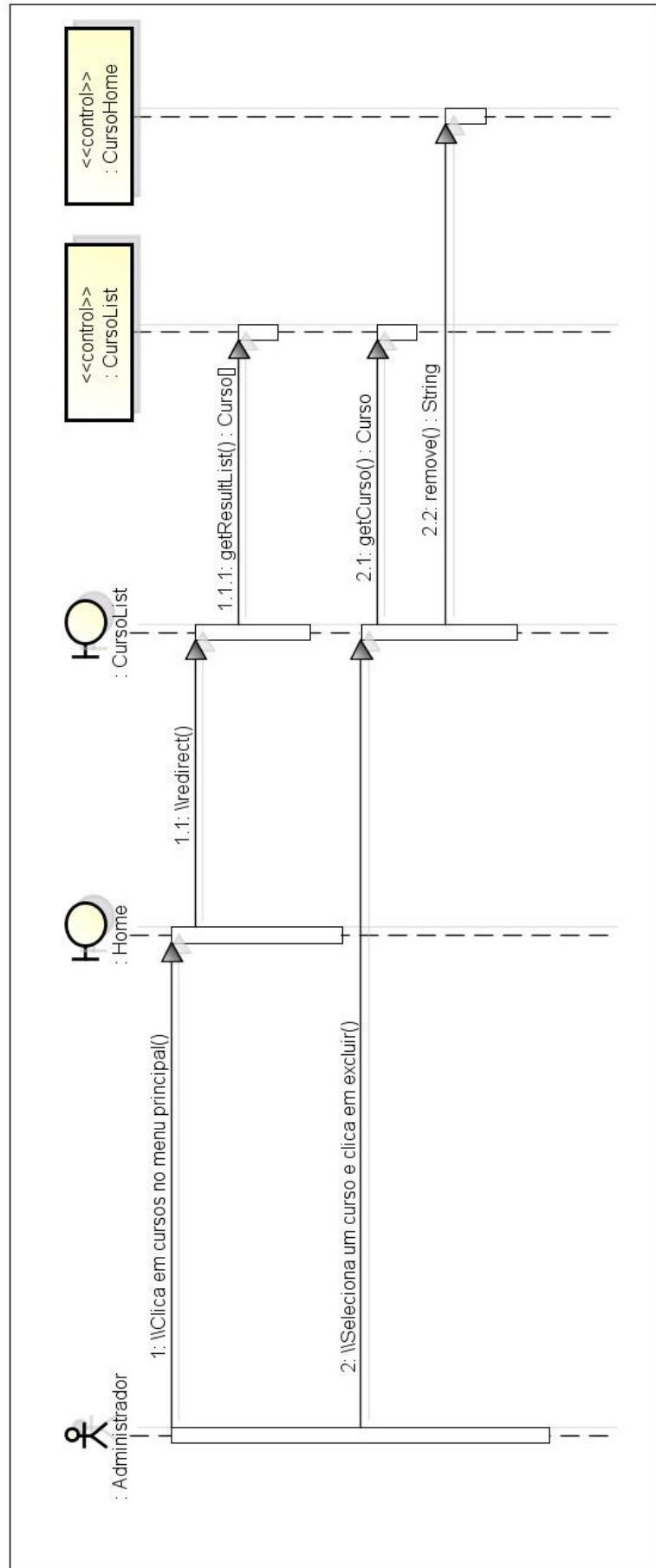


Fig. 37 – Diagrama de Sequência – Remover Curso. **Fonte:** Elaborado pelos autores.

5 ARTEFATOS DO CASO DE USO – ALTERAR INFORMAÇÕES PESSOAIS

Caso de Uso Relacionado: Alterar Informações Pessoais	
Descrição:	Alterar senha, e-mail.
Autor(es):	Administrador, Acadêmico.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator clica em Editar informações pessoais no menu principal.</p> <p>2 – Sistema exibe formulário para o ator alterar seus dados.</p> <p>3 – Ator altera suas informações e clica em gravar.</p> <p>4 – Sistema grava os dados e exibe mensagem de sucesso ao usuário.</p>
Fluxo Alternativo:	No item 3 do fluxo principal caso algum campo informado pelo ator esteja incorreto: <p>3.1 – O sistema exibe mensagem de erro.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 38 – Fluxo de Eventos – Alterar Informações Pessoais. **Fonte:** Elaborado pelos autores.

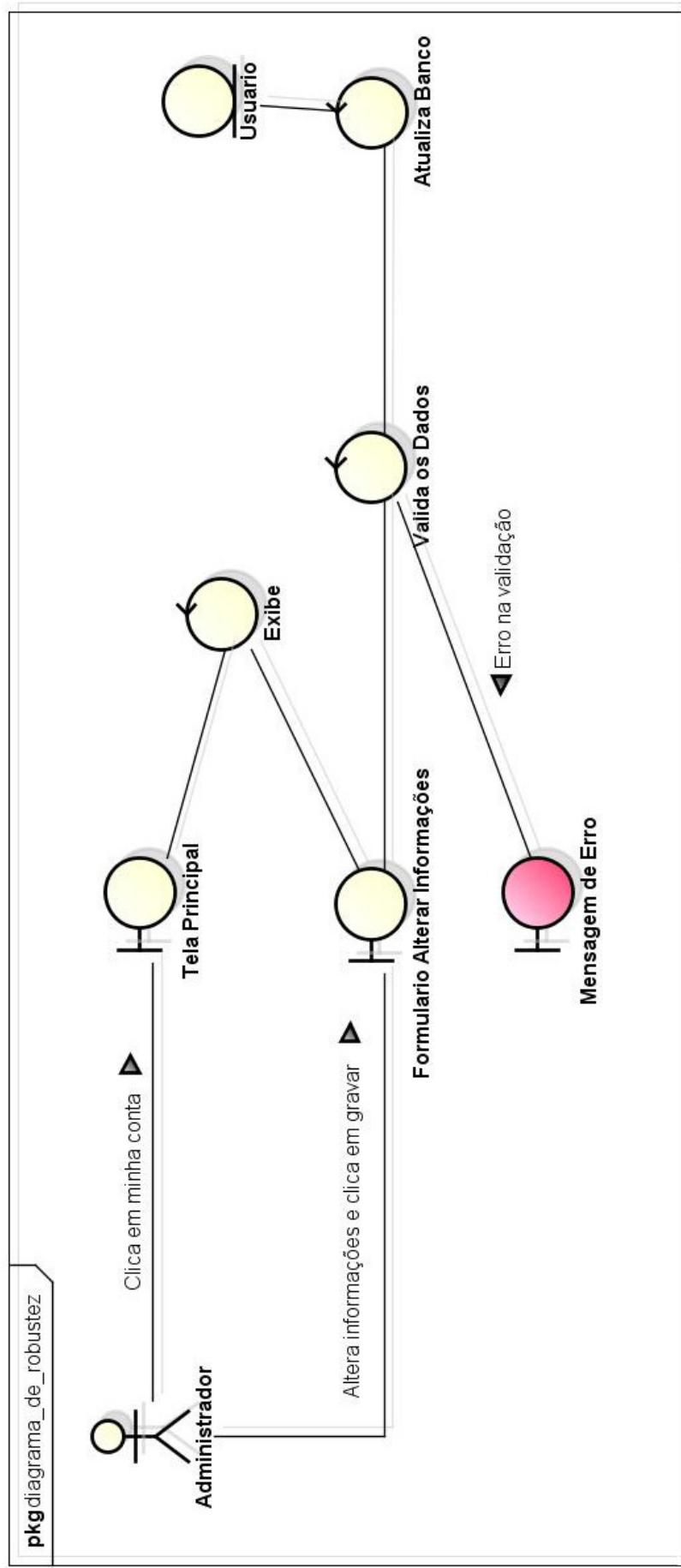


Fig. 39 – Diagrama de Robustez – Alterar Informações Pessoais. **Fonte:** Elaborado pelos autores.

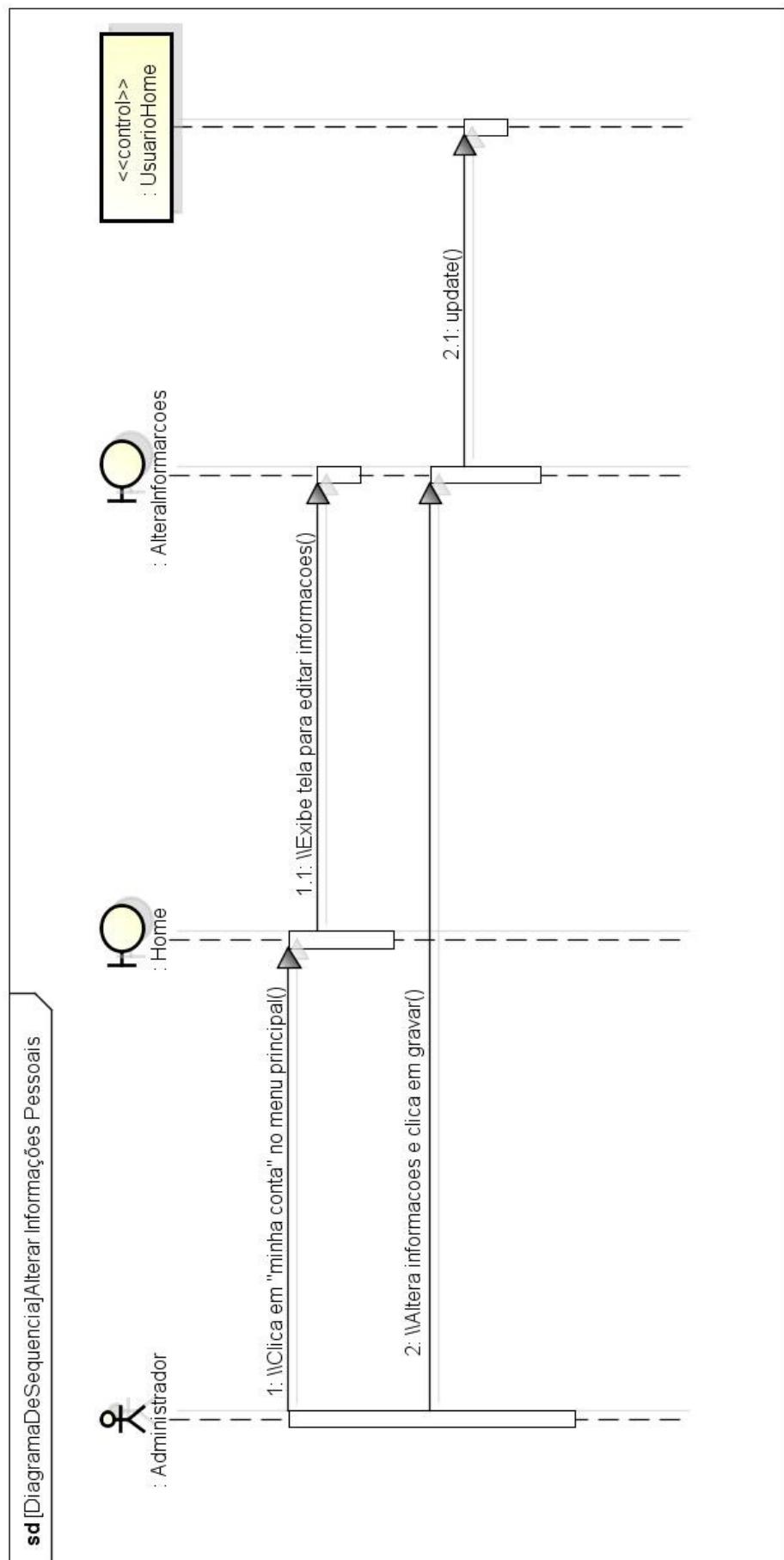


Fig. 40 – Diagrama de Seqüência – Alterar Informações Pessoais. **Fonte:** Elaborado pelos autores.

6 ARTEFATOS DO CASO DE USO – RECUPERAR SENHA

Caso de uso relacionado: Recuperar senha	
Descrição:	Recuperar uma Senha.
Autor(es):	Administrador, Acadêmico.
Fluxo Principal:	<p>1 - Ator clica em recuperar senha na tela inicial.</p> <p>2 - Sistema exibe tela com campo para o ator informar o seu e-mail.</p> <p>3 - Ator informa seu e-mail e clica em recuperar senha.</p> <p>4 - Sistema gera uma nova senha e envia por e-mail ao ator.</p>
Fluxo Alternativo:	No item 3 do fluxo principal caso o ator informe um e-mail que não está cadastrado sistema: 3.1 O sistema exibe mensagem de erro.
A Qualquer momento o ator pode cancelar a operação clicando no botão “Cancelar”.	

Fig. 41 – Fluxo de Eventos – Recuperar Senha. **Fonte:** Elaborado pelos autores.

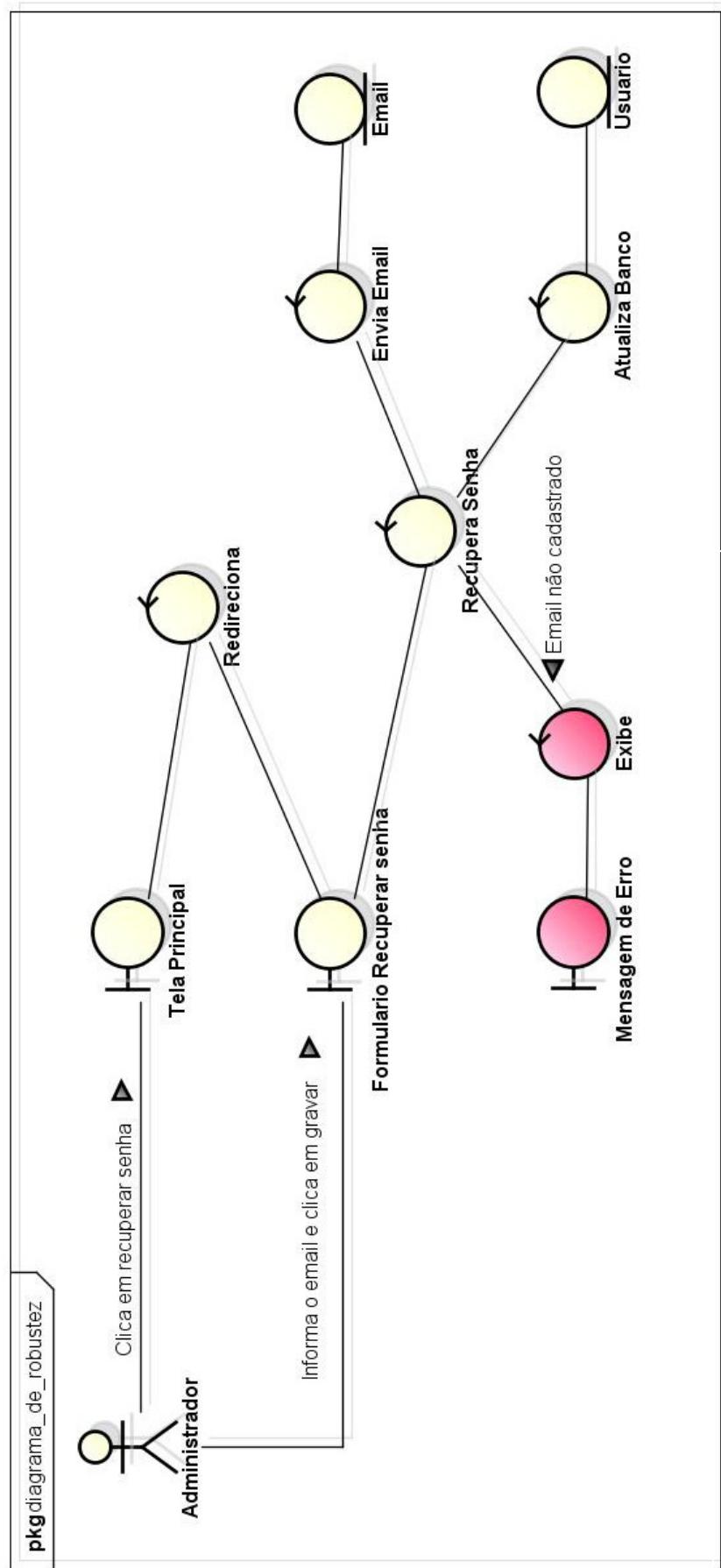


Fig. 42 – Diagrama de Robustez – Recuperar Senha. Fonte: Elaborado pelos autores.

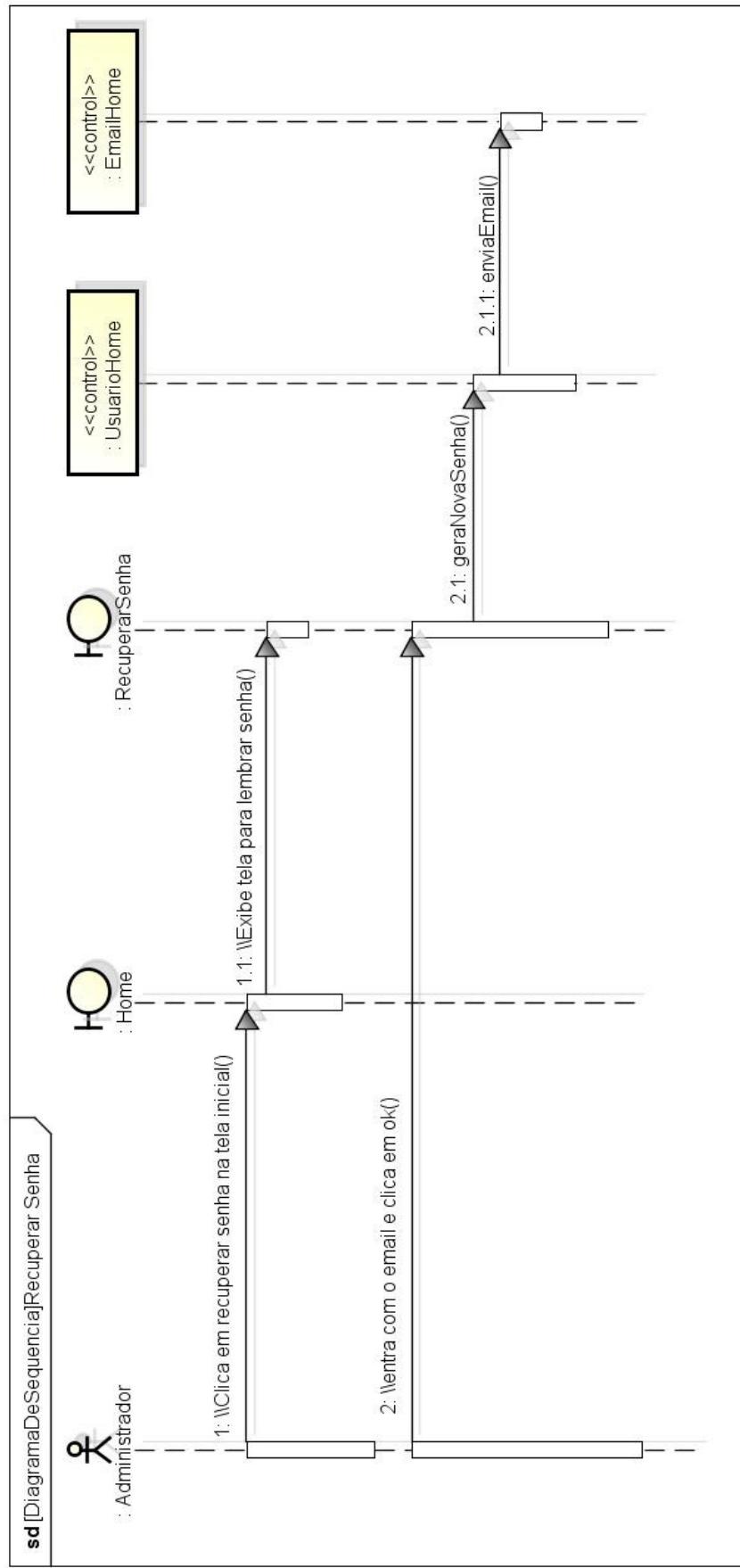


Fig. 43 – Diagrama de Sequência – Recuperar Senha. **Fonte:** Elaborado pelos autores.

7 ARTEFATOS DO CASO DE USO – PESQUISAR PROJETO

Caso de uso relacionado: Pesquisar Projeto	
Descrição:	Pesquisar por um Projeto no Sistema.
Autor(es):	Acadêmico.
Pré-condições	Estar autenticado no sistema.
Fluxo Principal:	<p>1 – Ator informa as palavras chaves na tela inicial do sistema e clica em pesquisar.</p> <p>2 – Sistema faz a busca nos índices e exibe tela com uma lista de resultados da pesquisa.</p> <p>3 – Ator seleciona um projeto e clica em editar.</p> <p>4 – Sistema exibe o projeto.</p>
A Qualquer momento o ator pode cancelar a operação clicando no botão “Sair”.	

Fig. 44 – Fluxo de Eventos – Pesquisar Projeto. **Fonte:** Elaborado pelos autores.

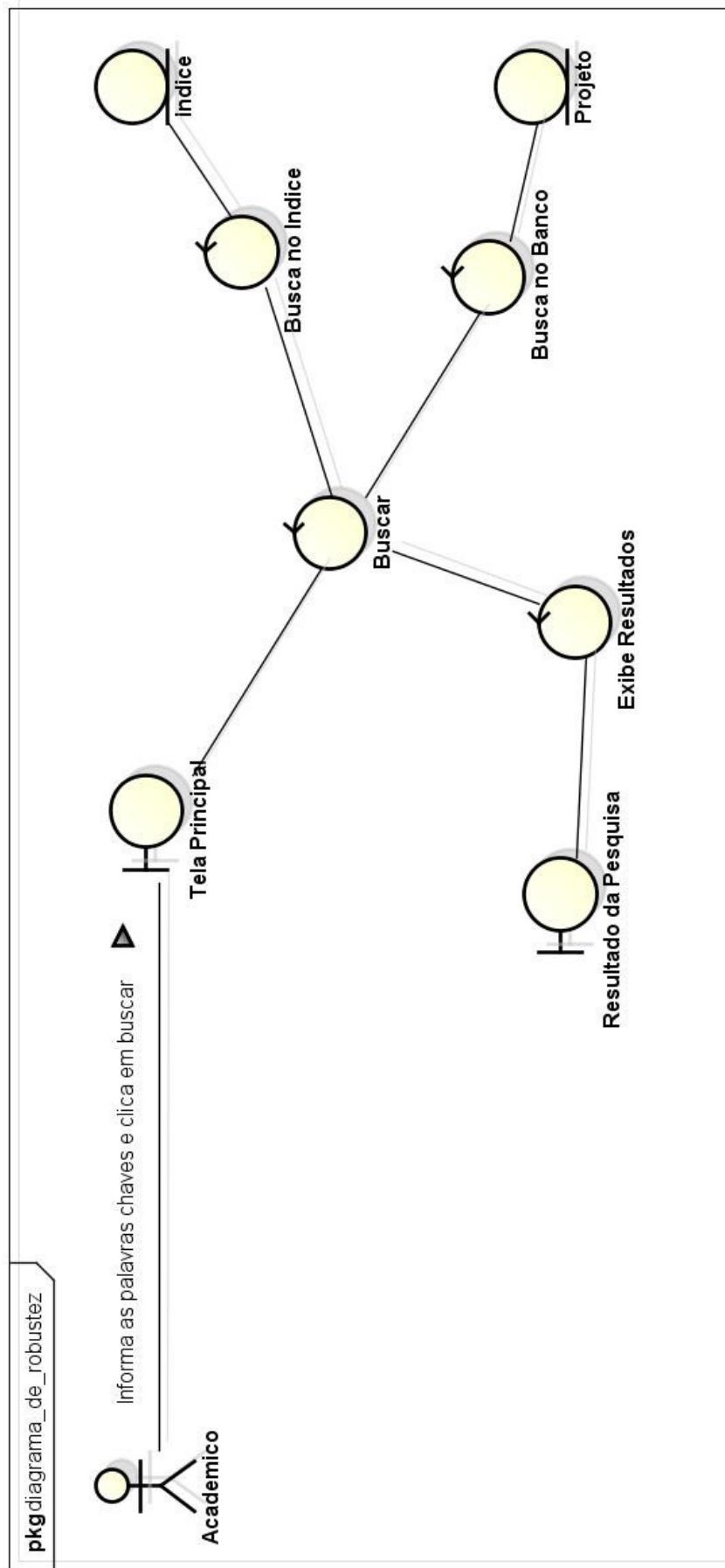


Fig. 45 – Diagrama de Robustez – Pesquisar Projeto. Fonte: Elaborado pelos autores.

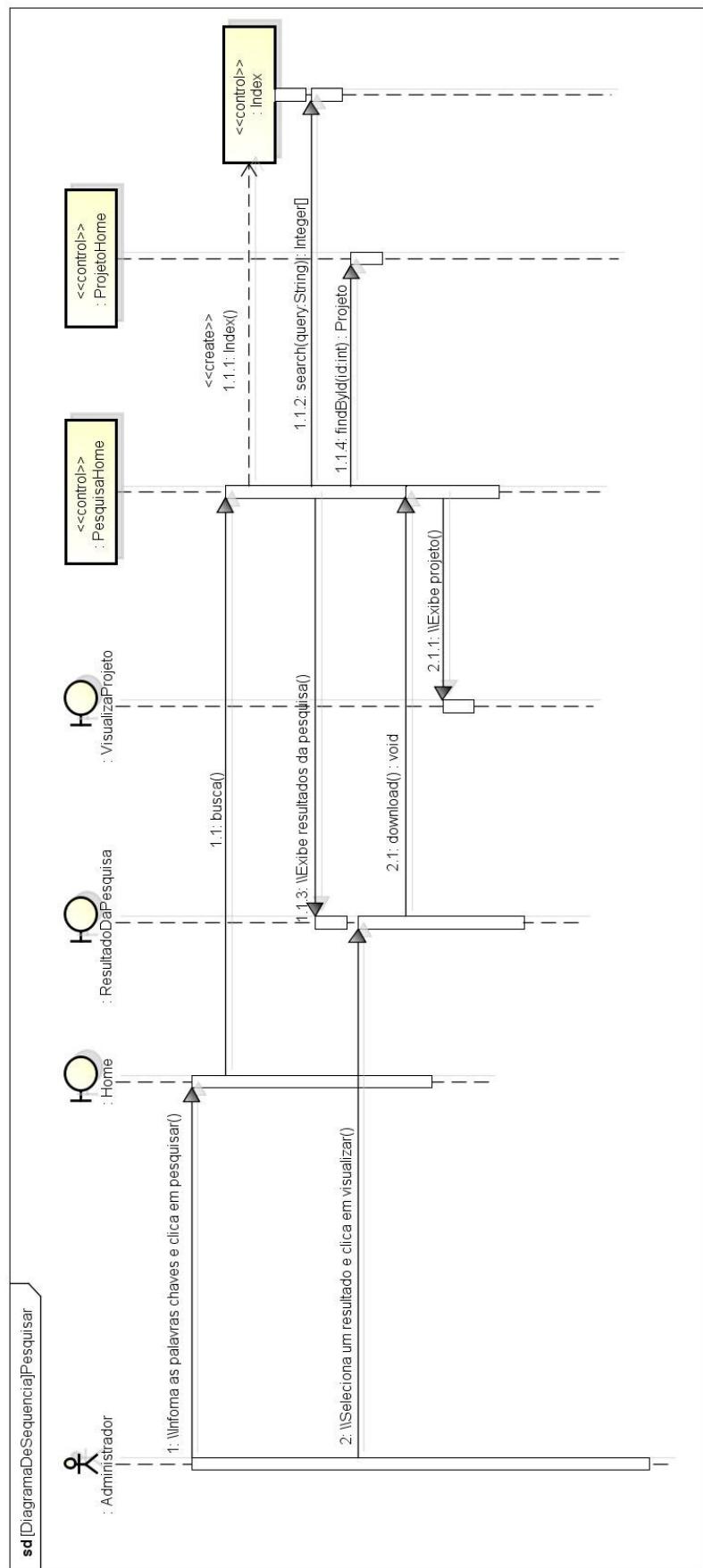


Fig. 46 – Diagrama de Sequência – Pesquisar Projeto. **Fonte:** Elaborado pelos autores.

PROTOTIPAÇÃO DAS INTERFACES

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

Home > Projeto > Cadastrar Projeto

Projeto

* campos obrigatórios

*Título:

*Autor(es):

*Orientador(es):

*Palavras Chave:

*Local:

*Data:

*Tipo:

▼

*Curso:

▼

*Arquivo:

Fig. 47 – Protótipo GUI Cadastro de Projeto. **Fonte:** Elaborado pelos autores.

Fig. 48 – Protótipo GUI Gerenciar Projetos. **Fonte:** Elaborado pelos autores.

Fig. 49 – Protótipo GUI Gerenciar Usuários. **Fonte:** Elaborado pelos autores.

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

[Home](#) > [Gerenciar Curso](#) > Cadastrar Curso

Curso _____

*Campo obrigatórios

*Nome:

[Gravar](#) [Cancelar](#)

Fig. 50 – Protótipo GUI Cadastrar Curso. **Fonte:** Elaborado pelos autores.

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

[Home](#) > [Gerenciar Curso](#)

[Cadastrar Curso](#)

Filtros _____

ID	Nome	Ações
01	Sistemas de informação	editar apagar
02	Letras	editar apagar

Fig. 51 – Protótipo GUI Gerenciar Cursos. **Fonte:** Elaborado pelos autores.

The screenshot shows the FindPro application's interface. At the top, there is a navigation bar with links: Projetos, Usuários, Cursos, Tipos de Projeto, Minha Conta, and Sair. Below the navigation bar, the URL Home > Gerenciar Tipos de Projeto is displayed. A button labeled 'Cadastrar Tipo de Projeto' is visible. A search bar labeled 'Filtros' is present. Below these elements is a table listing project types:

ID	Nome	Ações
01	Monografia	editar apagar
02	Artigo	editar apagar

Fig. 52 – Protótipo GUI Gerenciar Tipos de Projeto. **Fonte:** Elaborado pelos autores.

The screenshot shows the FindPro application's interface. At the top, there is a navigation bar with links: Projetos, Usuários, Cursos, Tipos de Projeto, Minha Conta, and Sair. Below the navigation bar, the URL Home > Gerenciar Tipos de Projeto > Cadastrar Tipo de Projeto is displayed. A label 'Tipo de Projeto' is followed by a text input field. Below the input field, there are two labels: '*Campo obrigatórios' and '*Nome:' followed by another text input field. At the bottom of the form are two buttons: 'Gravar' (Save) and 'Cancelar' (Cancel).

Fig. 53 – Protótipo GUI Cadastrar Tipo de Projeto. **Fonte:** Elaborado pelos autores.

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

[Home](#) > Minha Conta

Minha Conta

Email:

Senha:

Confirmação da senha:

Gravar

Cancelar

Fig. 54 – Protótipo GUI Alterar Informações Pessoais. **Fonte:** Elaborado pelos autores.

FindPro

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

[Home](#) > [Usuários](#) > Cadastrar usuário

Usuário _____

*Campos obrigatórios

*Nome:

*Email:

*Permissão:

Administrador ▾

Gravar **Cancelar**

Fig. 55 – Protótipo GUI Cadastrar Usuário. **Fonte:** Elaborado pelos autores.

[Projetos](#) | [Usuários](#) | [Cursos](#) | [Tipos de Projeto](#) | [Minha Conta](#) | [Sair](#)

FindPro

Titulo	Curso	Autor(es)	Orientador(es)	
Titulo 1	Sistemas de informação	autores	orientadores	visualizar

Fig. 56 – Protótipo GUI Resultados da Pesquisa. **Fonte:** Elaborado pelos autores.

MODELO DE DOMÍNIO

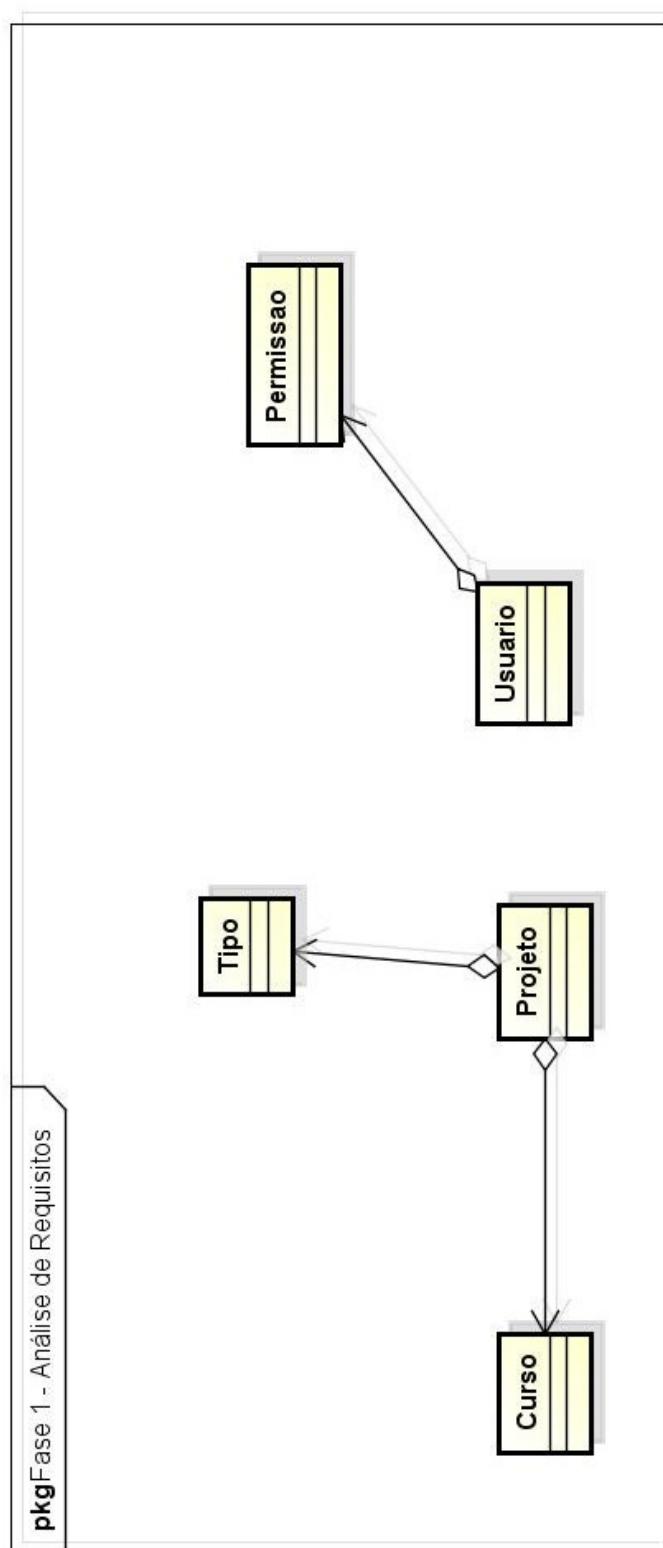


Fig. 57 – Modelo de Domínio. **Fonte:** Elaborado pelos autores.

MODELO DE DOMÍNIO ATUALIZADO

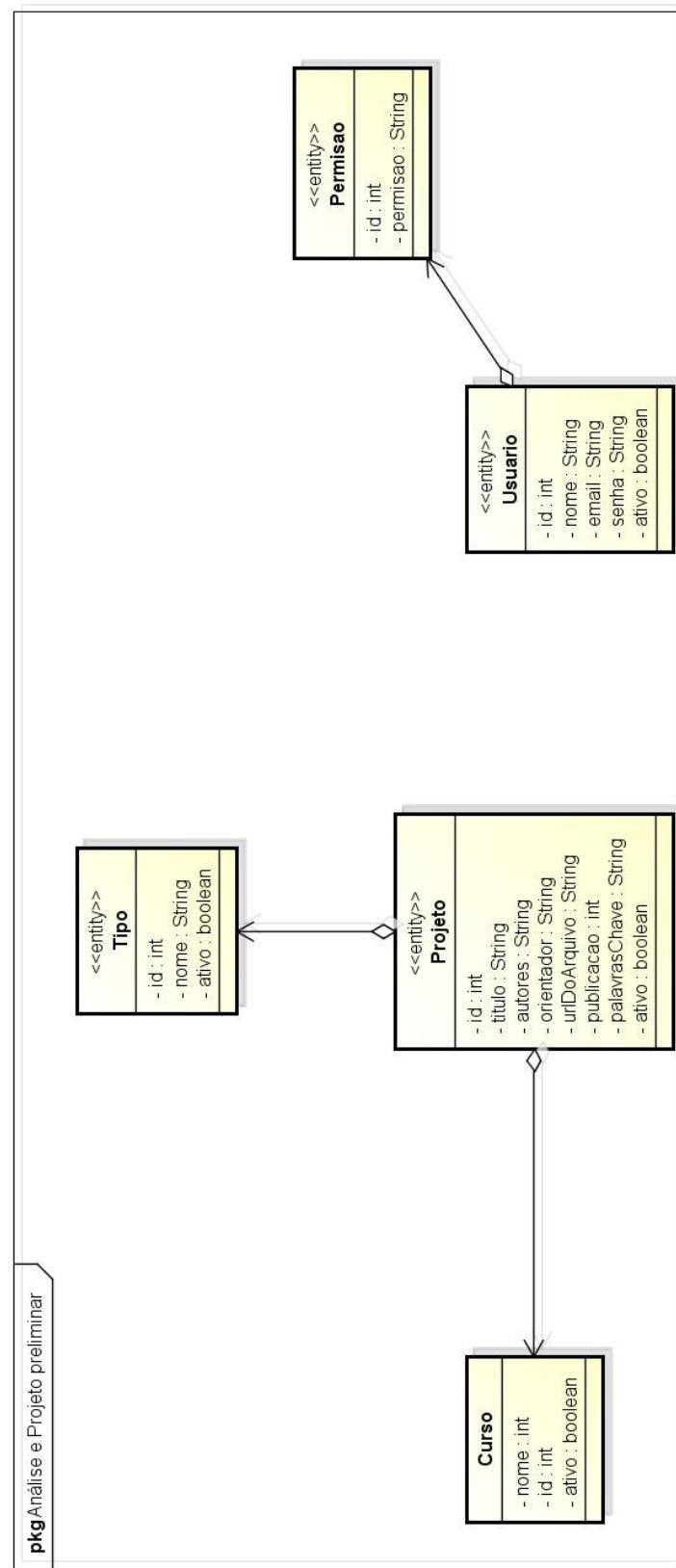


Fig. 58 – Modelo de Domínio Atualizado. Fonte: Elaborado pelos autores.

DIAGRAMA DO BANCO DE DADOS

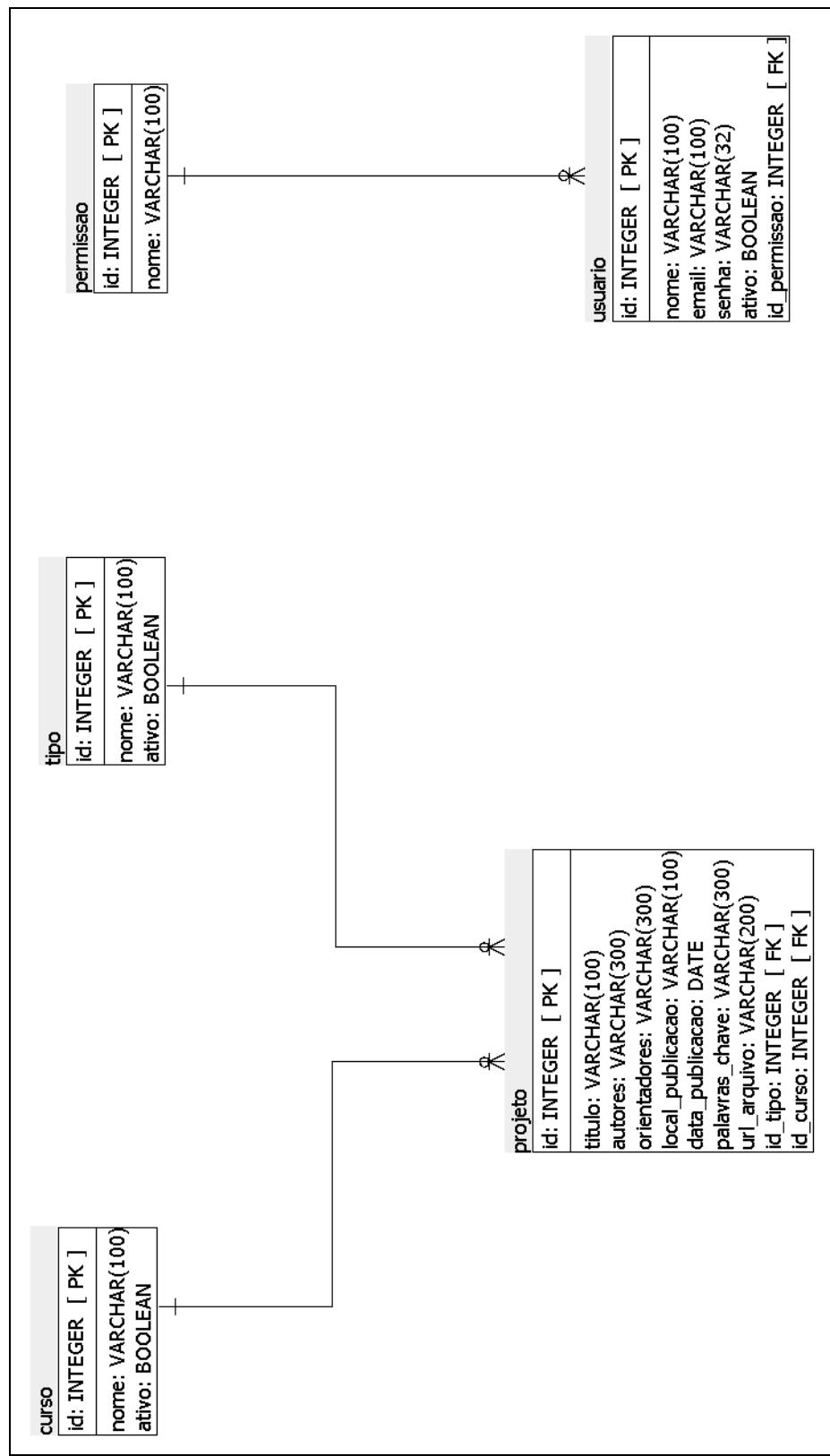


Fig. 59 – Diagrama do Banco de Dados. Fonte: Elaborado pelos autores.

DIAGRAMA DE CLASSES

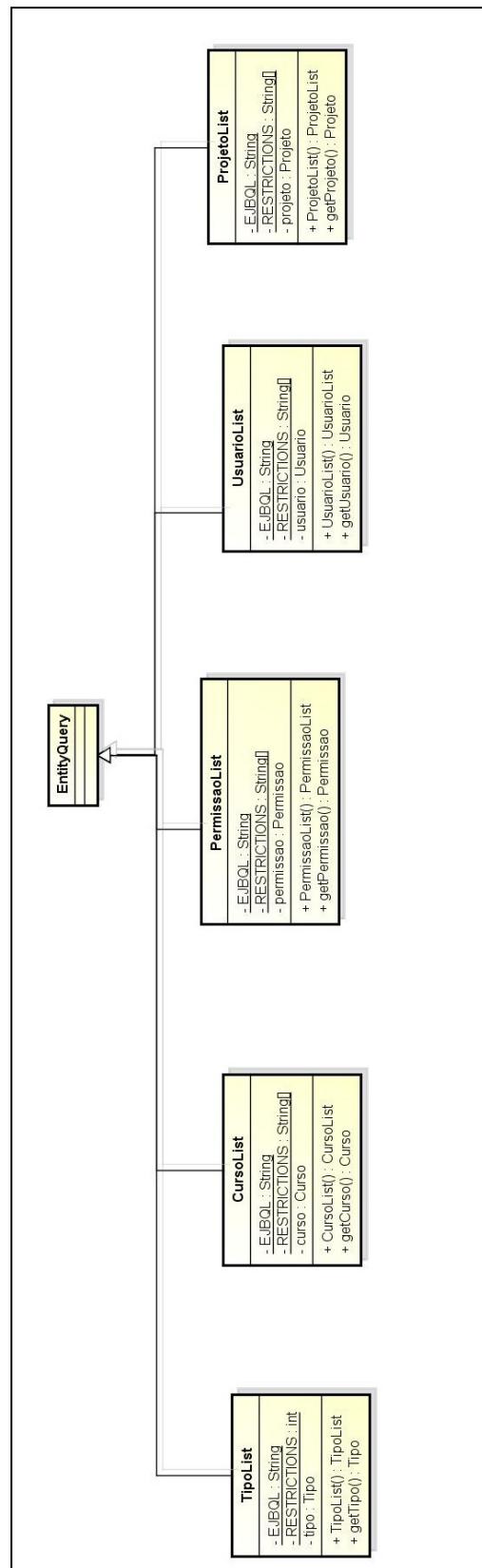


Fig. 60 – Diagrama do Classes - Parte 1. Fonte: Elaborado pelos autores.

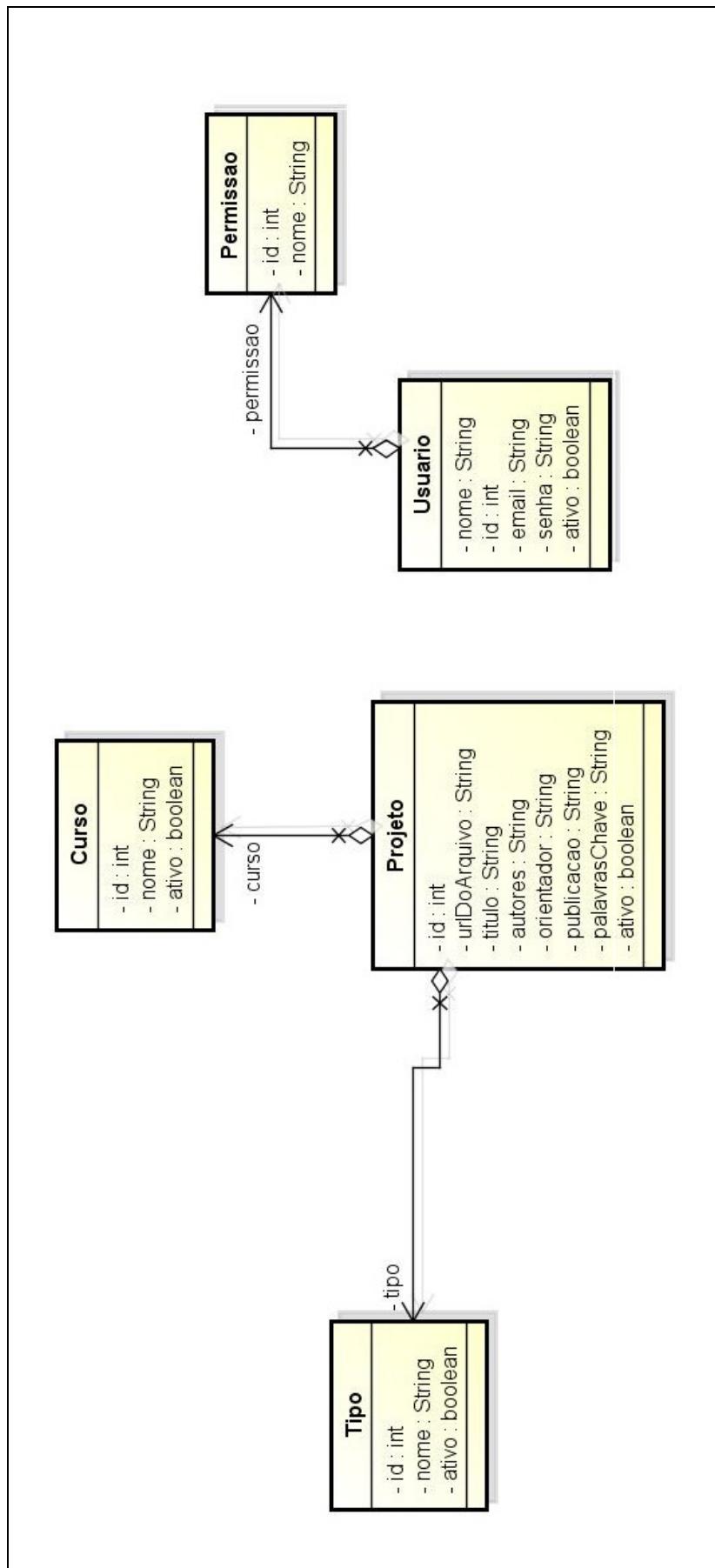


Fig. 61 – Diagrama do Classes - Parte 2. **Fonte:** Elaborado pelos autores.

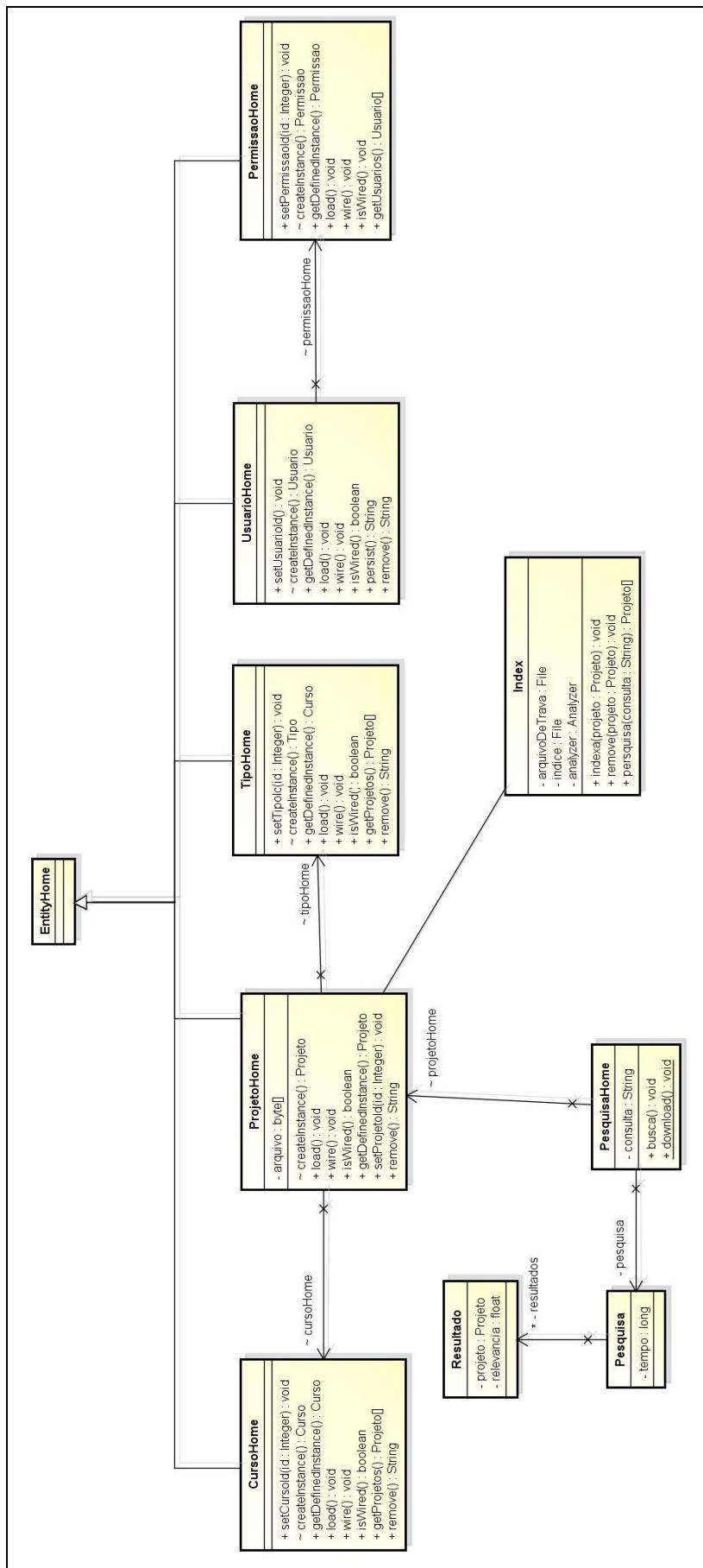


Fig. 62 – Diagrama das Classes - Parte 3. Fonte: Elaborado pelos autores.

APÊNDICE II
TUTORIAL APACHE LUCENE

TUTORIAL APACHE LUCENE

O Apache Lucene é uma biblioteca de pesquisa extremamente rica e poderosa, totalmente escrita em Java. Pode-se usar o Lucene para realizar a indexação de textos completos em objetos no banco de dados ou documentos de vários formatos (DOC, PDF, HTML, TXT e assim por diante).

Neste tutorial, vamos passar as noções básicas do uso do Lucene para adicionar a funcionalidade de pesquisa para JEE. O objeto principal deste tutorial é a classe Hotel, que possui um ID, um nome, uma cidade, e uma descrição.

De maneira geral, realizar a pesquisa de texto completo usando o Lucene requer duas etapas: (1) criar os índices do lucence sobre os documentos; (2) analisar e procurar nos índices pré-construídos resultados para a consulta do usuário.

Na primeira parte deste tutorial, vamos ensinar a criar os índices do Lucene. Na segunda parte, ensinaremos como usar os índices pré-construídos para responder as requisições.

Instalação

Em primeiro lugar, deve-se realizar o download do Lucene, que pode ser encontrado no endereço:

<http://www.apache.org/dyn/closer.cgi/lucene/java/>.

Após o download, basta descompactar o arquivo ZIP e colocar o lucene-core-X.Y.Z.jar dentro da pasta de bibliotecas externas de seu projeto Java.

1 CRIANDO O ÍNDICE

O primeiro passo na implementação de pesquisa de texto completo com o Lucene é a construção de um índice. Este índice armazena objetos do tipo **Document**, e é trabalho do desenvolvedor converter os arquivos (PDF, DOC, TXT) neste tipo de entidade e armazená-los no arquivo de índices como objetos do tipo nome/valor. Assim que um **Document** é finalizado, utiliza-se um objeto **IndexWriter** para gravá-lo. Agora vamos entrar em detalhes sobre como isso é feito.

A Classe IndexWriter: Criando os Índices

Para criar um índice, a primeira coisa que precisamos fazer é instanciar um objeto do tipo **IndexWriter**. Os objetos deste tipo são responsáveis por criar o índice e adicionar a ele novos objetos **Document**.

```
IndexWriter indexWriter = new IndexWriter("index-directory",
new StandardAnalyzer(), true);
```

O primeiro parâmetro especifica o diretório no qual o Lucene criará os índices, que neste caso é `index-directory`. O segundo parâmetro especifica o “document parser” ou “document analyzer”, que será utilizado pelo Lucene quando ele indexar seus dados. Aqui, nós estamos utilizando `StandardAnalyzer` para esse propósito. Mais detalhes sobre os tipos de analyzers serão abordados posteriormente. E o terceiro parâmetro indica ao Lucene que, o arquivos de índices deve ser criado caso ainda não exista.

A Classe Analyzer: Parseando os Documentos

O trabalho da classe **Analyzer** é “parsear” cada campo dos documentos dentro de campos indexados ou palavras chave. Vários tipos de **Analyzer** são oferecidos pelo Lucene, a Tabela a seguir mostra os mais importantes.

Analyzer	Descrição
StandardAnalyzer	Um Analyzer sofisticado de propósito geral.
WhitespaceAnalyzer	Um Analyzer muito simples que somente separa índices dos espaços em branco.
StopAnalyzer	Remove palavras simples da língua inglesa que não tem muita importância na indexação como, por exemplo, (o, a, os, as).
SnowballAnalyzer	Um interessante Analyzer experimental que trabalha com palavras primitivas (Ex.: uma procura pela palavra chuva retorna resultados como chover, chuvisco, chuvarada).

Existe uma série de analyzers específicos para cada idioma, incluindo analyzers para o Português, Alemão, Francês e idiomas asiáticos. Também existe a possibilidade de criar seu próprio **Analyzer**, embora o **StandardAnalyzer** seja o suficiente na maioria dos casos. Quando criamos uma classe **IndexWriter**, devemos especificar qual o tipo de **Analyzer** iremos utilizar, como feito anteriormente.

Adicionando Objetos Document no Índice

Agora, precisamos adicionar nossos objetos **Document** ao índice. Para indexar uma instância de **Document** precisamos, em primeiro lugar, identificar quais campos dos arquivos de texto desejamos que sejam indexados. Como mencionado anteriormente, a classe **Document** é apenas um *container* para uma série de campos indexados no formato nome/valor. A Tabela a seguir ilustra como esse processo é realizado.

```

Document doc = new Document();

doc.add(new Field("descricao", hotel.getDescricao(),
Field.Store.YES, Field.Index.TOKENIZED));

```

Para adicionar um campo a um objeto **Document**, devemos instanciar um objeto do tipo **Field**. A classe **Field** é composta de um nome e um valor (que aqui são representados pelos dois primeiros parâmetros enviados ao construtor). O valor deve ser do tipo **String** ou do tipo **Reader** (caso o objeto a ser indexado seja um arquivo). Os últimos dos parâmetros enviados ao construtor são usados para determinar como o campo será armazenado e indexado nos índices do Lucene.

- **Armazenando o valor:** O terceiro parâmetro especifica se o nome do campo precisa ser armazenado nos índices ou pode ser descartado depois da indexação. Armazenar o nome é de grande utilidade quando precisamos dele posteriormente, como, por exemplo, se desejássemos procurar um arquivo somente por esta propriedade. (Ex.: descricao:Hotel Califórnia, o Lucene retornaria somente os arquivos com esta descrição, anteriormente registrada);
- **Indexando o valor:** O quarto parâmetro especifica se, e como o valor deve ser indexado. Um bom exemplo de quando desejamos que o valor seja armazenado, mas não indexado, é quando o campo representa um ID de um objeto do banco de dados. Nesse caso, não faz sentido indexar este valor com o Lucene, mas provavelmente ele deve ser armazenado para recuperarmos o objeto do banco de dados posteriormente. Quando não necessitamos indexar um campo utilizamos a constante **Field.Index.NO**. Mas na grande maioria dos casos, iremos utilizar o valor **Field.Index.TOKENIZED**, que significa que o campo vai ser indexado utilizando o **Analyzer** passado ao **IndexWriter** como parâmetro. Caso desejássemos que um valor fosse indexado sem que ele seja analisado pelo **Analyzer**, utilizamos o valor **Field.Index.UN_TOKENIZED**, mas, neste caso, o valor será utilizado com está.

Voltando para o exemplo com hotéis, iremos utilizar um sistema de buscas bastante simples que trabalhará com os seguintes campos:

- **ID do Hotel:** com esse campo, poderemos recuperar o Hotel de um possível banco de dados.
- **Nome do Hotel:** esse campo será mostrado nos resultados da busca.
- **Descrição do Hotel:** caso queiramos que este campo apareça nos resultados da busca.
- Um texto com a descrição total de campos importantes do Hotel como **nome, cidade e descrição**.

A seguir, está o método responsável pela indexação em nossa classe **Indexer**, repare que para este método passamos um **Hotel** como parâmetro:

```
public void indexHotel(Hotel hotel) throws IOException {
    IndexWriter writer = getIndexWriter(false);
    Document doc = new Document();
    doc.add(new Field("id", hotel.getId(), Field.Store.YES,
                      Field.Index.NO));
    doc.add(new Field("nome", hotel.getNome(), Field.Store.YES,
                      Field.Index.TOKENIZED));
    doc.add(new Field("cidade", hotel.getCidade(), Field.Store.YES,
                      Field.Index.UN_TOKENIZED));
    doc.add(new Field("descricao",
                      hotel.getDescricao(), Field.Store.YES, Field.Index.TOKENIZED));
    String fullSearchableText = hotel.getNome()
        + " " + hotel.getCidade() + " " + hotel.getDescricao();
    doc.add(new Field("content", fullSearchableText, Field.Store.NO,
                      Field.Index.TOKENIZED));
    writer.addDocument(doc);
}
```

Assim que a indexação é finalizada, temos que fechar o **IndexWriter**, o qual atualiza e salva os arquivos associados ao índice no disco. Abrir e fechar objetos **IndexWriter** consome recursos consideráveis de memória, tempo e processamento, então, não é uma ideia muito boa realizar estas tarefas em cada operação de indexação. A seguir, temos um método que utiliza apenas um **FileWriter** para completar toda a operação:

```

public void rebuildIndexes() throws IOException {
    //
    // Apaga o índice existente
    //
    getIndexWriter(true);
    //
    // Indexa todos os objetos Hotel
    //
    Hotel[] hoteis = HotelDatabase.getHoteis();
    for(Hotel hotel: hoteis) {
        indexHotel(hotel);
    }
    //
    // Não se esqueça de fechar o IndexWriter quando acabar
    //
    closeIndexWriter();
}

```

E para referência, a seguir disponibilizamos todo o código da classe **Indexer**:

```

package lucene.demo.search;

import java.io.IOException;
import java.io.StringReader;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Document;
import lucene.demo.business.Hotel;
import lucene.demo.business.HotelDatabase;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.IndexWriter;

public class Indexer {

    public Indexer() {
        //Construtor
    }

    private IndexWriter indexWriter = null;

    public IndexWriter getIndexWriter(boolean create) throws IOException {

        if (indexWriter == null) {
            indexWriter = new IndexWriter("index-directory",
                new StandardAnalyzer(),create);
        }
        return indexWriter;
    }

    public void closeIndexWriter() throws IOException {
        if (indexWriter != null) {
            indexWriter.close();
        }
    }
}

```

```

public void indexHotel(Hotel hotel) throws IOException {
    System.out.println("Indexing hotel: " + hotel);
    IndexWriter writer = getIndexWriter(false);
    Document doc = new Document();

    doc.add(new Field("id", hotel.getId(), Field.Store.YES,
        Field.Index.NO));

    doc.add(new Field("name", hotel.getName(), Field.Store.YES,
        Field.Index.TOKENIZED));

    doc.add(new Field("city", hotel.getCity(), Field.Store.YES,
        Field.Index.UN_TOKENIZED));

    doc.add(new Field("description", hotel.getDescription(),
        Field.Store.YES, Field.Index.TOKENIZED));

    String fullSearchableText = hotel.getName() + " " +
        hotel.getCity() + " " + hotel.getDescription();

    doc.add(new Field("content", fullSearchableText, Field.Store.NO,
        Field.Index.TOKENIZED));

    writer.addDocument(doc);
}

public void rebuildIndexes() throws IOException {
    //
    // Apaga o índice existente
    //
    getIndexWriter(true);
    //
    // Indexa todos os objetos Hotel
    //
    Hotel[] hotels = HotelDatabase.getHotels();
    for(Hotel hotel : hotels) {
        indexHotel(hotel);
    }
    //
    // Não se esqueça de fechar o IndexWriter quando acabar
    //
    closeIndexWriter();
}
}

```

2 PROCURA NOS ÍNDICES

Agora que nós já temos nossos dados indexados podemos realizar algumas buscas. Neste tutorial, esta etapa é implementada na classe **SearchEngine**.

Na maioria dos casos precisamos utilizar somente duas classes para realizar uma busca textual com o Lucene: **QueryParser** e **IndexSearcher**. A classe **QueryParser** transforma as palavras contidas na busca do usuário em um objeto **Query** utilizado pelo Lucene, o qual é passado para o método **search()** da classe **IndexSearcher** como entrada. Baseado neste objeto **Query** e nos índices construídos anteriormente pelo Lucene, o método **search()** identifica os objetos **Document** correspondentes à procura e retorna todos eles como um único objeto **Hits**. Como pode ser visto na tabela a seguir:

```
package lucene.demo.search;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import lucene.demo.business.Hotel;
import lucene.demo.business.HotelDatabase;
import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;

public class SearchEngine {
    private IndexSearcher searcher = null;
    private QueryParser parser = null;

    /** Creates a new instance of SearchEngine */
    public SearchEngine() throws IOException {
        searcher = new IndexSearcher("index-directory");
        parser = new QueryParser("content", new StandardAnalyzer());
    }

    public Hits performSearch(String queryString)
    throws IOException, ParseException {
        Query query = parser.parse(queryString);
        Hits hits = searcher.search(query);
        return hits;
    }
}
```

Dentro do construtor da classe **SearchEngine**, nós criamos, primeiramente, um objeto **IndexSearcher** utilizando o diretório `index-directory` foi anteriormente criado. Em seguida criamos um **QueryParser**. O primeiro parâmetro enviado para seu construtor serve para especificar o campo padrão para procura, que é o campo `content` neste caso. O campo `content` serve como um campo padrão quando não se especifica o campo de pesquisa como, por exemplo, o nome, ou a cidade. Já o segundo parâmetro especifica o **Analyzer** que deve ser utilizado pelo **QueryParser** para parsear a **String** informada para procura.

A classe **SearchEngine** possuí um método chamado **performSearch()** que recebe uma **String** contendo as palavras que se deseja procurar e, devolve um lista dos objetos **Document** encontrados, todos eles encapsulados em uma instância do objeto **Hits**. O método pega a **String** de pesquisa, parseia esta **String** utilizando o objeto **QueryParser** e realiza a procura por meio do método **search()** classe **IndexSearcher**.

Nota Importante: Há um erro muito comum cometido pelos desenvolvedores em realação a classe **Analyzer**. Quando utilizamos o Lucene, nós especificamos o **Analyzer** duas vezes. Uma vez quando criamos o objeto **IndexWriter** (para a construção dos índices) e outra quando criamos um **QueryParser**. Note que, é imprescindível que usemos o mesmo tipo de **Analyzer** nestas duas vezes. Neste tutorial nós criamos o **IndexWriter** e o **QueryParser** utilizando o **StandardAnalyzer**.

De maneira simples, a **String** procura pode ser um lista simples de palavras como Hotel Califórnia. Esta procura retornaria tanto objetos **Document** que contenham Hotel quanto objetos que contenham Califórnia. Se desejássemos documentos que contivessem as duas palavras (independentemente de estarem juntas), teríamos que utilizar a palavra AND (em letras maiúsculas) entre elas, desta maneira: Hotel AND Califórnia. O operador OR também poderia ser utilizado, retornando os mesmos resultados da pesquisa sem o AND. Podemos também procurar por **Documents** utilizando as propriedades previamente indexadas como: nome:Marriott OR descricao:Confortável.

Recuperando os Documents Encontrados

O método **search()** da classe **IndexSearcher** retorna um lista de objetos **Document** encapsulados em um objeto **Hits**. Este objeto contém uma lista de objetos **Hit**, organizados por ordem de relevância. Dos objetos **Hits**, podemos obter os objetos **Document** como mostrado a seguir:

```
SearchEngine se = new SearchEngine();

Hits hits = se.performSearch("Hilton Hotel");

for(int i = 0; i < hits.length(); i++) {

    Document doc = hits.doc(i);

    String nomeHotel = doc.get("nome");

    ...
}
```

Como no exemplo anterior, assim que obtemos o objeto **Document**, podemos recuperar os valores indexados por meio do método **get()**. Outra possibilidade seria utilizar um **Iterator**, como no exemplo a seguir:

```
SearchEngine se = new SearchEngine();

Hits hits = se.performSearch("Hilton Hotel");

System.out.println("Resultados Encontrados: " + hits.length());

Iterator<Hit> iter = hits.iterator();

while(iter.hasNext()){
    Hit hit = iter.next();
    Document doc = hit.getDocument();
    System.out.println(doc.get("nome")
                      + " " + doc.get("cidade")
                      + " (" + hit.getScore() + ")");
}
```

No exemplo acima, podemos observar como o objeto **Hit** pode ser utilizado para buscar o **Document** correspondente, como também a valor de relevância dos objetos encontrados nesta procura. O **Score** nos dá uma ideia da pertinência dos resultados em cada documento, como no exemplo a seguir:

```
Resultados encontrados: 9
Hotel Notre Dame Paris (0.5789772)
Hotel Odeon Paris (0.40939873)
Hotel Tonic Paris (0.34116563)
Hotel Bellevue Paris (0.34116563)
Hotel Marais Paris (0.34116563)
Hotel Edouard VII Paris (0.16353565)
Hotel Rivoli Paris (0.11563717)
Hotel Trinité Paris (0.11563717)
Clarion Cloitre Saint Louis Hotel Avignon (0.11563717)
```

APÊNDICE III
TUTORIAL DE INTEGRAÇÃO LUCENE-PDFBOX

TUTORIAL DE INTEGRAÇÃO LUCENE-PDFBOX

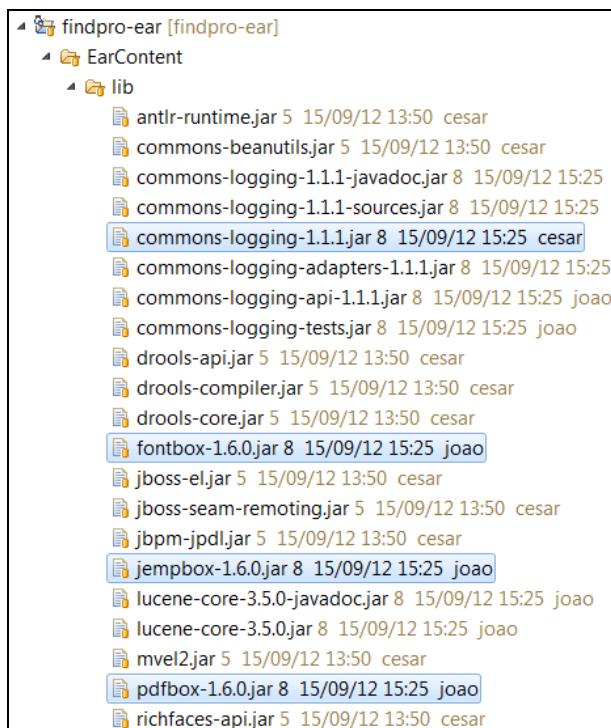
O Apache PDFBox é uma biblioteca Java de código fonte aberto para trabalhar com documentos PDF. Esta ferramenta permite a criação de novos documentos PDF, manipulação de documentos existentes e também a capacidade de extrair seu conteúdo.

Neste trabalho utilizamos esta ferramenta para extrair o texto dos trabalhos acadêmicos, visto que o *framework* Apache Lucene trabalha somente como indexador e buscador de texto puro e não oferece nenhuma ferramenta para extrair este texto de qualquer tipo de documento.

Para integrarmos estas duas ferramentas, Apache PDFBox e Apache Lucene, inicialmente realizamos o *download* do PDFBox no seguinte endereço:

<http://pdfbox.apache.org/download.html#pdfbox>

Em seguida, adicionamos seus arquivos ao projeto FindPro conforme demonstra a Figura a seguir, onde as bibliotecas em destaque pertencem ao PDFBox:



A Figura mostra quatro arquivos que devem ser adicionados para que o PDFBox funcione na aplicação. Já a inserção dos arquivos do *framework* Apache Lucene é abordada no tutorial do Apêndice II e não serão discutidos neste passo-a-passo.

Assim que todas as bibliotecas necessárias para a extração do conteúdo do documento (PDFBox), indexação e recuperação do texto (Lucene) foram adicionadas ao projeto, inicia-se o processo de criação do motor indexador e buscador de conteúdo textual.

A primeira etapa da criação destas funcionalidades, foi o desenvolvimento de um método capaz de retirar o texto dos documentos em PDF conforme pode ser observado a seguir:

```
public String extrairTexto(String caminhoArquivo)
{
    String textoExtraido = null;

    File doc = new File(caminhoArquivo);

    PDDocument pdDoc;

    try
    {
        pdDoc = PDDocument.load(doc);
        PDFTextStripper stripper = new PDFTextStripper();
        textoExtraido = stripper.getText(pdDoc);
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    return textoExtraido;
}
```

A classe **PDDocument** é uma representação de um documento em PDF na memória, ela serve de parâmetro para a classe **PDFTextStripper** responsável por retirar todo o texto do arquivo.

No caso deste sistema de recuperação, uma classe, chamada **Project2Document**, foi desenvolvida para realizar a extração e indexação do texto dos documentos. Além do método exibido na Figura acima, um construtor e mais um método foram adicionados a ela como poderá ser visto na Figura a seguir.

```

import org.apache.lucene.document.DateTools;
import org.apache.lucene.document.DateTools.Resolution;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.pdfbox.cos.COSDocument;
import org.apache.pdfbox.pdfparser.PDFParser;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.util.PDFTextStripper;
import org.domain.findpro.entity.Projeto;

public class Project2Document{

    private Projeto projeto;

    // Construtor
    public Project2Document(Projeto projeto) {
        this.projeto = projeto;
    }

    public Document parse()
    {
        Document document = new Document();

        // Adiciona ao índice o campo ID da entidade Projeto
        document.add(new Field(Fields.ID.getNome(),
                            projeto.getId().toString(),
                            Field.Store.YES,
                            Field.Index.NOT_ANALYZED));

        // Adiciona ao índice o nome dos Autores do Projeto
        document.add(new Field(Fields.AUTORES.getNome(),
                            projeto.getAutores(),
                            Field.Store.NO,
                            Field.Index.ANALYZED));

        // Adiciona ao índice o nome do Curso do Projeto
        document.add(new Field(Fields.CURSO.getNome(),
                            projeto.getCurso().getNome(),
                            Field.Store.NO,
                            Field.Index.NOT_ANALYZED));

        // Adiciona ao índice o tipo do Projeto
        document.add(new Field(Fields.TIPO.getNome(),
                            projeto.getTipo().toString(),
                            Field.Store.NO,
                            Field.Index.NOT_ANALYZED));

        // Adiciona ao índice o Local da Publicação do Projeto
        document.add(new Field(Fields.LOCAL_PUBLICACAO.getNome(),
                            projeto.getLocalPublicacao(),
                            Field.Store.NO,
                            Field.Index.NOT_ANALYZED));
    }
}

```

```

// Adiciona ao índice os orientadores do Projeto
document.add(new Field(Fields.ORIENTADORES.getNome(),
                      projeto.getOrientadores(),
                      Field.Store.NO,
                      Field.Index.ANALYZED));

document.add(new Field(Fields.TITULO.getNome(),
                      projeto.getTitulo(),
                      Field.Store.NO,
                      Field.Index.ANALYZED));

document.add(new Field(Fields.PALAVRAS_CHAVE.getNome(),
                      projeto.getPalavrasChave(),
                      Field.Store.NO,
                      Field.Index.ANALYZED));

String data = DateTools.dateToString(
                      projeto.getDataPublicacao(),
                      Resolution.YEAR);

document.add(newField(Fields.ANO_PUBLICACAO.getNome(),
                      data,
                      Field.Store.NO,
                      Field.Index.NOT_ANALYZED));

// Neste ponto o PDFBox entra em cena
String texto = this.extrairTexto("caminho");

document.add(new Field(Fields.TEXTO.getNome(),
                      texto,
                      Field.Store.NO,
                      Field.Index.ANALYZED));

return document;
}

```

Conforme se pode observar, todo o texto é extraído pelo método **extrairTexto()**, logo em seguida é indexado e adicionado ao objeto **document**. Outro ponto importante a ser discutido, são os nomes dos campos que referenciarão as informações posteriormente na busca. Estes nomes pertencem a uma **Enumeration** chamada **Fields**, desenvolvida pelos próprios autores e que pode ser encontrada no código fonte da aplicação.

Assim que o PDF é transformado e as informações são indexadas, o PDFBox sai de cena e o processo de recuperação pode então continuar, de acordo com o tutorial do Apache Lucene.