UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ELICITAR: PROTÓTIPO DE GERADOR DE CÓDIGO A PARTIR DE ESPECIFICAÇÕES COM PADRÕES DE REQUISITOS

LEANDRO VILSON BATTISTI

BLUMENAU
2014

2014/1-11

LEANDRO VILSON BATTISTI

ELICITAR: PROTÓTIPO DE GERADOR DE CÓDIGO A PARTIR DE ESPECIFICAÇÕES COM PADRÕES DE REQUISITOS

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

Profa. Joyce Martins, Mestre - Orientadora

BLUMENAU
2014

2014/1-11

ELICITAR: PROTÓTIPO DE GERADOR DE CÓDIGO A PARTIR DE ESPECIFICAÇÕES COM PADRÕES DE REQUISITOS

Por

LEANDRO VILSON BATTISTI

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

Presidente:	Profa. Joyce Martins, Mestre - Orientadora, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Jacques Robert Heckman, Mestre – FURB

Blumenau, 11 de julho de 2014

Dedico este trabalho a minha namorada, irmãos e amigos, mas, principalmente, aos meus pais que são os meus principais motivadores.

AGRADECIMENTOS

Aos meus pais, Vilson e Marilena, por nunca terem deixado de incentivar meus estudos, por eu ser a pessoa que sou hoje.

A minha namorada, Gielez, que me apoiou no desenvolvimento deste trabalho, pela compreensão dos finais de semanas dedicados a ele.

A minha orientadora, Joyce, que teve papel fundamental, por ajudar a organizar minhas ideias para o desenvolvimento deste.

Aos colegas da “Bancada das lamentações” por toda experiência trocada durante o desenvolvimento deste trabalho.

E, principalmente, aos que não acreditaram na ideia deste trabalho, por tornarem o desafio de torná-lo possível ainda mais interessante.

Dinheiro e beleza com o tempo se desgastam, ao final, o que nos resta é nossa história e sabedoria.

Leandro Vilson Battisti

RESUMO

Este trabalho apresenta a especificação e a implementação de um protótipo de ferramenta para realizar geração de código fonte para interfaces a partir de requisitos escritos em língua portuguesa. O processo divide-se em quatro etapas: cadastrar os requisitos conforme padrões de requisitos, utilizar processamento de linguagem natural para identificar componentes de interfaces, gerar arquivo de definição e gerar código a partir desse arquivo de definição. Na primeira etapa são utilizados os padrões de requisitos tipo de dados, estrutura de dados e entidade ativa, propostos por Withall (2007). Na etapa seguinte é usado o analisador morfológico do CoGrOO. A próxima etapa gera arquivos JSON ou XML que descrevem um formulário com base nas informações obtidas na etapa anterior contendo os componentes de interface identificados (`edit`, `memo` ou `checkbox`). Por último, formulários HTML são gerados usando templates Velocity.

Palavras-chave: Padrões de requisitos. Processamento de linguagem natural. Geração de código.

ABSTRACT

This paper presents the specification and implementation of a prototype tool for source code generation for user interfaces from requirements written in Portuguese. The process is divided into four steps: registering the requirements as requirements patterns, use natural language process to identify components interfaces, generate the definition file and generate the source code from this definition file. In the first stage are used requirements patterns data type, data structure and active entity, proposed by withall (2007). The next step is uses the morphological analyzer CoGrOO. The next step generates JSON or XML files that describe a form based on information obtained in the previous step containing identified components interface (edit, checkbox or memo). Finally, HTML forms are generated using Velocity templates.

Keywords: Standards requirements. Natural language processing. Code generation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

AWS – *Amazon Web Services*

CCSL – Centro de Competência em Software Livre

CoGrOO – *Corretor Gramatical Open Office*

CRUD – *Create, Read, Update, Delete*

CPF – Cadastro de Pessoas Físicas

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

JDK - *Java Development Kit*

JPA – Java *Persistent* API

JSF – Java *Server Faces*

JSON – JavaScript *Object Notation*

LGPL – GNU *Lesser General Public License*

MDA – *Model Driven Architecture*

PLN – Processamento de Linguagem Natural

RF – Requisito Funcional

RNF – Requisito Não Funcional

SBVR – *Semantics of Business Vocabulary and Rules*

SQL – *Structured Query Language*

UML – *Unified Modeling Language*

USP – Universidade de São Paulo

URL – *Uniform Resource Locator*

XML – *eXtensive Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Ambler (2003, p. 21), o objetivo principal do desenvolvimento de software é construir sistemas de maneira mais eficaz e eficiente possível e que satisfaça a necessidade do cliente. Para atender este objetivo, podem ser usados vários modelos de processo de software, entre eles a prototipação. Esta abordagem permite evidenciar de forma mais rápida como ficará o software antes de entregá-lo ao usuário. No entanto, este e outros processos de software necessitam que requisitos sejam levantados junto ao usuário através da engenharia de requisitos.

Sommerville (2007, p. 97) diz que a principal preocupação na engenharia de requisitos "é criar e manter documentos de requisitos de [um] sistema". Dessa forma, uma vez identificados e negociados, os requisitos devem ser documentados em um nível apropriado de detalhes para que possam servir de base para o restante do processo de desenvolvimento. Em geral, é produzido um documento de especificação de requisitos, normalmente utilizando linguagem natural, de forma que todos os *stakeholders*[1] possam entendê-lo.

Após o levantamento e a documentação dos requisitos, cria-se um protótipo do sistema que pode ser apresentado aos *stakeholders* a fim de validar se o entendimento do engenheiro de software está de acordo com suas expectativas. Como o requisito é escrito em linguagem natural e o responsável por desenvolver o protótipo não necessariamente é o engenheiro de software, podem ocorrer pequenas diferenças entre o que está escrito e o protótipo apresentado. Tendo este cenário em mente, se o protótipo não atender às necessidades do usuário, o processo terá que recomeçar no levantamento ou na documentação dos requisitos para que os ajustes necessários, tanto nos requisitos quanto no protótipo, sejam feitos.

Para resolver o problema podem ser usados padrões de requisitos[2] que, de acordo com Marques (2008, p. 20), "vêm sendo uma solução a ser adotada para agregar maior confiabilidade a [sic] etapa do levantamento de requisitos". Acredita-se também que se partes do protótipo, como as interfaces gráficas, tiverem o código correspondente gerado automaticamente durante a fase de levantamento de requisitos junto aos *stakeholders*, haverá uma proximidade maior entre o que o cliente espera e o que o software atende. Conforme afirma Herrington (2003, p. xvii), geradores de código não apenas eliminam o trabalho

[1] "O termo *stakeholder* é usado para se referir a qualquer pessoa ou grupo afetado pelo sistema direta ou indiretamente. Os *stakeholders* incluem usuários finais que interagem com o sistema e todo o pessoal na organização que possa ser afetado por sua instalação". (SOMMERVILLE, 2007, p. 98).

[2] "Um padrão de requisito funciona como um guia para escrever um tipo particular de requisito. O padrão explica como descrever determinado tipo de requisito, como expressá-lo e revela possíveis requisitos adicionais implícitos" (MARQUES, 2008, p. 20).

pesado, mas proveem benefícios para o ciclo de vida da engenharia de software baseando-se em produtividade, qualidade, consistência e abstração.

Com base nestes fatos, foi desenvolvido um gerador de código para interfaces gráficas, baseado nas especificações de requisitos feitas utilizando padrões de requisitos de informação, pois, segundo Marques (2008, p. 21), focam "aspectos como a definição de [como] um item de informação deve ser apresentado ou representado". A ferramenta proposta tem como entrada requisitos escritos em língua portuguesa e como saída código para a interface correspondente.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo para gerar código para interfaces gráficas de softwares a partir de especificações escritas utilizando padrões de requisitos.

Os objetivos específicos do trabalho são:

a) processar requisitos baseados em padrões de requisitos de informação;
b) processar requisitos escritos em língua portuguesa com vocabulário irrestrito;
c) processar componentes visuais mais comumente usados na construção de interfaces gráficas;
d) gerar arquivo de definição JSON[3] correspondente à interface gráfica descrita;
e) gerar código fonte a partir do arquivo de definição JSON em pelo menos uma linguagem de programação para validar a ferramenta.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. O próximo capítulo apresenta os aspectos teóricos estudados para o desenvolvimento do trabalho. São relatados temas como geração de código, padrões de requisitos, processamento de linguagem natural. Também são relacionados alguns trabalhos correlatos. O capítulo 3 aborda o desenvolvimento propriamente dito, detalhando os requisitos, a especificação e a implementação, além de trazer resultados e discussões. Por fim, no capítulo 4 são apresentadas as conclusões, bem como sugestões para possíveis extensões.

[3] JavaScript *Object Notation* (JSON) é um formato de intercâmbio de dados, baseado em um subconjunto de JavaScript, que possui similaridades com C, C++, C#, Java, JavaScript, entre outras (CROCKFORD, 2006, p. 1).

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo foi dividido de acordo com os conteúdos estudados para o desenvolvimento do protótipo. A seção 2.1 fala sobre geração automática de código. A seção seguinte apresenta a engenharia de requisitos. A seção 2.3 descreve padrões de requisitos. A seção 2.4 trata do processamento de linguagem natural. A seção 2.5 descreve brevemente o JSON. Por último, na seção 2.6 são descritos os trabalhos correlatos.

2.1 GERAÇÃO AUTOMÁTICA DE CÓDIGO

O termo geração automática de código refere-se a escrever programas que escrevem programas (HERRINGTON, 2003, p. 3). Segundo Silveira (2006, p. 16), "a geração de código é uma técnica de construção de código que utiliza determinadas ferramentas para gerar programas". Estas ferramentas podem variar de *scripts* de ajuda a ferramentas que transformam modelos abstratos de lógica de negócio em aplicações completas.

Para Herrington (2003, p. xvii), geradores automáticos de código podem ser usados com o objetivo de gerar a infraestrutura do código, deixando para os engenheiros de software desafios de programação na solução dos problemas. Ainda para ele, a geração automática de código além de eliminar o trabalho pesado, provê benefícios para a engenharia de software, tais como: qualidade, consistência do código, desenvolvimento ágil de software e flexibilidade.

Herrington (2003, p. 28) afirma que existem duas classes de geradores de código: passivos e ativos. Geradores passivos geram conjuntos de códigos e não mantém responsabilidade sobre estes códigos, deixando que os desenvolvedores possam alterar livremente. Os assistentes de geração das *Integrated Development Enviroment* (IDE) são exemplos de geradores passivos. Geradores ativos de código são responsáveis pelos códigos que geram a longo prazo. Isso significa que quando mudanças nestes códigos são necessárias, os desenvolvedores devem enviar novos parâmetros para o gerador e executá-lo novamente para obter o novo código.

Existem alguns modelos de geradores de código (ativos e passivos), entre os quais, cita-se (HERRINGTON, 2003, p. 29):

a) *muging*: gera um ou mais arquivos como saída, a partir de padrões encontrados em arquivos de entrada. Um exemplo deste tipo de gerador é o JavaDoc, que busca comentários com documentação (*doc comments*) nos códigos fontes Java e gera um arquivo *HyperText Markup Language* (HTML) como saída;

b) *inline-code expander:* gera um ou mais arquivos como saída, a partir de sintaxes "embarcadas" no código dos arquivos de entrada. No Quadro 1 observa-se um exemplo de um código fonte escrito em linguagem C com sintaxe *Structured Query Language* (SQL) embarcada e o código gerado por um gerador *inline-code expander*;

c) *mixed-code generator*: é similar ao modelo anterior, porém gera o arquivo de saída sobre o arquivo de entrada;

d) *partial-class generator*: diferente dos modelos citados até agora, não utiliza código fonte e sim uma representação abstrata do código a ser criado, sem a utilização de filtros ou a substituição de fragmentos de códigos (HERRINGTON, 2003, p. 87). O código gerado deve ser usado em conjunto com classes derivadas que implementam a regra de negócio, sendo um bom ponto de partida para a codificação em camadas, pois é possível substituir as classes de negócio quando necessário;

e) *tier generator*: gera todo o código fonte de uma camada ou seção de uma aplicação. Neste modelo o gerador tem informações suficientes para construir todo o código fonte, incluindo as classes e suas funções (HERRINGTON, 2003, p. 90).

Quadro 1 – Exemplo de código gerado pelo modelo *inline-code expander*

código fonte	código gerado

`void main(int argc, char *argv[])`
`{`
`  <sql-select: SELECT first,`
`  last`
`  FROM names>`
`  return;`
`}`
	`void main(int argc, char *argv[]) {`
	`  struct {`
	`  char *first;`
	`  char *last;`
	`  } *sql_output_1;`
	`  {`
	`  db_connection *db = get_connection();`
	`  sql_statement *sth = db->prepare("SELECT first, last FROM names");`
	`  sth->execute();`
	`  sql_output_1 =`
	`  malloc(sizeof(*sql_output_1) * sth->count());`
	`  for(long index = 0;`
	`  index < sth->count(); index++) {`
	`  // ... marshal data`
	`  }`
	`  }`
	`  return;`
	`}`

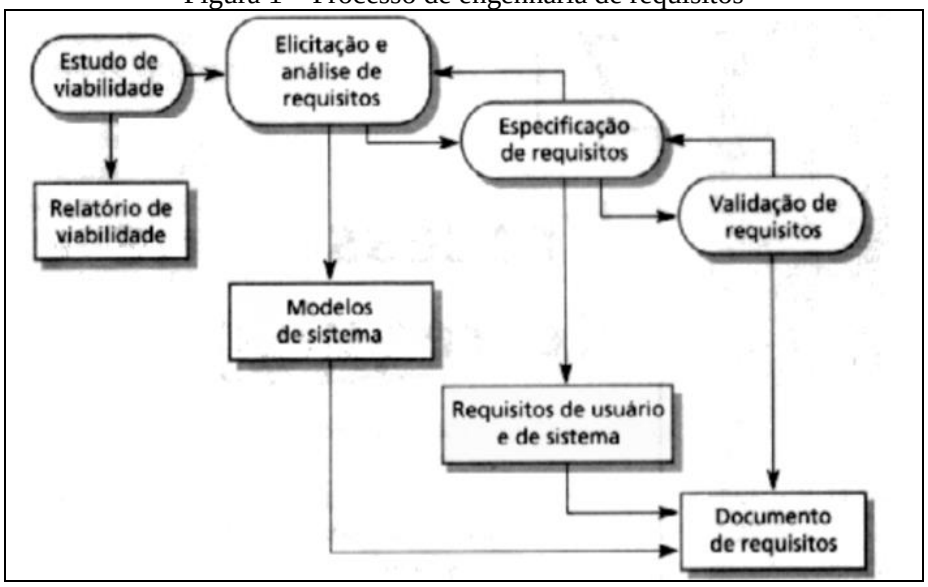
Fonte: Herrington (2003, p. 78).

Independente da classe (passivo ou ativo) ou do modelo, no desenvolvimento de um gerador de código deve-se atender as seguintes etapas (HERRINGTON, 2003, p. 18): identificar a saída desejada, definir a entrada e como a mesma será analisada, interpretar e recuperar as informações da entrada necessárias para gerar a saída e gerar os arquivos de saída a partir da entrada.

2.2 ENGENHARIA DE REQUISITOS

"A Engenharia de Requisitos é o processo pelo qual os requisitos de um produto de software são coletados, analisados, documentados e gerenciados ao longo de todo o ciclo de vida do software" (AURUM; WOHLIN, 2005 apud FALBO, 2012, p. 1). Para Pressman (2006, p. 116), a engenharia de requisitos serve para engenheiros de softwares compreenderem melhor o problema que devem resolver, incluindo o que o cliente quer e como que os usuários interagirão com o software. Ainda segundo ele, construir software elegante e que não resolva algo não serve para ninguém. Por isso, é importante entender o que precisa ser feito antes de construir um sistema, fornecendo por escrito a todos os envolvidos o entendimento do problema. Sommerville (2007, p. 95) divide a engenharia de requisitos em quatro subprocessos de alto-nível (Figura 1): estudo de viabilidade, obtenção (elicitação e análise), especificação e validação de requisitos.

Figura 1 – Processo de engenharia de requisitos



Fonte: Sommerville (2007, p. 96).

Em função da proposta deste trabalho, são descritos os subprocessos de especificação e validação de requisitos. Pressmann (2006, p. 129) afirma que especificação de requisitos é um documento com a descrição detalhada dos aspectos do software a ser construído antes do projeto começar. A especificação de requisitos, segundo Sommerville (2007, p. 99), é dividida em atividades, sendo elas: obtenção dos requisitos, que serve para coletar informações junto aos *stakeholders*; clarificação e organização, que agrupa estas informações em conjunto de requisitos relacionados; priorização e negociação, que definem os requisitos mais importantes; e por último documentação de requisitos. Na documentação de requisitos são produzidos documentos de requisitos (informais ou formais).

Sommerville (2007, p. 79) diz que requisitos de um sistema são descrições, em linguagem natural, dos serviços e das restrições fornecidos pelo sistema e devem refletir a necessidade do cliente. Os requisitos podem ser divididos em duas categorias: funcionais e não funcionais. Os Requisitos Funcionais (RF) descrevem o que o sistema deve fazer, enquanto os Requisitos Não Funcionais (RNF) são restrições sobre funções oferecidas pelo sistema. No Quadro 2 tem-se exemplo de um requisito funcional e um não funcional.

Quadro 2 – Exemplo de requisito funcional e não funcional

tipo de requisito	descrição
RF	O sistema deve fornecer telas apropriadas para o usuário ler os documentos.
RNF	A tela deve ser implementada como simples HTML

Fonte: adaptado de Sommerville (2007, p. 80).

Sabe-se, porém, que, de acordo com Decarle e Grahl (2008, p. 1), uma das maiores dificuldades no desenvolvimento está no levantamento de requisitos por problemas como falhas de comunicação e de documentação. No exemplo do Quadro 2 pode-se questionar o requisito funcional com relação a quais arquivos são permitidos, que, se não especificados, podem ocasionar falhas.

Estas falhas podem ser detectadas no subprocesso de validação, que, segundo Sommerville (2007, p. 105), dedica-se a mostrar que os requisitos que definem o sistema realmente atendem o desejado pelo usuário. Ainda segundo o autor, existem várias técnicas que podem ser utilizadas para validar um requisito, tais como revisão de requisito, prototipação e geração de casos de teste. Para Sommerville (2007, p. 106), mesmo com estas técnicas, ainda existem falhas na validação de requisitos por ser difícil imaginar todos os cenários de um software em funcionamento mesmo para um profissional habilidoso. Como resultado disto, raramente todos os problemas de requisitos são encontrados. Uma técnica que pode ser utilizada para dar maior confiabilidade é adotar padrões de requisitos (DECARLE; GRAHL, 2008, p. 20).

2.3 PADRÕES DE REQUISITOS

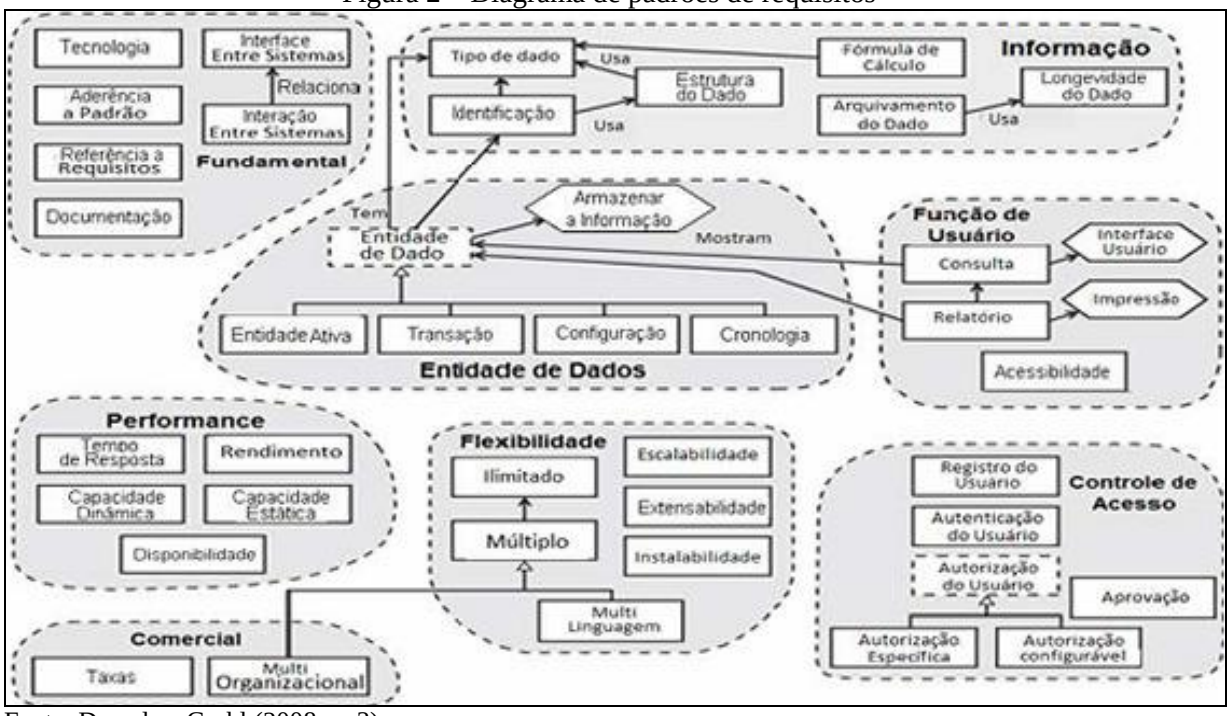
Withall (2007, p. 19) escreve que todo sistema por mais trivial que seja tem requisitos que têm similaridades ou que ocorrem na maioria dos sistemas. Um exemplo é o número do Cadastro de Pessoas Físicas (CPF) que é apresentado em alguns casos de uso, porém sempre é o mesmo requisito.

Sendo assim, um padrão de requisito é um guia para escrever um tipo de requisito, sendo que seu objetivo é que requisitos sejam escritos com mais qualidade, de forma mais rápida e com menos esforço. Um padrão é utilizado para descrever apenas um único requisito,

pois contém informações detalhadas e específicas para esse requisito (WITHALL, 2007, p. 3). Um padrão descreve soluções para problemas recorrentes. Assim sendo, não há necessidade de estudar o mesmo problema novamente para identificar uma solução, produzindo assim um catálogo com soluções bem estruturadas, extensíveis e reutilizáveis de padrões de requisitos (TAGLIATI; JOHNSON; ROUSSOS, 2007, p. 4).

Withall (2007, p. 5) apresenta a classificação dos padrões de requisitos em oito domínios (Figura 2): comercial, controle de acesso, entidade de dados, flexibilidade, função de usuário, fundamental, informação e performance. Estes grupos agregam 37 tipos de padrões de requisitos.

Figura 2 – Diagrama de padrões de requisitos



Fonte: Decarle e Grahl (2008, p. 3).

Neste trabalho são utilizados os padrões de requisitos: tipo de dados, estrutura de dados e entidade ativa. Tipo de dados e estrutura de dados ficam no domínio Informação.

> O domínio informação foca aspectos como a definição de um item de informação deve ser apresentado ou representado, como será a identificação única das entidades de dados, como será composto um determinado item de informação, como calcular determinado tipo de valor ou determiná-lo através de alguns passos lógicos, especificar a movimentação ou cópia de dados de um local para outro e por quanto tempo um certo tipo de dado deve ser mantido ou por quanto tempo deve estar disponível. (MARQUES, 2008, p. 21).

Entidade ativa pertence ao domínio Entidade de Dados. De acordo com Withall (2007, p. 33), o domínio Entidade de Dados não é um padrão em si, mas sim um padrão de agrupamento de características, sendo que qualquer descrição feita é aplicada como comum

aos outros quatro tipos de padrões pertencentes a este domínio, que são: entidade ativa, transação, configuração e cronológico.

> A entidade de dados é o domínio responsável por definir um tipo de entidade, sua vida útil, para qual as informações são armazenadas, definir eventos que ocorrerão ao iniciar uma transação, definir configurações do sistema e como registrar determinados eventos do sistema. (MARQUES, 2008, p. 21).

2.3.1 Tipo de dados

Este padrão de requisitos define como um tipo simples é representado e apresentado, como por exemplo, pode-se descrever o formato padrão de uma data. Withall (2007, p. 86) descreve que não é necessário escrever tipo de dados para dados que são bem conhecidos e simples como um tipo de produto ou a identificação de uma empresa, porém identifica isso como uma boa prática. Isto porque, segundo ele, isto promove a consistência do sistema, promove a reusabilidade e também previne que programadores definam por conta própria de forma arbitrária estas informações.

A definição de um tipo de dados precisa ser feita com informações que clientes possam entender, por exemplo, descrevendo informações como alfanumérico e não como `varchar` ou `string`, entre outros. Caso haja dois requisitos que tenham o mesmo formato, porém sejam aplicados a regras diferentes, eles devem ser escritos como dois tipos de dados distintos. Além disso, nunca deve ser utilizado neste padrão de requisito termos técnicos de bancos de dados ou de linguagens de programação, a não ser que alguma circunstância force isto. No Quadro 3 são apresentados exemplos do padrão de requisitos do tipo de dados.

Quadro 3 – Padrão de requisitos: tipo de dados

nome	definição
nome	Deve ter no máximo 50 caracteres.
data de nascimento	Deve ser maior que 31/12/1950.
sexo	Será representado por uma letra: “F” ou “M”.

O padrão de requisitos tipo de dados é usado neste trabalho em função de sua descrição ser próxima a atributos de classes.

2.3.2 Estrutura de dados

Estrutura de dados é um padrão de requisitos que define uma composição formada por múltiplas informações. Um requisito para uma estrutura de dados é utilizado para proteger repetições. Este padrão não contém funções ou armazena dados, sendo formado por outros requisitos. Uma estrutura também não serve para definir se uma informação é obrigatória, pois é uma informação que pode ser utilizada em vários lugares sob diferentes regras (WITHALL, 2007, p. 95).

Uma estrutura de dados precisa conter, no mínimo, um nome e uma lista de informações. A lista de informações pode ser um tipo de dados previamente definido, uma descrição de um tipo de dados ou outra estrutura de dados, sendo que a sequência com que eles aparecem não tem importância. No Quadro 4 têm-se exemplos de estrutura de dados.

Quadro 4 – Padrão de requisitos: estrutura de dados

nome	definição
nome pessoal	Os detalhes do nome de uma pessoa são formados pelos seguintes itens de informação: • primeiro nome; • nome do meio; • sobrenome; • iniciais; • título.
contato pessoal	Os detalhes do contato pessoal de uma pessoa são formados pelos seguintes itens de informação: • nome pessoal (definido anteriormente); • endereço; • telefone comercial; • telefone residencial; • celular; • fax; • *pager*; • e-mail.

Fonte: adaptado de Withall (2007, p. 96).

Withall (2007, p. 96) ainda faz uma analogia com o desenvolvimento do código fonte, informando que uma estrutura de dados sugere uma classe e uma tabela no banco de dados. Este padrão de requisito é usado neste trabalho por causa da reusabilidade dos tipos de dados.

2.3.3 Entidade ativa

Withall (2007, p. 129) define o padrão de requisitos entidade ativa como uma entidade que tem um tempo de vida com informações que podem ser armazenadas. Assim sendo, uma entidade ativa é primeiramente definida estabelecendo quais informações precisam ser armazenadas campo a campo e como a entidade pode ser unicamente identificada. O arquivamento das informações pode ser implícito ou explícito. Feito isto, podem ser definidas funções de inserção, alteração, exclusão e visualização da entidade ativa. Caso estas funções tenham particularidades, devem ser escritas em requisitos separados (WITHALL, 2007, p. 129).

Para Withall (2007, p. 130), este padrão de requisito deve ser formado, no mínimo, por: um nome, uma descrição, informações componentes, uma identificação única e uma entidade “pai”, caso a entidade ativa só possa existir se outra já tiver sido criada. O Quadro 5

apresenta um exemplo de entidade ativa.

Quadro 5 – Padrão de requisitos: entidade ativa

nome	definição
cliente	O sistema deve armazenar as seguintes informações sobre o cliente: • identificação de cliente; • senha; • contato pessoal (definido anteriormente); • cartão de crédito; • data de nascimento; • data de registro; • estado (ativo, bloqueado ou terminado), nunca é apresentado para o cliente. Cada cliente é unicamente definido pela identificação do cliente.

Fonte: adaptado de Withall (2007, p. 130).

O padrão de requisito entidade ativa foi escolhido para ser utilizado neste trabalho por suas características serem próximas a registros feitos em telas *Create*, *Read*, *Update*, *Delete* (CRUD), ou seja, telas simples de cadastro.

2.4 PROCESSAMENTO DE LINGUAGEM NATURAL

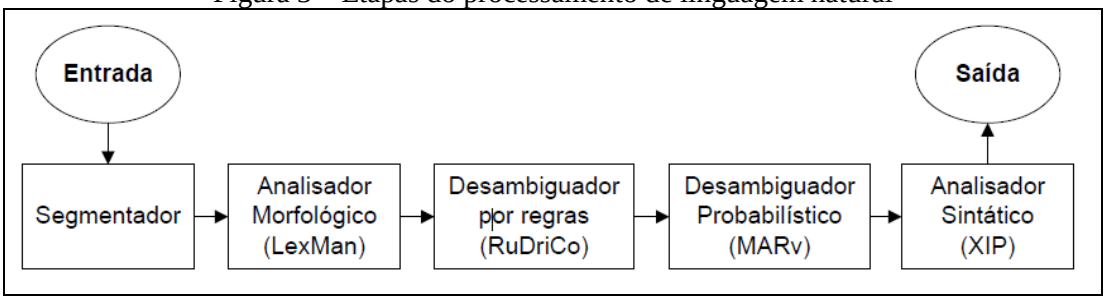
Conforme Vicente (2013, p. 17), Processamento de Linguagem Natural (PLN) tem como um de seus objetivos melhorar a interação entre humanos e máquinas de forma mais natural possível utilizando a linguagem natural. Bhagat et al. (2012, p. 125) explicam que sistemas de PLN usam diferentes níveis de análise linguística, sendo elas: fonética, referente à análise dos sons; léxica, diz respeito à formação das palavras; sintática, analisa a constituição das frases; semântica, relacionada ao significado não só de cada palavra, mas também do conjunto resultante delas; pragmática, referente à interpretação das frases; e morfológica.

> O analisador morfológico identifica palavras ou expressões isoladas em uma sentença, sendo este processo auxiliado por delimitadores (pontuação e espaços em branco). As palavras identificadas são classificadas de acordo com seu tipo de uso ou, em linguagem natural, categoria gramatical. (OLIVEIRA, 2002 apud OLIVEIRA NETO; TONIN; PIETRICH, 2010).

Ainda segundo os autores, o emprego do analisador morfológico é necessário, pois ele é essencial para identificar a formação de uma frase.

O PLN, de acordo com Vicente (2013, p. 2), divide-se em várias etapas, conforme observado na Figura 3.

Figura 3 – Etapas do processamento de linguagem natural



Fonte: Vicente (2013, p. 2).

Neste trabalho são utilizadas as etapas de segmentação e análise morfológica para extrair os *tokens* que representam palavras de determinadas classes. Vicente (2013, p. 1) descreve a segmentação de texto como um processo que transforma uma ou mais palavras em um segmento. Estes segmentos são etiquetados em categorias. Na língua portuguesa existem várias categorias, como verbo, substantivo e adjetivo, sendo que algumas palavras podem pertencer a mais de uma categoria. Por exemplo, nas frases “A casa é grande.” e “João casa no sábado.”, a palavra casa é o substantivo casa e o verbo casar, respectivamente. Desta maneira, o segmentador passará o segmento “casa” para o analisador morfológico.

> A análise morfológica consiste no processo de, dada qualquer palavra da língua, determinar todas as suas características morfológicas. Inversamente, a geração morfológica consiste em, dado um conjunto de características morfológicas, construir a palavra que as possui. (MEDEIROS, 1995, p. 17).

Conforme Vicente (2013, p. 1), o analisador morfológico, com a ajuda de um dicionário, etiquetará os segmentos com todas as categorias possíveis e posteriormente fará o processamento de desambiguação para obter a categoria correta. Para Medeiros (1995, p. 20), os dicionários eletrônicos e léxicos computacionais são diferentes. Enquanto dicionários eletrônicos são implementações digitais dos tradicionais dicionários de uso geral, os léxicos computacionais têm o objetivo de dar suporte a uma determinada aplicação específica de linguagem natural, sendo que estes léxicos computacionais podem ser construídos diretamente a partir de dicionários eletrônicos e utilizados no PLN. Neste trabalho foi utilizado um conjunto de léxicos computacionais para extração de características a partir dos requisitos que são utilizadas para mapear quais componentes devem ser gerados.

2.5 JSON

JSON é baseado em um subconjunto de JavaScript, sendo completamente independente de linguagem de programação, mas possuindo similaridades com C, C++, C#, Java, JavaScript, entre outras. Define regras de formatação para representação de dados estruturados (CROCKFORD, 2006, p. 1). Os dados são representados com informações de

chave e valor, podendo ser aninhados. No Quadro 6 é possível visualizar um exemplo de interface descrita em formato JSON, a partir dos requisitos especificados no Quadro 3.

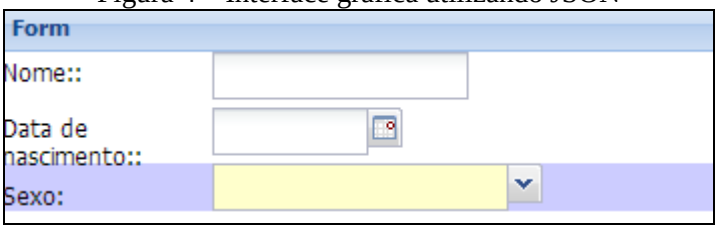
Quadro 6 – Exemplo de interface em JSON

```
{  xtype:"form", title:"Form", items:[ {    xtype:"textfield",
                                            fieldLabel:"Nome:",
                                            name:"textvalue"
                                        },{ xtype:"datefield",
                                            fieldLabel:"Data de nascimento:",
                                            name:"datevalue"
                                        },{ xtype:"combo",
                                            fieldLabel:"Sexo",
                                            name:"combovalue",
                                            hiddenName:"combovalue",

                                        } ]
}
```

A partir dos metadados do quadro anterior, pode-se, por exemplo, gerar a interface da Figura 4.

Figura 4 – Interface gráfica utilizando JSON



2.6 TRABALHOS CORRELATOS

Nas pesquisas realizadas não foram encontradas ferramentas ou trabalhos científicos que tratam da geração automática de código a partir de padrões de requisitos. Porém, existem ferramentas que geram diagramas da *Unified Modeling Language* (UML) baseadas em linguagem natural, assim como existem diversos trabalhos envolvendo geração automática de código. São descritos os seguintes trabalhos: Paradigma (SILVA; MARTINS, 2008), a ferramenta para geração de classes a partir do PLN (BHAGAT et al., 2012) e Delphi2Java-II (SILVEIRA, 2006).

2.6.1 Paradigma

Silva e Martins (2008) apresentam uma ferramenta para geração de classes em UML, baseada em processamento de linguagem natural.

Segundo Silva e Martins (2008, p. 142), a ferramenta etiqueta o texto de entrada (especificação de requisitos ou cenários de casos de uso) com marcações que identificam palavras em categorias morfossintáticas. Em seguida são aplicados padrões linguísticos, classificando cada palavra como classe, atributo ou operação. Também são identificadas as associações entre as classes e suas multiplicidades. Por último, as classes são geradas de

forma automática. O usuário deve “interagir no processo automatizado para vincular os atributos e operações às respectivas classes” (SILVA; MARTINS, 2008, p. 142).

2.6.2 Ferramenta para geração de classes a partir de PLN

No estudo de Bhagat et al. (2012) são apresentadas as etapas para a extração de informações escritas em linguagem natural para a geração de diagramas de classe. A regra básica para a extração é analisar o texto em linguagem natural e extrair elementos de modelagem orientada a objetos. Os passos são (BHAGAT et al., 2012, p. 126):

a) processar os requisitos do sistema, extraindo os *tokens*;
b) fazer a análise morfológica das palavras;
c) efetuar a análise sintática validando frases de acordo com as regra gramaticais da língua inglesa, para identificar objetos, sujeitos, ações, atributos, entre outros;
d) realizar a análise semântica para identificar as associações, os métodos e os atributos de cada objeto, sendo substantivos e sujeitos identificados como objetos, verbos como métodos e adjetivos como atributos dos objetos;
e) criar associações e relações entre as classes e os objetos extraídos a partir das preposições existentes;
f) gerar um modelo lógico do diagrama de classes com base nas informações extraídas nos passos anteriores;
g) desenhar o diagrama de classes a partir do modelo lógico;
h) gerar código em VB.NET ou Java a partir do modelo lógico.

2.6.3 Delphi2Java-II

Silveira (2006) apresenta uma ferramenta para realizar a conversão de formulários elaborados na linguagem de programação Delphi para aplicações na linguagem de programação Java. Foram seguidas as seguintes etapas para desenvolver a ferramenta (SILVEIRA, 2006, p. 19):

a) identificar a saída desejada: interfaces gráficas Java desenvolvidas em *Swing*;
b) definir a entrada: formulários Delphi, um arquivo com extensão `dfm`;
c) interpretar e recuperar as informações da entrada, definindo a formatação e a geração de saída;
d) gerar a saída a partir das informações extraídas do arquivo de entrada.

No processamento dos formulários Delphi são utilizados analisadores léxico, sintático e semântico. Na geração da saída das classes Java é utilizado o motor de *templates* Velocity.

3 DESENVOLVIMENTO

Este capítulo apresenta as etapas do desenvolvimento do protótipo e sua utilização, sendo elas:

a) requisitos funcionais e requisitos não funcionais escritos usando padrões de requisitos;
b) especificação do protótipo com diagramas da UML;
c) implementação, incluindo as técnicas e ferramentas utilizadas, o processamento para identificação das ligações entre os diferentes requisitos, a geração dos arquivos de definição de componentes e a operacionalidade do protótipo.

Os resultados obtidos também são descritos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Quadros 7 a 23 apresentam os principais requisitos funcionais e não funcionais escritos utilizando padrões de requisitos, conforme modelo proposto por Withall (2007). Primeiro são apresentados os RNFs e posteriormente os RFs.

Quadro 7 – RNF001: Utilização de padrões de requisitos

Nome	Utilização de padrões de requisitos.
Tipo do requisito	Aderência a padrão.
Objetivo	Delimitar os padrões de requisitos do protótipo conforme um modelo específico.
Descrição	O protótipo deve ser baseado nos padrões de requisitos tipo de dados, estrutura de dados e entidade ativa, propostos por Withall (2007).

Quadro 8 – RNF002: Interface para usuário

Nome	Interface para usuário.
Tipo do requisito	Tecnologia.
Objetivo	Determinar a tecnologia de acesso do usuário ao protótipo.
Descrição	O protótipo deve fornecer todas as funções de usuário em uma interface 100% web acessível a partir dos navegadores Google Chrome versão 35.0.1916.153m e Mozilla Firefox versão 4.

Quadro 9 – RNF003: Linguagem de programação para desenvolvimento

Nome	Linguagem de programação para desenvolvimento.
Tipo do requisito	Tecnologia.
Objetivo	Determinar a linguagem de programação que será utilizada no desenvolvimento do protótipo.
Descrição	O protótipo deve ser desenvolvido utilizando Java 1.6. para ser compatível com a versão JDK da Amazon Web Service.

Quadro 10 – RNF004: Formato do arquivo de saída

Nome	Formato do arquivo de saída.
Tipo do requisito	Tecnologia.
Objetivo	Determinar o formato do arquivo de saída gerado pelo protótipo.
Descrição	O protótipo deve gerar como saída um arquivo do tipo JSON ou *eXtensive Markup Language* (XML).

Quadro 11 – RNF005: Documentação de ajuda

Nome	Documentação de ajuda.
Tipo do requisito	Documentação.
Objetivo	Determinar a documentação de ajuda ao usuário.
Descrição	O protótipo deve fornecer uma documentação das funções disponíveis para o usuário.

Quadro 12 – RF001: Tipo do requisito

Nome	Tipo do requisito.
Tipo do requisito	Tipo de dado.
Objetivo	Descrever o formato do tipo do requisito.
Descrição	O tipo do requisito deve ser um campo alfanumérico de até 100 caracteres.

Quadro 13 – RF002: Nome do requisito

Nome	Nome do requisito.
Tipo do requisito	Tipo de dado.
Objetivo	Descrever o formato do nome do requisito.
Descrição	O nome do requisito deve ser um campo de alfanumérico de até 255 caracteres.

Quadro 14 – RF003: Objetivo do requisito

Nome	Objetivo do requisito.
Tipo do requisito	Tipo de dado.
Objetivo	Descrever o formato do objetivo do requisito.
Descrição	O objetivo do requisito deve ser um campo alfanumérico sem limite de caracteres.

Quadro 15 – RF004: Descrição do requisito

Nome	Descrição do requisito.
Tipo do requisito	Tipo de dado.
Objetivo	Descrever o formato das informações referentes a um requisito.
Descrição	A descrição do requisito deve ser um campo de alfanumérico sem limite de caracteres.

Quadro 16 – RF005: Estrutura de um requisito

Nome	Estrutura de um requisito.
Tipo do requisito	Estrutura de dado.
Objetivo	Apresentar as informações básicas de um requisito.
Descrição	Um requisito deverá ter os seguintes itens de informações: tipo, nome, objetivo e descrição.

Quadro 17 – RF006: Identificador do requisito

Nome	Identificador do requisito.
Tipo do requisito	Identificação.
Objetivo	Garantir um valor único para o requisito.
Descrição	Cada requisito deve ter um identificador numérico único gerado pelo protótipo.

Quadro 18 – RF007: Cadastro do requisito

Nome	Cadastro do requisito.
Tipo do requisito	Entidade ativa.
Objetivo	Armazenar as informações sobre requisitos.
Descrição	O protótipo deve armazenar as seguintes informações sobre os requisitos: identificação do requisito e requisito.

Quadro 19 – RF008: Tipo da característica

Nome	Tipo de característica.
Tipo do requisito	Tipo de dado.
Objetivo	Apresentar os tipos de características disponíveis.
Descrição	Os tipos de características são: obrigatório, opcional, multilinha, texto, lógico.

Quadro 20 – RF009: Característica

Nome	Característica.
Tipo do requisito	Tipo de dado.
Objetivo	Descrever o léxico computacional que identifica uma característica, isto é, palavras reconhecidas nas descrições dos requisitos que servem para identificar os componentes de interface.
Descrição	Texto que identifica uma característica na descrição de um requisito.

Quadro 21 – RF010: Cadastro da característica

Nome	Cadastro da característica.
Tipo do requisito	Entidade ativa.
Objetivo	Armazenar as informações sobre características.
Descrição	O protótipo deve armazenar as seguintes informações sobre as características: tipo da característica e característica.

Quadro 22 – RF011: Geração do arquivo de definição

Nome	Geração do arquivo de definição.
Tipo do requisito	Transação.
Objetivo	Descrever a geração de arquivos de definição a partir de um formulário.
Descrição	Deve ser possível gerar um arquivo de definição para um formulário. Tanto o formulário a ser utilizado assim como o tipo do arquivo de definição a ser gerado, que pode ser XML ou JSON, deve ser especificado pelo usuário.

Quadro 23 – RF012: Geração do HTML a partir do arquivo de definição

Nome	Geração do HTML a partir de arquivo de definição.
Tipo do requisito	Transação.
Objetivo	Descrever a geração de HTML a partir de um arquivo de definição.
Descrição	Deve ser possível gerar um arquivo HTML a partir do arquivo de definição previamente gerado pelo protótipo.

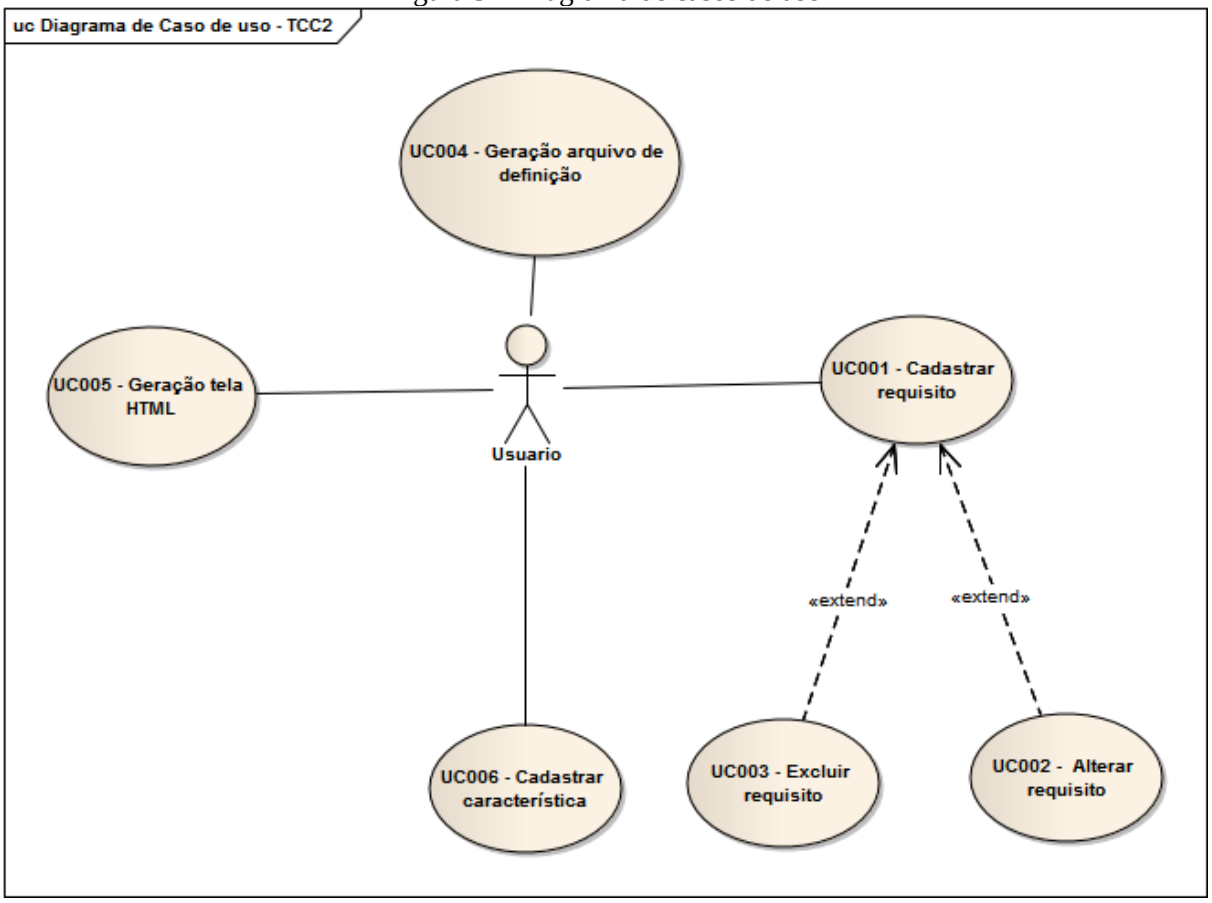
3.2 ESPECIFICAÇÃO

Na especificação foram definidos os seguintes diagramas da UML: de casos de uso, de pacotes, de classes e de sequência. A diagramação foi feita utilizando a ferramenta Enterprise Architect.

3.2.1 Diagrama de casos de uso

No diagrama de casos de uso são apresentadas as ações que o `Usuário` pode executar no protótipo. A Figura 5 apresenta o diagrama de casos de uso desenvolvido.

Figura 5 – Diagrama de casos de uso



A seguir são apresentados os detalhes de cada caso de uso. Os casos de uso UC001 (Quadro 24), UC002 (Quadro 25) e UC003 (Quadro 26) são para manipular as informações dos requisitos.

Quadro 24 – Caso de uso 001: Cadastrar requisito

UC001 – Cadastrar Requisito	
Descrição	Descreve o fluxo de cadastro de um requisito.
Pré-condição	Usuário com acesso à tela de cadastro de requisitos.
Atores	Usuário
Fluxo principal	1. Usuário seleciona um dos tipos de padrão de requisitos existentes. 2. Protótipo preenche o campo descrição com o modelo de texto correspondente, de acordo com o tipo do padrão de requisito selecionado. 3. Usuário preenche o campo nome. 4. Usuário preenche o campo objetivo. 5. Usuário preenche o campo descrição, podendo ou não seguir o modelo. 6. Usuário pressiona o botão salvar. 7. Protótipo apresenta mensagem de cadastro de requisito realizado com sucesso, além dos dados cadastrados.
Pós-condição	Requisito cadastrado com sucesso.
RF associado	RF001, RF002, RF003. RF004, RF005, RF006, RF007.
RNF associado	RNF001, RNF002, RNF003.

Quadro 25 – Caso de uso 002: Alterar requisito

UC002 – Alterar requisito	
Descrição	Descreve o fluxo de alteração de um requisito.
Pré-condição	Usuário com acesso à tela de cadastro de requisitos. Deve haver pelo menos um requisito cadastrado.
Atores	Usuário
Fluxo principal	1. Usuário seleciona, entre os requisitos cadastrados, o requisito desejado. 2. Usuário pressiona o botão para alterar o requisito. 3. Protótipo carrega os dados do requisito nos campos correspondentes. 4. Usuário altera os dados conforme necessário. 5. Usuário pressiona o botão salvar. 6. Protótipo apresenta mensagem de cadastro de requisito realizado com sucesso, além dos dados cadastrados.
Pós-condição	Requisito alterado com sucesso.
RF associado	RF001, RF002, RF003. RF004, RF005, RF006, RF007.
RNF associado	RNF001, RNF002, RNF003.

Quadro 26 – Caso de uso 003: Excluir requisito

UC003 – Excluir requisito	
Descrição	Descreve o fluxo de exclusão de um requisito.
Pré-condição	Usuário com acesso à tela de cadastro de requisitos. Deve haver pelo menos um requisito cadastrado.
Atores	Usuário
Fluxo principal	1. Usuário seleciona, entre os requisitos cadastrados, o requisito desejado. 2. Usuário pressiona o botão para excluir o requisito. 3. Protótipo apresenta uma mensagem para o usuário confirmar exclusão. 4. Usuário confirma a exclusão (pressiona em sim). 5. Protótipo exclui o registro do requisito do banco de dados, não o apresentando entre os requisitos cadastrados.
Fluxo alternativo	Usuário não confirma a exclusão (pressiona em não). O protótipo mantém o registro do requisito.
Pós-condição	Requisito excluído com sucesso.
RF associado	RF001, RF002, RF003. RF004, RF005, RF006, RF007.
RNF associado	RNF001, RNF002, RNF003.

O caso de uso UC004 (Quadro 27) descreve como o arquivo de definição de um formulário, com os componentes de interface reconhecidos a partir dos textos dos requisitos, é gerado. Um formulário é definido (automaticamente pela ferramenta) a partir dos requisitos entidade ativa. São considerados formulários todas as entidades ativas cadastradas. O caso de uso U005 (Quadro 28) descreve como gerar um formulário HTML a partir do arquivo de definição.

Quadro 27 – Caso de uso 004: Geração arquivo de definição

UC004 – Gerar arquivo de definição	
Descrição	Descreve o fluxo de geração dos arquivos de definição de formulários.
Pré-condição	Usuário com acesso à tela de geração de arquivo de definição. Deve haver pelo menos um requisito entidade ativa cadastrado.
Atores	Usuário
Fluxo principal	1. Usuário seleciona o formulário desejado. 2. Usuário pressiona o botão para gerar XML ou o botão para gerar JSON. 3. Protótipo apresentada a tela para salvar o arquivo correspondente. 4. Usuário seleciona o local onde deseja salvar o arquivo de definição. 5. Usuário pressiona o botão salvar. 6. Protótipo gera o arquivo de definição correspondente.
Fluxo alternativo	Usuário pressiona o botão cancelar. O protótipo não gera arquivo de definição.
Fluxo de exceção	Usuário pressiona os botões para gerar arquivo de definição sem ter selecionado um formulário. O protótipo apresenta mensagem de seleção obrigatória.
Pós-condição	Arquivo de definição gerado com sucesso.
RF associado	RF009
RNF associado	RNF004, RNF002

Quadro 28 – Caso de uso 005: Geração tela HTML

UC005 – Gerar HTML	
Descrição	Descreve o fluxo de geração de HTML a partir do arquivo de definição.
Pré-condição	Usuário com acesso à tela de geração de arquivo HTML. Ter o arquivo de definição gerado através do UC004.
Atores	Usuário
Fluxo principal	1. Usuário pressiona o botão procurar. 2. Usuário seleciona o arquivo de definição (XML ou JSON) desejado. 3. Protótipo apresenta o nome do arquivo selecionado. 4. Usuário pressiona o botão enviar. 5. Protótipo apresenta uma barra de progresso de envio do arquivo. 6. Protótipo apresenta mensagem informando que arquivo foi enviado com sucesso. 7. Usuário pressiona o botão gerar. 8. Protótipo apresenta a tela para salvar o arquivo HTML. 9. Usuário seleciona o local onde deseja salvar o arquivo. 10. Usuário pressiona o botão salvar. 11. Protótipo gera o arquivo HTML correspondente.
Fluxo alternativo	Usuário pressiona o botão cancelar. O protótipo não gera arquivo HTML.
Fluxo de exceção	Usuário pressiona o botão gerar sem ter selecionado um arquivo ou o arquivo de definição é inválido. O protótipo apresenta mensagem de exceção.
Pós-condição	Arquivo HTML gerado com sucesso.
RF associado	RF013
RNF associado	RNF004, RNF002

O caso de uso UC006 (Quadro 29) descreve o cadastro de características. As características são palavras-chave reconhecidas nas descrições dos requisitos, utilizadas como parte do processo de transformação dos requisitos em componentes de interface de um formulário.

Quadro 29 – Caso de uso 006: Cadastrar característica

UC006 – Cadastrar característica	
Descrição	Descreve o fluxo para o cadastro de características.
Pré-condição	Usuário com acesso à tela de cadastro de características.
Atores	Usuário
Fluxo principal	1. Usuário seleciona um tipo de característica. 2. Usuário preenche o campo característica com informação que representa esta característica. 3. Usuário pressiona o botão salvar. 4. Protótipo cadastra a característica.
Pós-condição	Característica cadastrada com sucesso.
RF associado	RF013, RF010. RF011
RNF associado	RNF002

3.2.2 Diagrama de pacotes

Para facilitar a visualização das classes especificadas foi criado um diagrama de pacotes, apresentado na Figura 6. Neste diagrama são apresentados os seguintes pacotes:

a) `beans`: contém as classes que alimentam as informações da interface web do protótipo;

b) `components`: armazena as classes que definem os componentes de interface suportados pelo protótipo para serem identificados e gerados;

c) `db`: armazena as classes de comunicação com o banco de dados;

d) `exceptions`: armazena as classes de exceção;

e) `mail`: contém a classe responsável por comunicar com o servidor de *e-mails* e enviá-los;

f) `model`: armazena as classes de persistência utilizadas pelo Java *Persistent* API (JPA);

g) `patterns`: armazena as classes que definem os padrões de requisitos para o protótipo, além das classes que fazem o reconhecimento de componentes a partir destes padrões de requisito;

h) `requirements`: armazena classes que definem as estruturas dos requisitos;

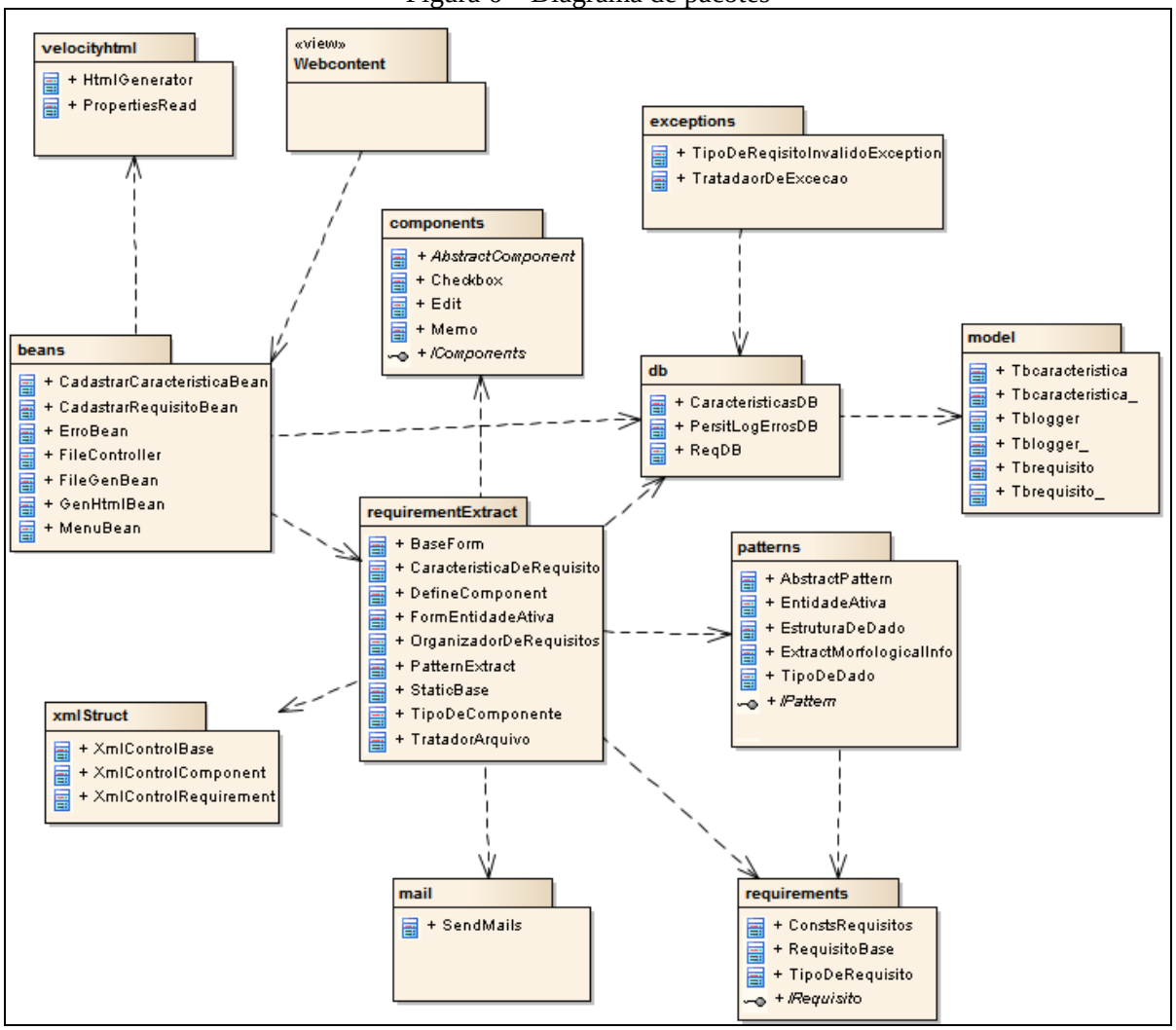
i) `requirementExtract`: é o pacote principal onde estão contidas as classes que controlam o reconhecimento de componentes a partir de requisitos;

j) `velocityHtml`: contém as classes necessárias para transformar o arquivo de definição gerado em um arquivo HTML;

k) `webcontent`: armazena os arquivos que definem toda a implementação web do protótipo;

l) `xmlStruct`: contém as classes de manipulação de arquivos XML.

Figura 6 – Diagrama de pacotes

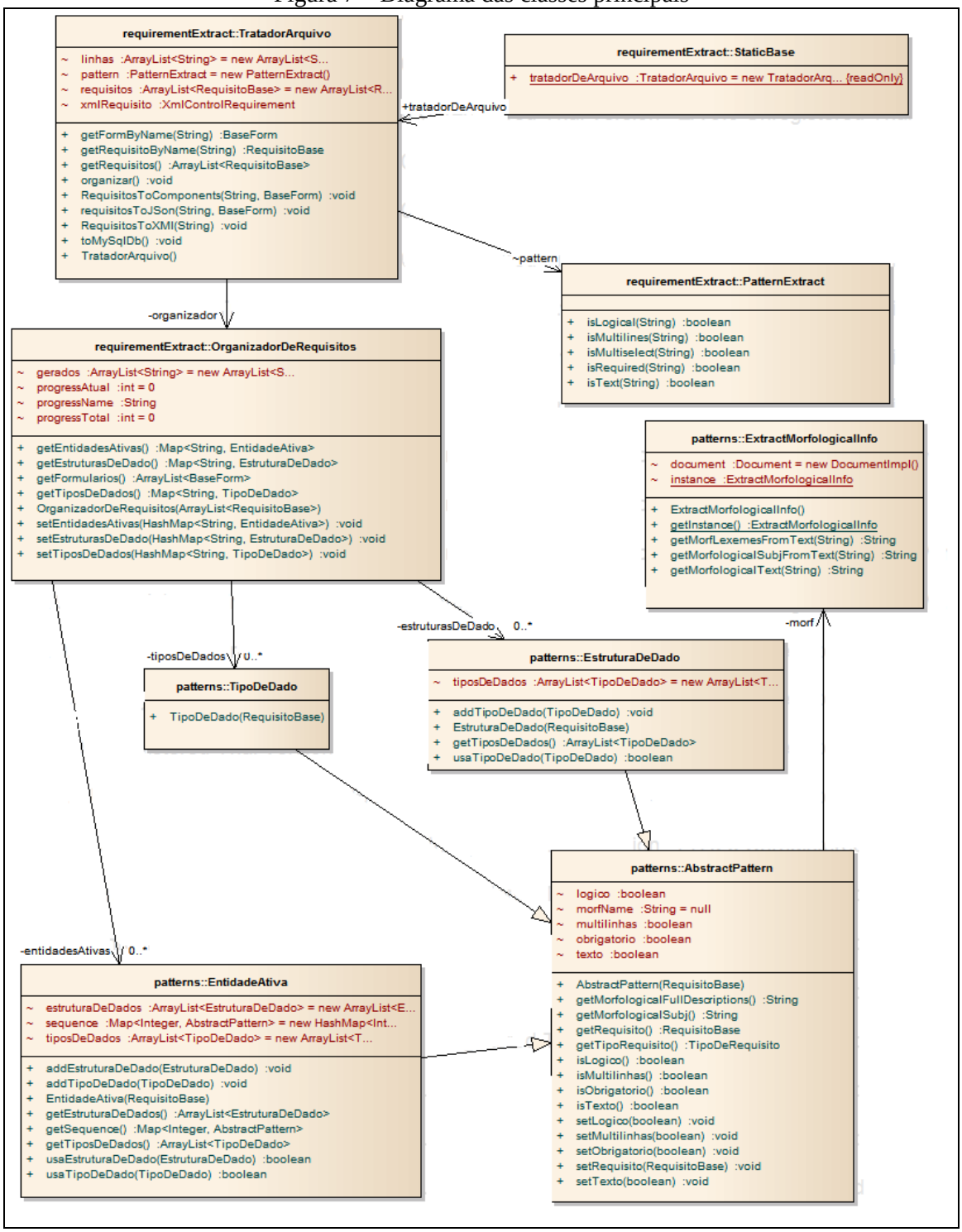


3.2.3 Diagrama de classes

Nesta seção são descritas as principais classes do protótipo desenvolvido. A Figura 7 apresenta o diagrama de classes com as principais classes do processo de extração e organização das informações dos requisitos cadastrados para gerar o arquivo de definição que posteriormente pode ser utilizado para gerar um código fonte. Estas classes são classes de controle e atuam em diferentes pontos do fluxo da geração. Nos relacionamentos em que não são apresentadas as cardinalidades considera-se a existência de um único objeto da classe destino na classe origem.

A classe `StaticBase` tem um objeto estático que é utilizado na aplicação, responsável por disparar todo o processo de controle da extração de informações. Este objeto é uma instância da classe `TratadorArquivo`, responsável por manter a lista de requisitos e por disparar a transformação destes requisitos em arquivos de definição (JSON ou XML), além de ser a interface de interação com a classe `OrganizadorDeRequisitos`.

Figura 7 – Diagrama das classes principais



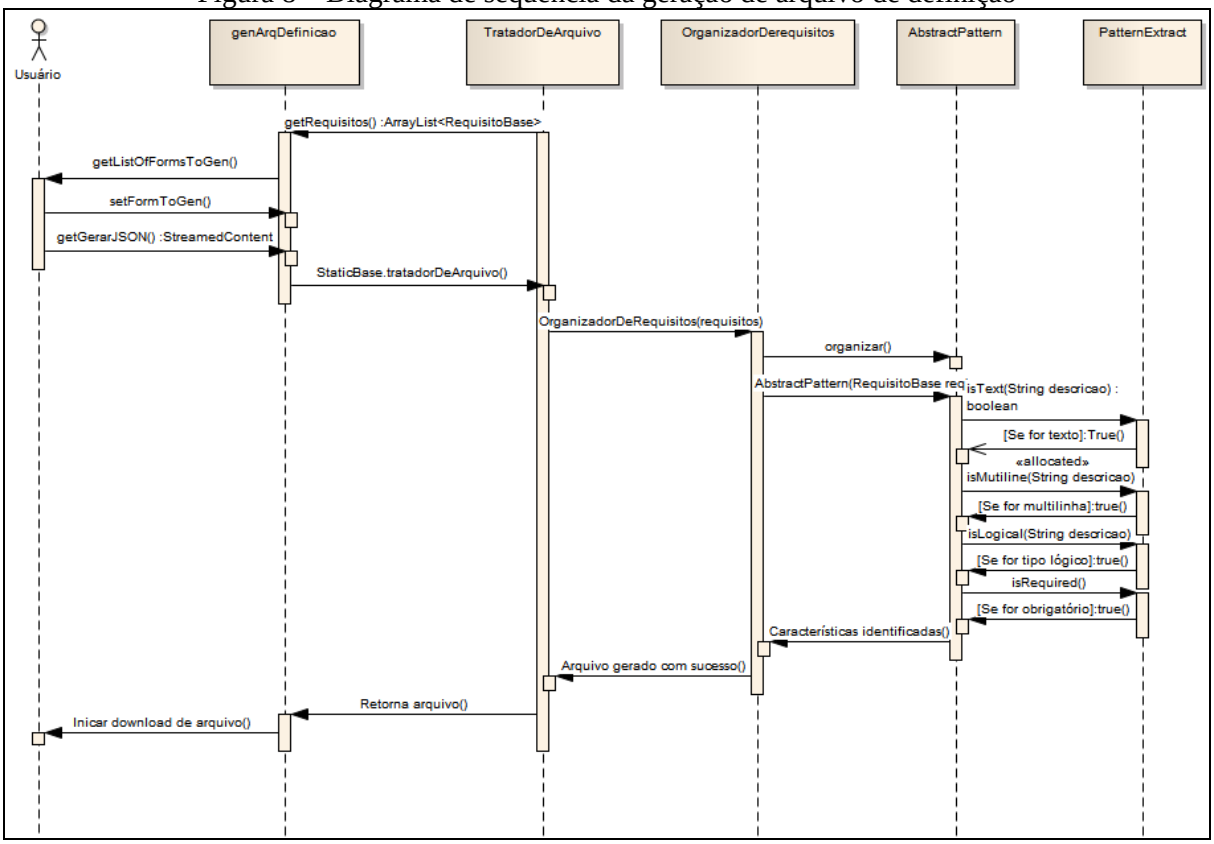
A classe `OrganizadorDeRequisitos` mantém a lista de requisitos agrupados por tipo de padrão de requisitos, assim como também é responsável por todo o processo de mapeamento de dependências entre os requisitos, como por exemplo quais requisitos tipo de dados estão associados à entidade ativa ou estrutura de dados.

A classe `EntidadeAtiva` é a responsável por representar requisitos entidade ativa, armazenando requisitos do padrão de requisitos tipo de dados e estrutura de dados. Esta classe, assim como as classes `TipoDeDados` e `EstruturaDeDados`, é derivada da `AbstractPattern`, que é base para as classes de tipo de padrões de requisitos e é responsável pela identificação das características dos requisitos, como obrigatoriedade e se é um requisito de valores lógicos. Para executar este processo, utiliza a classe `ExtractMorfologicalInfo`, que usa o analisador morfológico para extrair as informações das palavras.

3.2.4 Diagrama de sequência

A seguir (Figura 8) é apresentado o diagrama de sequência referente à funcionalidade de geração de arquivo de definição JSON. Inicialmente são apresentados ao usuário os formulários para os quais ele pode gerar um arquivo de definição (`getListOfFormsToGen`). Após o usuário selecionar o formulário desejado, o método `organizar` faz a organização dos requisitos em listas, uma para cada tipo de padrão de requisito, criando, para cada requisito, uma classe derivada de `AbstractPattern`, do padrão correspondente. Para tanto, a classe `PatternExtract` identifica as características suportadas pelo protótipo através dos métodos `isTexto`, `isMutiline`, `isLogical` e `isRequired`. Tendo as características mapeadas, são disparados métodos para identificar as relações entre todos os requisitos (`organizarEstruturaDeDado`, `organizarEntidadeAtiva`). Em seguida, é executado o método `generateComponents`, que lê as características reconhecidas e identifica quais são os componentes de interface adequados para cada requisito. Por fim, é gerado o arquivo JSON que é enviado para o usuário salvá-lo no local desejado.

Figura 8 – Diagrama de sequência da geração de arquivo de definição



3.3 IMPLEMENTAÇÃO

A seguir são relacionadas as técnicas e as ferramentas utilizadas na implementação, assim como são descritos os principais processos e a operacionalidade do protótipo.

3.3.1 Técnicas e ferramentas utilizadas

As técnicas e ferramentas utilizadas para o desenvolvimento do protótipo proposto foram as seguintes:

a) a IDE Eclipse e a linguagem Java para a codificação do protótipo;
b) o banco de dados MySQL para armazenamento de informações;
c) a biblioteca `EclipseLink`, que implementa a especificação JPA, para abstração da comunicação com o banco de dados;
d) o *framework* `Primefaces`, que implementa a especificação Java *Server Faces* (JSF), para abstração de implementação das páginas web;
e) o motor de *templates* Velocity para a geração de páginas HTML a partir do arquivo de definição gerado;
f) a biblioteca `CoGrOO` para a análise léxico morfológica de textos em português dos requisitos especificados.

CoGrOO é uma biblioteca de código livre, sob licença GNU *Lesser General Public License* (LGPL), sendo seus mantenedores oficiais o Centro de Competência em Software Livre (CCSL) da Universidade de São Paulo (USP). O Quadro 30 apresenta um exemplo de sua utilização. Inicialmente deve-se inicializar a biblioteca informando a linguagem natural dos textos que serão analisados, no caso português do Brasil (`pt, BR`). Após cria-se um objeto (`document`) que representa o texto a ser analisado e inicializa-o com o texto propriamente dito (`O rato roeu a roupa do rei de Roma`). Em seguida é feita a análise deste documento através do método `cogroo.analyze`. Por último, é feita uma iteração em todos os *tokens* analisados a fim de buscar as informações morfológicas identificadas pelo CoGrOO. Nesta iteração são extraídas as informações de categoria gramatical, gênero, número, entre outras informações, que podem ser usadas em uma análise textual.

Quadro 30 – Exemplo de uso do CoGrOO

```
// inicialização do CoGrOO
ComponentFactory factory = ComponentFactory.create(new Locale("pt","BR"));
Analyzer cogroo = factory.createPipe();

// documento a ser analisado
Document document = new DocumentImpl();

// frase que se deseja analisar
String documentText = "O rato roeu a roupa do rei de Roma";
String temp = Normalizer.normalize(documentText, java.text.Normalizer.Form.NFD);
temp.replaceAll(":", " ");
documentText = temp.replaceAll("[^\\p{ASCII}]", "");
document.setText(documentText);

// análise do documento
cogroo.analyze(document);

// itera nas informações do analisador para recuperar as informações desejadas
for (Sentence sentence : document.getSentences()) {
   System.out.println("Sentença: "+sentence.getText()+"\n");
   System.out.println("Tokens: \n");

   // para cada token encontrado na frase
   for (Token token : sentence.getTokens()) {
      // retorna um token da frase original
      String lexeme = token.getLexeme();
      // retorna a forma simples da palavra
      String lemmas = Arrays.toString(token.getLemmas());
      // retorna a tag que identifica a categoria do token (substantivo, adjetivo,
      // verbo, entre outras)
      String pos = token.getPOSTag();
      // retorna a estrutura (feminino, masculino, singular, plural, tempo verbal
      // entre outras)
      String feat = token.getFeatures();
      System.out.println
      (String.format(" %-10s %-12s %-6s %-10s\n", lexeme, lemmas, pos, feat));
   }
}
```

O Quadro 31 apresenta a saída das informações processadas no Quadro 30.

Quadro 31 – Saída do exemplo do uso do CoGrOO

```
Sentença: O rato roeu a roupa do rei de Roma
Tokens:
o             [o]           art         M=S
rato          [rato]        n           M=S
roeu          [roer]        v-fin       PS=3S=IND
a             [o]           art         F=S
roupa         [roupa]       n           F=S
de            [de]          prp         -
o             [o]           art         M=S
rei           [rei]         n           M=S
de            [de]          prp         -
Roma          [Roma]        prop        F=S
```

3.3.2 Associação entre os requisitos

Para que seja possível gerar código fonte a partir de padrões de requisitos, primeiro é feita a associação entre os requisitos descritos:

a) associam-se os requisitos tipo de dados a estrutura de dados e entidades ativas;

b) associam-se os requisitos estrutura de dados a outras estruturas de dados e entidades ativas.

Para cada requisito cadastrado é verificado quem o utiliza, buscando o nome do requisito na descrição de outros requisitos. O Quadro 32 apresenta a implementação do método `organizarEstruturaDeDado` da classe `OrganizadorDeRequisitos`, que busca tipo de dados em estrutura de dados.

Quadro 32 – Código fonte: organização do requisito estrutura de dados

```
private void organizarEstruturaDeDado() {
   progressName = "Organizando Estrutura de Dados";
   TipoDeDado td;
   EstruturaDeDado ed;
   Set<String> tiposDeDadosNomes = tiposDeDados.keySet();
   Set<String> estruturasNomes = estruturasDeDado.keySet();

   for (Iterator<String> iterator = tiposDeDadosNomes.iterator();
        iterator.hasNext();) {
      String chave = iterator.next();
      if (chave != null) {
         td = (TipoDeDado) tiposDeDados.get(chave);
         for (Iterator<String> iteratorED = estruturasNomes.iterator();
               iteratorED.hasNext();) {
            String chaveED = iteratorED.next();
            if (chaveED != null) {
               ed = ((EstruturaDeDado) estruturasDeDado.get(chaveED));
               if (ed.usaTipoDeDado(td)) {
                  ed.addTipoDeDado(td);
               }
            }
         }
      }
   }
}
```

3.3.3 Extração das características de um requisito

O próximo passo consiste na extração das características para determinar qual o componente de interface que deve ser gerado. Para isto busca-se nas descrições dos requisitos tipo de dados palavras tratadas como substantivos e adjetivos que sejam correspondentes a características listadas no Quadro 33.

Quadro 33 – Lista de características

descrição do requisito	característica	exemplo de uso
texto, alfanumérico, numérico, caracteres	texto	O requisito é **texto**.
lógico	lógico	É um campo **lógico**.
texto longo, multilinha	texto de múltiplas linhas, texto longo	É um **texto longo**.
obrigatório	campo obrigatório	É um campo **obrigatório**.
opcional	campo opcional	O campo é um valor **opcional**.

Estas características ficam armazenadas no banco de dados e a extração é feita através da classe `PatternExtract`. A extração dos substantivos e adjetivos é feita com o uso do CoGrOO através do método `getMorfCaracteristicasFromText` (Quadro 34).

Quadro 34 – Código fonte: extração de substantivos e adjetivos

```
public String getMorfCaracteristicasFromText(String text) {
   text = text.toLowerCase();
   analyzeText(text);
   StringBuilder fulltext = new StringBuilder();

   for (Sentence sentence : document.getSentences()) {
      for (Token token : sentence.getTokens()) {
         String lexeme = token.getLexeme();
         String pos = token.getPOSTag().toLowerCase();
         if (pos.equals("adj") || pos.equals("n")) {
            fulltext.append(lexeme+" ");
         }
      }
   }
   return fulltext.toString();
}
```

O Quadro 35 apresenta o código que, a partir das informações morfológicas extraídas, usando expressões regulares, determina se a informação é ou não obrigatória.

Quadro 35 – Código fonte: extração de características obrigatórias

```
public boolean isRequired(String requirementDesc) {
   String obrigatorio = getCaracteristicaObrigatorio();
   String opcional = getCaracteristicaOpcional();

   Pattern padrao = Pattern.compile(obrigatorio, Pattern.CASE_INSENSITIVE);
   Matcher pesquisa = padrao.matcher(requirementDesc);
   if (pesquisa.find())
      return true;
   padrao = Pattern.compile(opcional, Pattern.CASE_INSENSITIVE);
   pesquisa = padrao.matcher(requirementDesc);
   if (pesquisa.find())
      return false;
   return false;
}
```

3.3.4 Definição dos componentes de interface a partir das características

Com as características extraídas, determina-se o componente de interface que deve ser gerado. Para isto, definiu-se uma representação para cada padrão de requisito tratado, conforme apresentado no Quadro 36.

Quadro 36 – Representação dos padrões de requisitos

padrão de requisito	representação	exemplo
tipo de dados	componente de interface, podendo ser texto (`edit`), memorando (`memo`) ou marcação (`checkbox`)	CEP
estrutura de dados	conjunto de componentes de interface que compõem uma única informação	Endereço
entidade ativa	formulário (tela) formado por um conjunto de informações, podendo ser estruturas de dados ou tipos de dados.	Cadastro de endereço

Ainda, as características listadas no Quadro 33 foram mapeadas para componentes de interface (Quadro 37). Observa-se que o reconhecimento é limitado à lista de palavras léxicas computacionais cadastradas no protótipo, conforme apresentado no Quadro 29.

Quadro 37 – Características versus componentes de interface

característica	componente
texto	`edit`
lógico	`checkbox`
texto longo, multilinha	`memo`
obrigatório	esta característica é combinada com as anteriores para informar que o preenchimento é obrigatório
opcional	esta característica é combinada com as anteriores para informar que o preenchimento é opcional

O método `getComponentFromPattern`, visualizado no Quadro 38, é responsável por determinar o componente de interface correspondente a cada característica extraída dos requisitos.

Quadro 38 – Código fonte: definição do componente de interface a partir das características

```
public AbstractComponent getComponentFromPattern(AbstractPattern pattern) {
   AbstractComponent comp = null;
   String descricao = pattern.getRequisito().getNome();
   String nome =  descricao.replaceAll("\\s","");
   if (pattern.isTexto()) {
      if (pattern.isMultilinhas()) {
         comp = new Memo();
         comp.setNome("mm"+nome);
      }  else {
         comp = new Edit();
         comp.setNome("ed"+nome);
      }
   } else if (pattern.isLogico()) {
            comp = new Checkbox();
            comp.setNome("cb"+nome);
   } else {
      comp = new Edit();
      comp.setNome("ed"+nome);
   }
   comp.setDescricao(descricao);
   comp.setObrigatorio(pattern.isObrigatorio());
   return comp;
}
```

3.3.5 Geração do arquivo de definição

A geração do arquivo de definição ocorre iterando na estrutura montada pela classe `OrganizadorDeRequisitos`, descrita na seção 3.3.2 A partir dessa estrutura, busca-se as entidades ativas, considerando que cada entidade ativa compõe um formulário diferente. Tem-se então uma listagem de todos os formulários cadastrados a partir dos quais são gerados os arquivos de definição. O Quadro 39 mostra um trecho do código que gera o arquivo de definição no formato XML.

Quadro 39 – Código fonte: geração do arquivo de definição em XML

```
for (int i = 0; i < forms.size(); i++) {
   form = forms.get(i);
   el = xmlComponente.addNewSection(raiz, "formulario");
   xmlComponente.addSingleNodeText(el, "Titulo", form.getTitle());
   componentes = xmlComponente.addNewSection(el, "componentes");
   for (int j = 0; j < form.getComponents().size(); j++) {
      comp = form.getComponents().get(j);
      compType = comp.getTypeStr();
      nodeComponente = xmlComponente.addNewSection(componentes, "componente");
      xmlComponente.addSingleNodeText(nodeComponente, "tipo", compType);
      xmlComponente.addSingleNodeText(nodeComponente, "nome", comp.getNome());
      xmlComponente.addSingleNodeText(nodeComponente, "descricao",
      comp.getDescricao());
      if (comp.isObrigatorio()) {
         xmlComponente.addSingleNodeText(nodeComponente,"obrigatorio", "sim");
      } else {
         xmlComponente.addSingleNodeText(nodeComponente,"obrigatorio", "nao");
      }
   }
}
```

3.3.6 Geração de código HTML a partir do arquivo de definição

O último processo implementado no protótipo foi gerar um arquivo HTML a partir do arquivo de definição para mostrar o resultado da identificação de componentes de interface a partir da descrição de requisitos, validando a ferramenta desenvolvida. Nesta etapa optou-se por utilizar o motor de *templates* Velocity. O Quadro 40 apresenta o *template* que gera campos `edit` a partir de uma coleção (`$componentes`) contendo os componentes reconhecidos.

Quadro 40 – Exemplo de *template* Velocity

```
#foreach ($comp in $componentes)
   #if($comp.getTipo() == "tcEdit")
      <li >
      <label id="$comp.getNome()">
         $comp.getDescricao()
         #if($comp.getObrigatorio() == "sim")
            <span >*</span>
         #end
      </label>
      <div>
         <span>
            <input type="text" size="10" name="$comp.getNome" " />
         </span>
      </div>
      </li>
   #end
#end
```

3.3.7 Operacionalidade da ferramenta

O protótipo, denominado Projeto Elicitar, foi disponibilizado na Amazon Web Service sob a *Uniform Resource Locator* (URL) `http://elicitar.elasticbeanstalk.com`. A tela principal é apresentada na Figura 9.

Figura 9 – Tela principal do protótipo

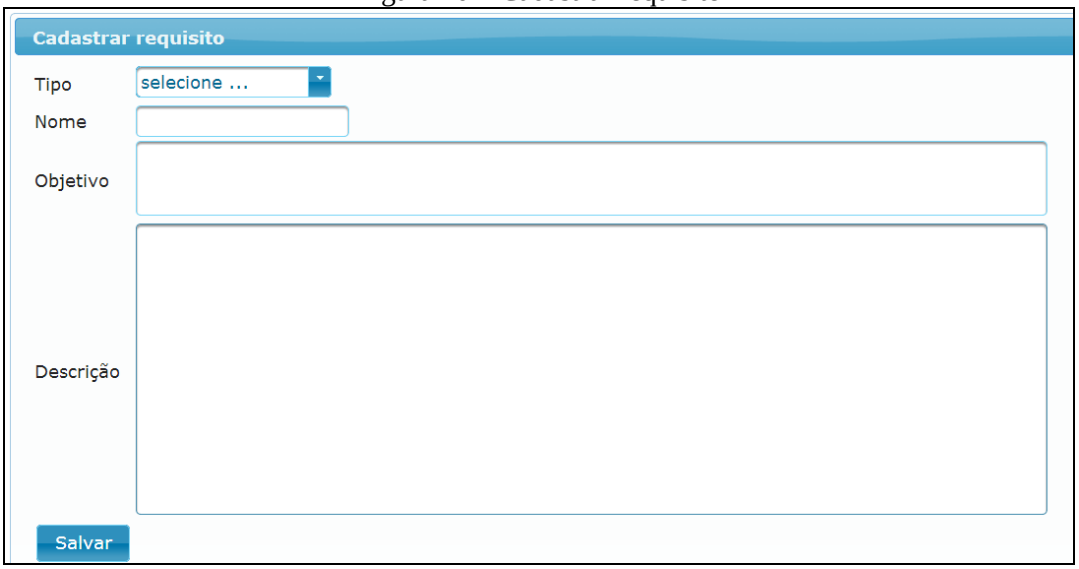
As funcionalidades principais são: cadastrar requisito, gerar definição, gerar HTML e cadastrar característica. As funcionalidades secundárias são:

a) enviar *feedback*, que contém um questionário a ser respondido pelos usuários com uma avaliação da ferramenta;
b) abrir proposta, que contém a proposta contendo os objetivos da ferramenta;
c) novidades da versão, com a documentação das alterações da ferramenta;
d) abrir ajuda, que contém uma documentação explicando e exemplificando o uso do protótipo.

3.3.7.1 Cadastrar requisito

Esta tela do protótipo contém duas partes: a primeira contém os campos para cadastrar requisitos e a segunda contém a relação de todos os requisitos cadastrados. A Figura 10 apresenta os campos para cadastrar requisitos.

Figura 10 – Cadastrar requisito



Para cadastrar um requisito deve-se preencher os seguintes campos:

a) `tipo`: representa o padrão de requisito que se deseja cadastrar, podendo ser tipo de dados, estrutura de dados ou entidade ativa. Quando for selecionado um tipo de padrão de requisito, caso o campo `descrição` esteja vazio, o protótipo preenche o campo `descrição` com um modelo padrão de texto e um exemplo para orientação da forma com que o requisito deve ser descrito. O exemplo de modelo para descrição de requisitos tipo de dados é apresentado no Quadro 41;
b) `nome`: representa o nome do requisito a ser cadastrado, sendo utilizado para reconhecer o requisito como único e também para referências em outros requisitos;
c) `objetivo`: é uma descrição breve do objetivo do requisito, informativa, não sendo considerada no processamento do protótipo;

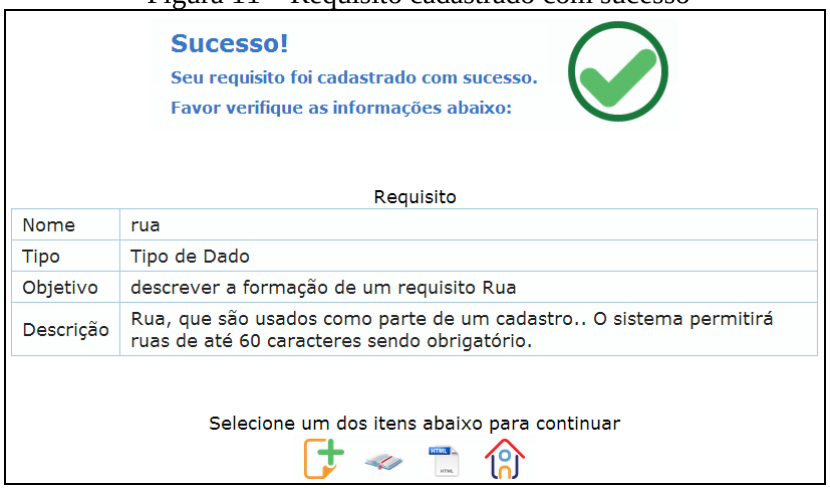
d) `descrição`: é utilizado para descrever as informações do requisito.

Quadro 41 – Exemplo de modelo de requisito: tipo de dado

A descrição pode ser preenchida conforme o modelo abaixo: <<Nome do tipo de dado>>, que são usados para <<Finalidade do Tipo de Dado>>, deve ser do formulário <<Formulário Tipo de Dado>>. (<<Declaração (descrição) do formato da exibição>>). (<<Restrições das declarações>>.) (<<Manipulação especial da declaração>>.) EXEMPLO: O sistema permitirá endereços de e-mail de até 60 caracteres.

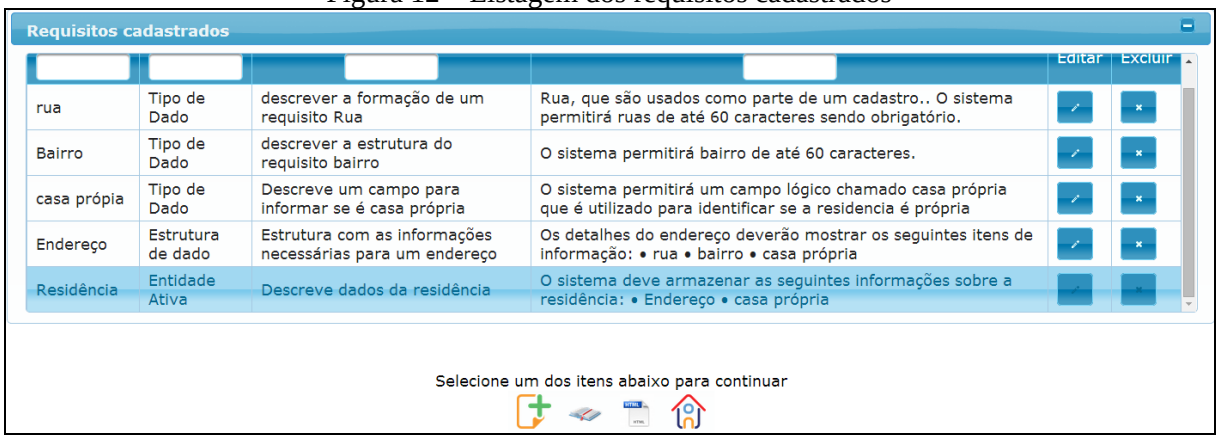
Ao preencher os campos e pressionar o botão salvar, conforme pode ser observado na Figura 11, são apresentadas a mensagem de cadastro de requisito realizado com sucesso e as informações cadastradas para conferência.

Figura 11 – Requisito cadastrado com sucesso



Pode-se verificar o requisito cadastrado na seção que apresenta todos os requisitos cadastrados no protótipo (Figura 12). Na seção requisitos cadastrados são apresentadas as informações de cada requisito e, ao lado, dois botões, um para alterar e outro excluir o requisito. Ao pressionar o botão alterar, os campos apresentados anteriormente (Figura 10) são preenchidos com as informações do requisito, permitindo que o usuário altere-os conforme desejado. Ao pressionar o botão excluir, é apresentada uma mensagem solicitando a confirmação, que, em caso de ser positiva, exclui o requisito e o elimina da seção de requisitos cadastrados.

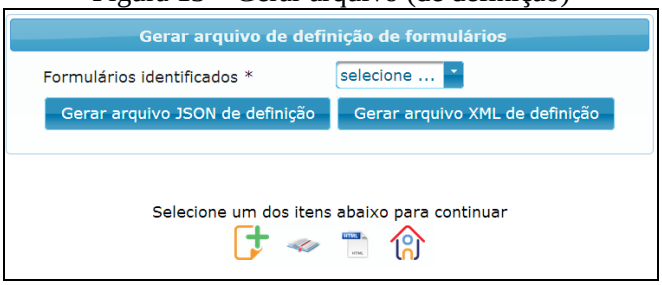
Figura 12 – Listagem dos requisitos cadastrados



3.3.7.2 Gerar definição

Para gerar um arquivo de definição deve-se selecionar um formulário entre os formulários identificados pelo protótipo e em seguida pressionar o botão correspondente ao arquivo de saída desejado: JSON ou XML. O usuário deverá selecionar o local para salvar o arquivo a ser gerado e pressionar o botão salvar. A tela para gerar arquivo de definição é apresentada na Figura 13.

Figura 13 – Gerar arquivo (de definição)



A partir dos requisitos `endereço` e `própria` já cadastrados que compõem o formulário `Residência`, ilustrados na Figura 12, é gerado o arquivo de definição JSON do Quadro 42.

Quadro 42 – Exemplo de JSON gerado

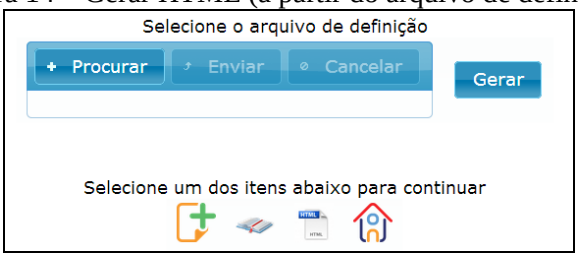
```
{"formularios":
[
  {"formulario0":{"Titulo":"Residência",
                  "componentes": {"componente1": {"tipo":"tcEdit",
                                                  "nome":"edrua",
                                                  "obrigatorio":true,
                                                  "descricao":"rua"          },
                                  "componente0": {"tipo":"tcCheckbox",
                                                  "nome":"cbcasapropria",
                                                  "obrigatorio":false,
                                                  "descricao":"casa própria" },
                                  "componente2": {"tipo":"tcEdit",
                                                  "nome":"edBairro",
                                                  "obrigatorio":false,
                                                  "descricao":"Bairro"       }
                                 }
                 }

] }
```

3.3.7.3 Gerar HTML

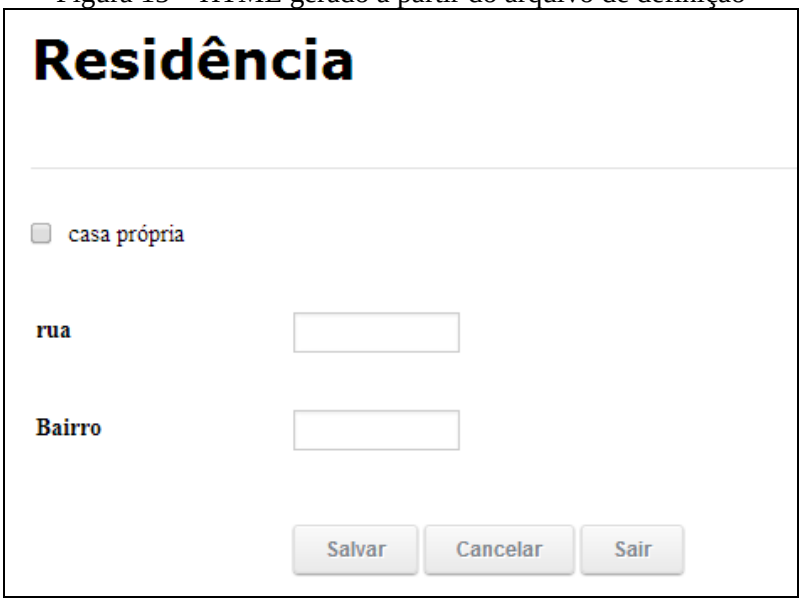
A Figura 14 apresenta a última etapa do processo suportado pelo protótipo. Nesta etapa seleciona-se o arquivo de definição, gerado conforme descrito na seção anterior, através do botão procurar. Feito isto, o botão enviar é habilitado, devendo ser pressionado para que o arquivo de definição seja enviado para o protótipo. Por último, pressiona-se o botão gerar. Neste momento o protótipo processa o arquivo de definição, gerando o HTML correspondente. O usuário deverá selecionar o local para salvar o arquivo e pressionar o botão salvar.

Figura 14 – Gerar HTML (a partir do arquivo de definição)



A partir do arquivo de definição JSON apresentado no Quadro 42 é gerado o HTML cujo formulário é apresentado na Figura 15.

Figura 15 – HTML gerado a partir do arquivo de definição

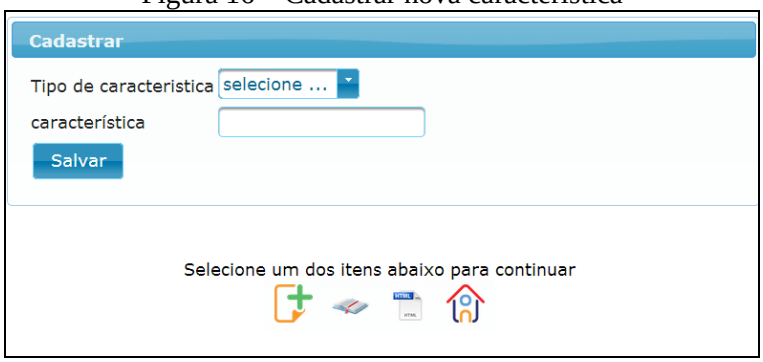


3.3.7.4 Cadastrar característica

É possível aumentar o dicionário de conhecimento do protótipo com características (Figura 16). Para cadastrar uma nova característica, seleciona-se o tipo da característica no campo correspondente, podendo ser: obrigatório, opcional, multilinha, texto, lógico. Após

isto, deve-se preencher o campo característica. Ao pressionar em salvar, a nova característica será cadastrada.

Figura 16 – Cadastrar nova característica



3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o desenvolvimento de um protótipo que gera arquivos de definição (JSON ou XML) para geração de telas através da análise de especificações de requisitos escritas conforme os padrões de requisitos propostos por Withall (2007). O protótipo foi disponibilizado para usuários finais sob a URL `http://elicitar.elasticbeanstalk.com/` em 26 de maio de 2014. Além das funcionalidades apresentadas na seção anterior, é possível preencher um questionário (Apêndice A), permitindo que os usuários enviem um *feedback* com a percepção de uso do protótipo. A pesquisa foi aplicada sem restrição de perfil de usuário (analista de sistema, projetista, programador, arquiteto ou outra função). Porém, a maioria dos usuários que responderam foram analistas de sistemas e programadores. Foram obtidas dez respostas, resumidas no Quadro 43.

Ao analisar as respostas das perguntas 5, 6 e 9, pode-se observar que o protótipo obteve um bom resultado, pois 44% dos arquivos HTML gerados teve um resultado próximo do esperado para as especificações desenvolvidas pelos usuários, a ferramenta teve a usabilidade aprovada como boa com 78% das respostas e nota geral 4 com 67% das respostas. Este resultado deve levar em consideração que a ferramenta:

a) é limitada ao reconhecimento de três componentes de interface, sendo eles caixa de texto (`edit`), memorando (`memo`) e caixa de checagem (`checkbox`);
b) é limitada à quantidade de características, palavras léxicas computacionais reconhecidas, conforme descrito na seção 3.3.3.

Quadro 43 – Resultado do questionário aplicado

nº	pergunta	respostas
1	Qual sua profissão? Pode ser o papel que desempenha.	programador: 44%
2	Em sua percepção qual o percentual de telas CRUD (*Create*, *Read*, *Update*, *Delete*), ou seja, telas simples de cadastro, de suas aplicações?	entre 41% e 60%: 44% das respostas
3	Como você considera o uso de padrões de requisitos para a especificação de aplicações?	bom: 67% das respostas
4	Como você avalia a aplicabilidade da ferramenta Elicitar para a geração de telas a partir de padrões de requisito?	muito boa: 44% das respostas boa: 44% das respostas
5	O arquivo HTML gerado pela ferramenta Elicitar foi conforme o esperado?	próximo do desejado: 44% das respostas
6	Como você avalia a usabilidade da ferramenta Elicitar?	boa:78% das respostas
7	Você já utiliza alguma ferramenta ou *framework* que facilite a geração de telas?	não: 67 % das respostas
8	Quanto você acredita que uma ferramenta ou *framework* para gerar as telas da aplicação diretamente a partir da especificação aumenta a produtividade?	entre 41% e 60%: 33% das respostas acima de 80%: 33% das respostas
9	De 0 a 5, qual a sua nota geral para a ferramenta Elicitar?	nota 4: 67% das respostas
10	Caso esta ferramenta continue sendo desenvolvida, você a utilizaria profissionalmente?	a maioria das vezes: 44% das respostas

Atualmente a ferramenta tem poucos requisitos cadastrados para efetuar uma análise mais crítica quanto à geração de um dicionário mais rico em palavras que possam ser reconhecidas como componentes de interface. Isto limitou o domínio de palavras existentes. Porém há técnicas que podem ser aplicadas para enriquecer o dicionário de reconhecimento das palavras. O dicionário léxico computacional utilizado na ferramenta foi desenvolvido utilizando requisitos de uma grande empresa de software, com mais de 1 mil funcionários localizada no sul do Brasil. Para isto foi criada uma transcrição de um projeto desta empresa para padrões de requisitos. Tendo os requisitos escritos em padrões de requisitos, desenvolveu-se uma estrutura de entrada e um algoritmo para transformar estes requisitos em componentes de interface. Durante o desenvolvimento do protótipo foi identificada uma base formal em formato de ontologia denominada *Semantics of Business Vocabulary and Rules* (SBVR)[4] que pode ser adotada para identificar as características dos requisitos em associação com o PLN.

Apesar dos resultados obtidos, este trabalho não atingiu todos os objetivos específicos propostos. Atualmente a ferramenta gera como saída, além do arquivo de definição, um arquivo HTML e não código fonte em pelo menos uma linguagem de programação.

[4] SBVR "é um padrão da OMG para descrição formal das regras de negócio. Esse padrão fornece uma linguagem abstrata baseada em lógica de primeira ordem com uma extensão de lógica modal que pode ser utilizada pelas organizações para definição da semântica de vocabulário de negócio e regras de negócio" (JESUS, 2013, p.7).

Com relação aos trabalhos correlatos, a ferramenta Elicitar mais se aproxima do que foi realizado por Bhagat et al. (2012). A seguir no Quadro 44 é apresentada uma breve comparação entre eles.

Quadro 44 – Comparação com trabalho correlato

característica	ferramenta Elicitar	Bhagat et al. (2012)
plataforma	web	desktop
linguagem natural reconhecida	português	inglês
formato do texto de entrada	requisitos escritos em padrões de requisitos	texto livre
formato de saída	XML ou JSON com definições	texto livre
geração de código	arquivo de definição	código Java ou VB.NET

4 CONCLUSÕES

Hoje mesmo com a evolução das ferramentas disponíveis para melhorar o desenvolvimento dos artefatos gerados pela engenharia de software, ainda há dificuldades no levantamento e documentação de requisitos. Para solucionar o problema podem ser usados tanto padrões de requisitos como forma de melhorar a descrição dos mesmos, como também ferramentas para extrair informações diretamente dos requisitos. Essa foi a solução proposta.

Para desenvolver a ferramenta que extrai informações a partir de requisitos e as transforma em códigos fonte (ou outros artefatos) foi necessário o estudo de padrões de requisitos (entrada da ferramenta), bem como de processamento de linguagem natural (técnica usada para processar a entrada) e de geração de código (para definição da saída).

Durante o desenvolvimento do trabalho ficou claro que a quantidade de informações possíveis na descrição de um requisito é bastante extensa. Isto porque, sendo escrito em linguagem natural, não existe uma sequência única de descrição, nem limitação nos termos adotados. Por isso, este trabalho ficou restrito ao reconhecimento de um domínio pequeno de padrões de requisitos (tipo de dados, estrutura de dados, entidade ativa) e de componentes de interface (`edit`, `memo`, `checkbox`). Contudo, apesar das restrições, os resultados obtidos foram bons e os objetivos propostos alcançados.

Observou-se, na etapa de testes, ganho com a especificação de requisitos feita utilizando padrões de requisitos, já que os mesmos se assemelham a padrões de desenvolvimento, o que gera reusabilidade dos requisitos cadastrados na ferramenta. Ainda durante os testes realizados, pode-se identificar falhas de especificação na documentação de requisitos, que foram rapidamente percebidas e corrigidas graças a geração da interface a partir destes requisitos, validando a ferramenta desenvolvida.

As tecnologias e ferramentas adotadas mostraram-se eficazes e tornaram mais ágil o desenvolvimento do protótipo apesar do pouco conhecimento destas no início do projeto. O CoGrOO mostrou-se um analisador morfológico adequado com resultados semelhantes a outros analisadores, com a vantagem de não necessitar acesso à internet por se tratar de uma biblioteca off-line. A linguagem Java foi uma escolha assertiva para este projeto, pois o CoGrOO é escrito nessa linguagem de programação, facilitando a integração. Outras bibliotecas também foram utilizadas para o desenvolvimento deste protótipo em função disto, quais sejam: `Primefaces`, usada para o desenvolvimento da interface web do protótipo, `Eclipselink`, para abstração do banco de dados, e Velocity, para geração do arquivo HTML a partir do arquivo de definições gerado pela ferramenta. A escolha do formato do arquivo de

definição em JSON, por se tratar de um arquivo texto, tornou o processamento simples, podendo ser integrado a outras plataformas, independente de codificação de caracteres como pode ocorrer com outros formatos. Por último, a publicação da ferramenta na Amazon Web Services (AWS) permitiu disponibilizá-la de forma rápida, tornando possível a execução de testes por usuários finais.

4.1 EXTENSÕES

Existem pontos a serem melhorados e incrementados no protótipo desenvolvido, sendo eles:

a) suportar outros padrões de requisitos propostos por Withall (2007) que complementam informações para geração de interfaces, como por exemplo os padrões do domínio de interface de usuário;
b) suportar novos componentes de interface, como `combobox`, `radiobutton` e `grid`;
c) implementar técnicas de inteligência artificial como ontologias para a identificação de características;
d) usar SBVR para a identificação de características;
e) gerar documento formal com os requisitos cadastrados;
f) gerar código fonte em linguagem de programação a partir do arquivo de definição, utilizando assim abordagem *Model Driven Architecture* (MDA);
g) implementar um *workflow* para controle do processo de entrada e saída do protótipo;
h) transformar o protótipo em *plugin* para IDEs como Eclipse e Genexus[5].

[5] GeneXus é uma ferramenta, desenvolvida pela Artech, destinada a ajudar analista e usuários durante todo o ciclo de vida de uma aplicação, sendo capaz de manter de forma automática a estrutura base e a aplicação. A aplicação pode ser gerada para diferentes plataformas (ARTECH CONSULTORES, 2012).

REFERÊNCIAS

AMBLER, Scott W. **Modelagem ágil**: práticas eficazes para a programação eXtrema e o processo unificado. Porto Alegre: Bookman, 2003.

ARTECH CONSULTORES. **GeneXus:** overview. Chicago, 2012. Disponível em: <http://www.genexus.com/files/wp-visiongeneral?pt>. Acesso em: 15 jul. 2014.

BHAGAT, Sujata et al. Class diagram extraction using NLP. In: INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ENGINEERING & TECHNOLOGY - SPECIAL ISSUE OF INTERNATIONAL JOURNAL OF ELECTRONICS, COMMUNICATION & SOFT COMPUTING SCIENCE & ENGINEERING, 1st, 2012, Meerut. **Proceedings…** Meerut: Subharti Institute of Technology & Engineering, 2012. p. 125-128. Disponível em: <http://www.ijecscse.org/papers/SpecialIssue/comp2/190.pdf>. Acesso em: 14 set. 2013.

CROCKFORD, Douglas. **RFC 4627**: the application / json media type for JavaScript Object Notation (JSON). [S.l.], 2006. 10 p. Disponível em: <http://www.ietf.org/rfc/rfc4627.txt?number=4627>. Acesso em: 11 set. 2013.

DECARLE, Luiz S.; GRAHL, Everaldo A. **Experiência prática de aplicação de padrões de requisitos de software**. 2008. 9 f. Artigo de Conclusão de Curso (Especialização em Gestão de Desenvolvimento de Software), Instituto Catarinense de Pós-Graduação, Blumenau.

FALBO, Ricardo A. **Engenharia de requisitos**. Espírito Santo, 2012. Disponível em: <http://www.inf.ufes.br/~falbo/files/Notas_Aula_Engenharia_Requisitos.pdf>. Acesso em: 14 set. 2013.

HERRINGTON, Jack. **Code generation in action**. Greenwich: Manning, 2003.

JESUS, Jandisson S. **Um método para implementação de regras de negócio a partir de semântica SBVR**. 2013. 91 f. Dissertação (Mestrado em Ciências) **–** Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-22012014-144303/pt-br.php>. Acesso em: 22 jun. 2014.

MARQUES, Tiago W. **Gerência de requisitos com adoção de padrões**. 2008. 124 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2008-II/2008-2-26-vf-tiagowmarques.pdf>. Acesso em: 25 out. 2013.

MEDEIROS, José C. D. **Processamento morfológico e correcção ortográfica do português**. 1995. 207 f. Dissertação (Mestrado em Engenharia Eletrotécnica e de Computadores) **–** Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa. Disponível em: <http://bibliotecas.utl.pt/cgi-bin/koha/opac-detail.pl?biblionumber=43755&shelfbrowse_itemnumber=54232>. Acesso em: 15 jun. 2014.

OLIVEIRA NETO, João M.; TONIN, Sávio D.; PIETRICH, Soraia S. Processamento de linguagem natural e suas aplicações computacionais. In: ESCOLA REGIONAL DE INFORMÁTICA, 2., 2010, Manaus. **Anais eletrônicos...** Manaus: INPA, 2010. Não paginado. Disponível em: <http://www.inpa.gov.br/erin2010/Artigo/Artigo9.pdf>. Acesso em: 01 set. 2013.

PRESSMAN, Roger S. **Engenharia de software**. 6. ed. Tradução Rosângela Delloso Penteado. Porto Alegre: Bookman, 2006.

SILVA, Wilson C.; MARTINS, Luis E. G. Paradigma: uma ferramenta de apoio à elicitação e modelagem de requisitos baseada em processamento de linguagem natural. In: WORKSHOP ON REQUIRIMENTS ENGINEERING, 11th, 2008, Barcelona. **Proceedings...** Barcelona: [S.n.], 2008. p. 140-151. Disponível em <http://www.inf.puc-rio.br/wer/WERpapers/artigos/artigos_WER08/silva.pdf>. Acesso em: 14 set. 2013.

SILVEIRA, Janira. **Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados**. 2006. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2006-I/2006-1janirasilveiravf.pdf>. Acesso em: 14 set. 2013.

SOMMERVILLE, Ian. **Engenharia de software**. 8. ed. Tradução Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa. São Paulo: Pearson Addilson Wesley, 2007.

TAGLIATI, Luca V.; JOHNSON, Roger; ROUSSOS, George. **Requirements analysis evolution through patterns**. Londres, 2007. Disponível em: <http://www.dcs.bbk.ac.uk/~gr/pdf/seke07_requirement_patterns_2.pdf>. Acesso em: 25 out. 2013.

VICENTE, Alexandre M. F. **LexMan:** um segmentador e analisador morfológico com transdutores. 2013. 90 f. Dissertação (Mestrado em Engenharia Informática e de Computadores) – Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa. Disponível em: <https://fenix.tecnico.ulisboa.pt/publico/department/theses.do?method=showThesisDetails&selectedDepartmentUnitID=1911260507896&thesisID=2353642449252&_request_checksum_=0cdf21976e78b993473d2c8668af9a458403a101>. Acesso em: 15 jun. 2014.

WITHALL, Stephen. **Software requirements patterns**. Redmond: Microsoft Press, 2007.

APÊNDICE A – Questionário de avaliação da ferramenta

No dia 26 de maio de 2014 foi disponibilizado, além do protótipo, um questionário para avaliação da ferramenta pelos usuários. Este questionário encontra-se no Quadro 45.

Quadro 45 – Questionário

pergunta	**respostas possíveis**
1 - Qual sua profissão? Pode ser o papel que desempenha (selecione entre as opções o que mais se assemelha com suas funções).	( ) Analista de sistema
	( ) Arquiteto
	( ) Programador
	( ) Projetista
	( ) Outros_____

2 - Em sua percepção qual o percentual de telas CRUD (Create, Read, Update, Delete), ou seja, telas simples de cadastro, de suas aplicações? Escolha o percentual que mais se aproxima.					
Até 20%	Entre 21% e 40%	Entre 41% e 60%	Entre 41% e 60%	Mais de 81%	Minhas aplicações não têm telas CRUD

3 - Como você considera o uso de padrões de requisitos para a especificação de aplicações? Baseie-se na experiência obtida durante o uso da ferramenta Elicitar.					
Muito ruim	Ruim	Regular	Bom	Muito bom	Ótimo

4 - Como você avalia a aplicabilidade da ferramenta Elicitar para a geração de telas a partir de padrões de requisito?					
Muito ruim	Ruim	Regular	Boa	Muito boa	Ótima

5 - O arquivo HTML gerado pela ferramenta Elicitar foi conforme o esperado?				
Nenhum pouco	Pouco	Parcialmente	Próximo do desejado	Gerou perfeitamente

6 - Como você avalia a usabilidade da ferramenta Elicitar?					
Muito ruim	Ruim	Regular	Boa	Muito boa	Ótima

7 - Você já utiliza alguma ferramenta ou framework que facilite a geração de telas?			
Sim, um *framework*	Sim, uma ferramenta	Sim, ambos	Não

8 - Quanto você acredita que uma ferramenta ou framework para gerar as telas da aplicação diretamente a partir da especificação aumenta a produtividade?					
Até 20%	Entre 21% e 40%	Entre 41% e 60%	Entre 61% e 80%	Mais de 81%	O uso de ferramentas desse tipo não aumentam a produtividade

9 - De 0 a 5, qual a sua nota geral para a ferramenta Elicitar?

10 - Caso esta ferramenta continue sendo desenvolvida, você a utilizaria profissionalmente?				
Nunca	Talvez	Algumas vezes	A maioria das vezes	Sempre