UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LEANDRO CAPELLI BOMBASSARO

Geração Assistida de Código XML a Partir da Descrição Textual da Especificação de Elementos Notacionais na BPMN

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Lucinéia Heloisa Thom
Coorientador: Carlos Habekost dos Santos

Porto Alegre

2015

AGRADECIMENTOS

À minha orientadora Lucinéia Heloisa Thom e meu co-orientador Carlos Habekost por toda dedicação e paciência que tiveram comigo.

À minha avó Iria por todo o suporte que me deu.

Aos meus pais por todo amor e carinho.

À minha querida namorada por sempre estar ao meu lado quando eu preciso.

“Do. Or do not. There is no try.”
– Master Yoda (Star Wars Episode V)

RESUMO

Processos de Negócio podem ser encontrados por toda parte. O modo como os processos são definidos e executados são muito importantes para o funcionamento de uma organização. A forma mais usual de documentar os processos é através de diagramas de processo de negócio. Diversas notações existem para criação desses diagramas, sendo a mais adotada pelos participantes dos processos a Notação e Modelo de Processos de Negócio (BPMN). A BPMN apresenta uma codificação XML para seus elementos notacionais, contudo, esta codificação não agrega toda a semântica apresentada pela descrição textual que define os elementos. Nesse contexto, este trabalho apresenta um método para geração de código XML a partir da descrição textual da especificação de elementos notacionais na BPMN. A metodologia desenvolvida para atingir esse objetivo inclui: técnicas de NLP para análise sintática, utilizando-se o *Stanford Parser*; obtenção de estruturas Sujeito-Verbo-Objeto e Sujeito-Verbo-Caso (uma nova estrutura apresentada neste trabalho) a partir das relações obtidas na análise sintática; análise semântica dessas estruturas e a respectiva interpretação das mesmas a um código XML. Um protótipo foi implementado para validação da metodologia proposta obtendo resultados na extração semelhantes aos que podem ser encontrados na especificação da BPMN.

Palavras-chave: BPMN. extração de código. XML. Sujeito-Verbo-Objeto.

Assisted XML Code Generation from the Textual Description Especification of BPMN Notational Elements

ABSTRACT

Business processes can be found everywhere. The way these processes are defined and executed are very important for the operation of an organization. The most common way to document processes is through business process diagrams. Several notations exist for creating these diagrams, being Business Process Model and Notation (BPMN) the most adopted by process users. BPMN provides an XML encoding for its notational elements, yet this encoding does not represents all the semantic defined in the elements textual description definition. In this context, this work presents a method for XML code generation from the textual description of notational elements of Business Process Model and Notation (BPMN). The methodology developed to achieve this goal includes NLP techniques using the Stanford Parser; obtaining Subject-Verb-Object and Subject-Verb-Case (a new structure presented in this work) structures from relations obtained in syntactic analysis; semantic analysis of these structures and their interpretation to an XML code. A prototype was implemented to validate the proposed method obtaining results similar in the code extraction to those that can be found in the specification of BPMN .

Keywords: BPMN. code extraction. XML. Subject-Verb-Object.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BPM	Gerenciamento de Processos de Negócios
BPD	Diagrama de Processo de Negócio
BPMN	Notação e Modelo de Processos de Negócios (*Business Process Model and Notation*)
XOR	desvio exclusivo
AND	desvio paralelo
OR	desvio inclusivo
OMG	*Object Management Group*
NLP	Processamento de Linguagem Natural (*Natural Language Processing*)
POS	*parts of speech*
SVO	Sujeito-Verbo-Objeto
SVC	Sujeito-Verbo-Caso
XML	*eXtensible Markup Language*
XSD	*XML Schema Definiton*
API	*Application Programming Interface*
SCR	*Software Cost Reduction*
GUI	*Graphic User Interface*

SUMÁRIO

1 INTRODUÇÃO

Processos de negócio (neste trabalho também denominado processo por simplificação) podem ser encontrados em toda parte, em uma venda em varejo, manufatura de um produto em uma fábrica, ou serviços como atendimento a um cliente, solicitação de empréstimo, entre outros. Um processo é um conjunto de eventos, atividades e decisões que agregam algum valor para a organização e/ou um ou mais clientes (DUMAS et al., 2013), ou seja, que conjuntamente realizam um objetivo de negócio (WESKE, 2012).

Como todas as ações que uma organização realiza podem ser definidas como processo, é importante que esses estejam bem definidos, uma vez que o modo como são projetados e realizados afeta tanto a qualidade de um serviço que o cliente recebe quanto a eficiência com que esses serviços são entregues (DUMAS et al., 2013).

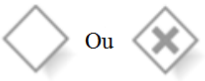
Uma organização pode atingir seu objetivo de negócio de maneira eficiente e efetiva apenas se os participantes envolvidos no processo, assim como respectivos recursos utilizados, como sistemas de informação, estiverem em harmonia (WESKE, 2012). A compreensão sobre processos e a forma de analisá-los e melhorá-los foi, ao longo dos anos, gerando diversas disciplinas, como *Total Quality Management*, *Lean*, *Six Sigma* e outras. A partir da influência dos melhores aspectos de cada disciplina, e adicionando outros aspectos (Ex: melhoria constante, ênfase em tecnologia, entre outros), têm-se o Gerenciamento de Processos de Negócio ou BPM (do inglês *Business Process Management*).

A disciplina de BPM visa a melhoria dos processos de negócio, assumindo que o negócio de uma organização é visto como um conjunto de processos e BPM está agindo de maneira a melhorar esses processos (von ROSING; SCHEER; von SCHELL, 2015). Para facilitar a compreensão e visualização desses processos por todos os seus participantes, foi definida uma notação formal para modelagem dos processos de negócio mantida pela *Object Management Group* (OMG), a BPMN (Notação e Modelo de Processos de Negócio, do inglês *Business Process Model and Notation*), atualmente na versão 2.0.2 (OMG, 2013).

1.1 Motivação

A BPMN oferece uma representação para cada elemento notacional a partir de uma regra textual, uma representação gráfica e uma codificação XSD (*XML Schema Definition*), conforme Figura 1.1. O modelo de processo de negócio modelado a partir de uma notação é chamado de Diagrama de Processo de Negócio (BPD, do inglês *Business Process Diagram*) (OMG, 2013). A atual versão é a 2.0.2, de 2013.

Figura 1.1 – Exemplo de descrição de elemento notacional da BPMN

Descrição Textual	Each *token* arriving at any incoming Sequence Flows activates the gateway and is routed to exactly one of the outgoing Sequence Flows. In order to determine the outgoing Sequence Flows that receives the *token*, the conditions are evaluated in order. The first condition that evaluates to true determines the Sequence Flow the *token* is sent to. No more conditions are henceforth evaluated.
Representação Gráfica	 Ou

XML Schema	`<xsd:element name="exclusiveGateway" type="tExclusiveGateway" substitutionGroup="flowElement"/>`
	`<xsd:complexType name="tExclusiveGateway">`
	`<xsd:complexContent>`
	`<xsd:extension base="tGateway">`
	`<xsd:attribute name="default" type="xsd:IDREF" use="optional"/>`
	`</xsd:extension>`
	`</xsd:complexContent>`
	`</xsd:complexType>`

Fonte: adaptado de (SANTOS; THOM; FANTINATO, 2015, p. 330)

Trabalhos iniciais demonstram que desde a BPMN 1.x, a notação não apresenta uma definição semântica clara, nem um formato nativo de serialização (CHINOSI; TROMBETTA, 2012). A situação é recorrente na versão 2.0 (CORREIA; ABREU, 2012), (LYAZIDI; MOULINE, 2013). Na versão 2.0.2 foi realizada a formalização da semântica dos elementos notacionais, definindo estruturas XSD para cada elemento (OMG, 2013).

No trabalho de Santos, Thom e Fantinato (2015), foi realizado um estudo sobre a falta de sincronismo entra a regra textual e o respectivo XSD de um elemento notacional. Alguns exemplos, identificados no trabalho dos referidos autores, na qual a regra não está sendo aplicada no XSD incluem:

- Um fluxo de mensagem deve conectar elementos de diferentes piscinas.
- Um evento de fim não pode apresentar nenhum fluxo de sequência na sua saída.
- Uma atividade de *loop* deve ser executada enquanto a condição para se manter o *loop* for avaliada como verdadeira.
- Em um desvio inclusivo, o *token* deve ser encaminhado para cada fluxo de sequência que tenha a sua condição avaliada como verdadeira.

Santos, Thom e Fantinato (2015) também propõem o incremento do XML Schema dos elementos notacionais da BPMN a partir da definição de regras de extração de XML. A partir desse XML extraído, o mesmo é adicionado na codificação original, tornando-o mais completo em relação as regras textuais.

O XML extraído no trabalho de Santos, Thom e Fantinato (2015) é realizado de forma manual. Tento essa premissa como base, o presente trabalho apresenta seguinte hipótese:

- Pode-se gerar código XML a partir da descrição dos elementos notacionais da BPMN, em linguagem natural (texto livre)?

1.2 Objetivos

O objetivo geral deste trabalho é gerar código XML a partir da descrição textual da especificação dos seguintes elementos notacionais da BPMN: desvio exclusivo (XOR), desvio paralelo (AND) e desvio inclusivo (OR).

Os objetivos específicos são:

- Desenvolver um protótipo para geração de código a partir de uma descrição em linguagem natural de elementos notacionais específicos.
- Gerar código XML seguindo o formato especificado pela BPMN a partir da descrição textual dos desvios, apresentada em OMG (2013).
- Gerar código XML utilizando a descrição da semântica dos elementos notacionais, definida em OMG (2013), juntamente com as regras de extração propostas em Santos, Thom e Fantinato (2015).

1.3 Contribuições

As principais contribuições deste trabalho são:

- Uso de estrutura Sujeito-Verbo-Objeto como apoio a interpretação de frases.
- Mapeamento entre semântica de estruturas Sujeito-Verbo-Objeto para código XML.
- Um protótipo que demonstra a possibilidade de extração de código XML a partir de uma descrição textual em linguagem natural, tal como a BPMN.

1.4 Organização do Trabalho

O capítulo 2 apresenta alguns trabalhos relacionados a geração de código. O capítulo 3 apresenta os fundamentos teóricos essenciais para o entendimento deste trabalho. No capítulo 4 é apresentada a metodologia utilizada e a implementação do protótipo. O capitulo 5 apresenta dados e testes relacionados a validação do protótipo. O capítulo 6 apresenta as conclusões e perspectivas de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Saeki, Horai e Enomoto (1989) propõem um método para se derivar uma especificação formal a partir de uma especificação informal onde serão identificadas as partes compreensíveis as pessoas e as partes compreensíveis as máquinas. Duas fases são realizadas para a obtenção desse objetivo. Na primeira, denominada fase de *design*, é construído um "documento de concepção de módulo" a partir da especificação informal, através da extração e classificação dos substantivos e verbos, e definição de suas relações. Esse documento apresenta uma estrutura modular de uma especificação formal, com informações de nomes de classes, métodos e mensagens de protocolos.

Na segunda fase, denominada de elaboração, é realizada a refinação da especificação informal de acordo com o documento de concepção de módulo. O resultado ainda é uma descrição informal em linguagem natural, contudo, um pouco mais detalhada e estruturada. Nessa especificação é aplicado o método de *design* obtendo um novo documento de concepção de módulo. O ciclo *design*-elaboração se repete até ser esclarecido o significado de todas as palavras da especificação informal, resultando numa especificação formal semelhante a um pseudocódigo. Devido às limitações de Processamento de Linguagem Natural (NLP do inglês *Natural Language Processing*) na época da realização da pesquisa, a classificação das palavras é realizada de forma manual.

Pandita et al. (2012) propõem uma abordagem para inferir especificações de métodos de classes a partir da descrição em linguagem natural em uma API (*Application Programming Interface*), com o objetivo de facilitar a verificação do código de um cliente que utiliza uma biblioteca da API em relação a descrição da biblioteca presente na API. O trabalho de Pandita et al. (2012) é realizado na linguagem C# .NET, mas, como são analisados documentos em linguagem natural, as especificações podem ser reusadas independente da linguagem de programação da biblioteca.

Na realização do trabalho referido, o texto é analisado por um *parser* em busca de descrições de argumentos, valores de retorno, exceções e informações adicionais sobre o método. O resultado é preprocessado para tratar a ambiguidade, equivalência semântica e palavras chaves da linguagem de programação, além da sintaxe própria em uma linguagem de programação que não correspondem a funções normais de caracteres de texto, como o ".", através de tabelas e listas geradas manualmente contendo estas informações. Após o pré-processamento é realizada a análise do texto para formação de expressões lógicas de primeira ordem. Um pós-processamento é realizado sobre as expressões lógicas para remover modificadores irrelevantes de predicados, classificar os predicados em uma classe semântica baseada nos dicionários do domínio e aumentar as informações obtidas na sentença quando possível. Por último, as sentenças pós-processadas são utilizadas como entrada para o gerador de contrato de código.

Carvalho et al. (2014) propõem a geração de testes de caso utilizando especificações SCR (*Software Cost Reduction*) como um formalismo intermediário, isto é, são extraídas dos textos de entrada especificações SCR, e estas são utilizadas para geração dos casos de teste. O foco do trabalho é a aviação industrial, então os requerimentos dos testes de caso são escritos em uma linguagem natural controlada desenvolvida pelos autores, a SysReq-CNL, e validada pela Embraer. Uma linguagem natural controlada é um subgrupo de uma linguagem existente que utiliza um conjunto restrito de regras gramaticais e um vocabulário de domínio de aplicação com um léxico predefinido (CARVALHO et al., 2014 apud ATLEE; GANNON, 1993).

A abordagem inicia com a análise sintática de uma requisição escrita de acordo com a SysReq-CNL. Após, uma análise semântica é realizada, baseada na teoria de gramática dos casos. Nessa teoria os verbos são classificados de acordo com o quadro de construção casual (*case frame*) em que podem ser inseridos. Com base na representação semântica, as especificações SCR são geradas, tornado-as independentes das regras sintáticas da SysReq-CNL. As especificações SCR são utilizadas como entrada para a T-VEC, uma ferramenta para geração automática de testes de caso a partir das especificações. O trabalho descreve que a aplicação atingiu 100% de precisão no aspecto que todos os dados gerados se apresentavam nos documentos manuais gerados por especialista, contudo gerou apenas 85% dos dados em relação a esses documentos.

Lei et al. (2013) tratam do problema da tradução de uma especificação de entradas de um programa, escrita em linguagem natural, para um código executável que apresente as estruturas de dados correspondentes aos dados de entrada apresentados no texto. A abordagem do problema é formulada através de dois passos. No primeiro é inferida uma "árvore de especificação" a partir da especificação em linguagem natural. Para isso, a especificação é dividida em frases nominais com o uso do *UIUC-shallow parser*. Essas frases são classificadas como "frases chave", onde pode haver informações relevantes a geração do código, ou "frases de fundo", que são frases utilizadas para organizar o documento ou se referir a "frases chave" anterioras. A partir das "frases chave" são extraidas as "árvores de especificação". Modelos generativos Bayesianos são treinados para realizar as ações necessárias para gerar as "árvores de especificação".

No segundo passo, as "árvores de especificação" são traduzidas para uma gramática usando um conjunto de regras e palavras-chave que identificam tipos básicos como **int**. Um *parser top-down* é gerado para a partir dessa gramática para formar as estruturas de dados. Como essa gramática é relativamente simples, uma abordagem padrão como Yacc poderia gerar um *parser bottom-up*. Foram utilizados 106 problemas da *ACM International Collegiate Programming Contests* para o trinamento dos modelos e testes de execução. Ao todo, o programa foi executado 20 vezes para cada texto e atingiu uma precisão de 89,3% de acertos na geração de código.

No Capítulo 4 são apresentados os aspectos similares e ideias influenciadas pelos trabalhos aqui apresentados dentro de seus respectivos contextos.

3 FUNDAMENTOS SOBRE GERENCIAMENTO DE PROCESSOS DE NEGÓCIOS E EXTRAÇÃO DE CÓDIGO

Este capítulo apresenta os fundamentos teóricos necessários à compreensão deste trabalho: Gerenciamento de Processos de Negócio, Notação e Modelo de Processos de Negócio e Extração de Código a partir de Linguagem Natural.

3.1 Gerenciamento de Processos de Negócio

O Gerenciamento de Processos de Negócio (BPM) é uma disciplina que inclui um conjunto de métodos, técnicas e ferramentas para as atividades de identificação, descoberta, análise, redesenho, implementação, monitoramento e controle de processos de negócios (DUMAS et al., 2013), (WESKE, 2012) e (von ROSING; SCHEER; von SCHELL, 2015). Através desse conjunto de atividades, BPM visa auxiliar organizações a atingir eficiência e eficácia, através da análise e aprimoramento dos seus processos de negócio (FRIEDRICH; MENDLING; PUHLMANN, 2011). O ciclo de vida de BPM inclui as seguintes fases Figura 3.1 (DUMAS et al., 2013):

- Na fase de **identificação**, os processos relevantes a um problema de negócio da organização são identificados, delimitados e relacionados. O resultado da fase de identificação é uma arquitetura de processos nova ou atualizada que proporciona uma visão geral dos processos da organização e suas relações.

- Na fase de **descoberta**, o estado atual de cada processo é documentado, normalmente na forma de BPDs, chamados modelo de processo *as-is*. Existem diversas notações para modelagem de BPDs como EPC (*Event Process Chain*), diagramas de atividade da UML (*Unified Modeling Language*) e redes de Petri. Contudo, BPMN é um padrão para modelagem de processos, adotada pela maioria das ferramentas de BPM disponíveis (SANTOS; THOM; FANTINATO, 2015).

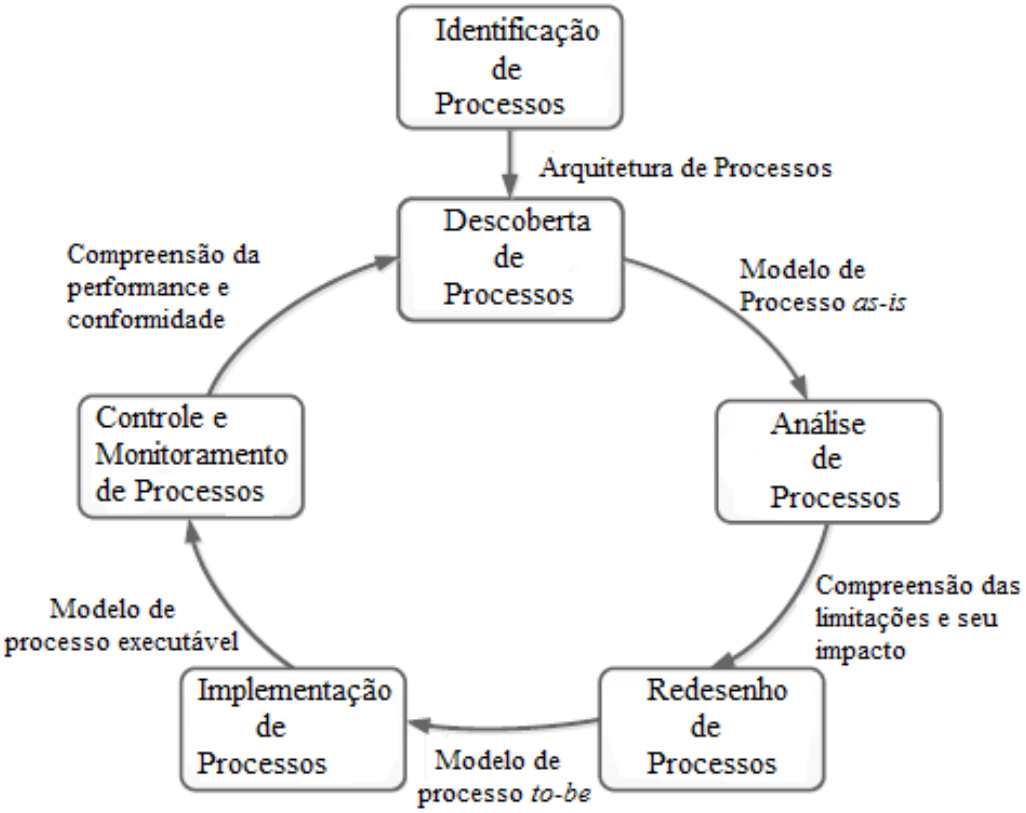
  As fases de identificação e descoberta são definidas em uma única fase de modelagem em von Rosing, Scheer e von Schell (2015).

- Na fase de **análise**, são identificadas as limitações associadas aos processos *as-is*, que são documentadas e, quando possível, levantadas medidas de performance. Como resultado têm-se uma coleção estruturada de limitações, na qual seus tratamentos devem ser priorizados de acordo com seu impacto no processo e, esporadicamente, em termos de esforço requerido para resolvê-los.

  Weske (2012) define que a identificação, descoberta e análise ocorram em uma única fase.

- Na fase de **redesenho** são definidas mudanças no processo que possam colaborar na solução de problemas identificados na fase de análise, de forma que possibilite à organização

Figura 3.1 – Ciclo de vida de BPM



Fonte: Adptado de (DUMAS et al., 2013)

atinja seus objetivos de performance. Então, o modelo de processo de negócio é ajustado e neste são aplicadas as mudanças necessárias para atender a realidade desejada, chegando ao modelo denominado *to-be*.

- Na fase de **implementação** são realizadas as modificações necessárias para que o processo possa ser implementado e automatizado na organização. Os aspectos abrangidos são: mudanças no gerenciamento da organização e automação dos processos. Mudanças no gerenciamento se referem ao conjunto de atividades requeridas para adaptar o modo de trabalho dos participantes ao processo de negócio. Automação refere-se ao desenvolvimento (ou aprimoramento) e aplicação de sistemas de informação que suportam a automação do processo.

  Tanto Weske (2012), como von Rosing, Scheer e von Schell (2015) definem as fases de redesenho e implementação como uma fase única.

- Na fase de **monitoramento e controle**, a partir do momento em que o processo inicia sua execução e têm-se várias instancias, os dados relevantes são analisados para determinar o

quanto o processo está de acordo com as medidas e objetivos de performance. Além disso, são identificadas outras limitações que possam ocorrer com as mudanças realizadas neste ou em processos relacionados.

Weske (2012) e von Rosing, Scheer e von Schell (2015) dividem o monitoramento e controle em duas fases distintas.

Após essas fases obtêm-se o processo melhorado, contudo, com o passar do tempo será necessária uma nova adaptação, realizando, novamente o ciclo de vida (DUMAS et al., 2013). Para isso, é importante que com os dados obtidos durante as execuções do processo possam novamente identificar, descobrir, analisar, redesenhar e implementar mudanças necessárias para que o processo sempre esteja alinhado com a organização.

3.2 Notação e Modelo de Processos de Negócio

A BPMN tem como objetivo prover uma notação que seja compreensível aos participantes de um negócio, desde o analista, passando pelos desenvolvedores responsáveis pela implementação do processo de negócio, até os participantes responsáveis por gerenciar e monitorar esses processos (OMG, 2013), (WESKE, 2012) e (CHINOSI; TROMBETTA, 2012).

BPMN possui um amplo conjunto de elementos notacionais, apresentando mais de 100 diferentes (DUMAS et al., 2013), (CORREIA; ABREU, 2012), visando suportar diversos níveis de abstração do processo, desde o nível de negócios até o nível de implementação (WESKE, 2012).

Devido à elevada quantidade de elementos notacionais, a BPMN pode ser considerada uma linguagem complexa (DUMAS et al., 2013), mas para ser compreendida pelos diversos participantes de um processo e ao mesmo tempo ter a capacidade de reprensentar os diversos níveis de abstração, os elementos notacionais foram divididos em cinco categorias específicas. Dentro de cada categoria, existem variações dos elementos básicos, que adicionam funcionalidades aos elementos notacionais, de forma a suportar requisistos mais complexos (OMG, 2013). Como demonstra a Figura 3.2, os desvios são representados por um losango, mas eles podem apresentar símbolos diferentes dentro do losango para especificar comportamentos diferentes. As cinco categorias básicas de elementos são:

- Objetos de Fluxo: são os principais elementos gráficos que definem o comportamento de um processo de negócio.
- Dados: São dados produzidos, consumidos, armazenados e/ou necessários ao processo.
- Conectores: são utilizados para definir a ordem dos acontecimentos em um processo, mensagens entre os participantes, associação dos dados, ou indicar a qual elemento estão associadas as informações adicionais representadas no diagrama.

- Divisões: representam os participantes do processo.
- Artefatos: sao utilizados para se adicionar informações sobre o processo.

A Figura 3.2 mostra os elementos notacionais que são objeto de estudo deste trabalho, esses foram escolhidos por serem considerados os mais problemáticos em termos de compreensão do usuário (SANTOS; THOM; FANTINATO, 2015). A Figura 3.3 mostra os elementos adicionais usados no exemplo de modelo.

Figura 3.2 – Desvios

 desvio exclusivo	Desvios exclusivos avaliam as condições dos fluxos de sequência de saída e enviam o *token* no caminho do primeiro fluxo verdadeiro. Para os fluxos de entrada, o *token* continua o processo logo que chega ao desvio
 desvio paralelo	Desvios paralelos enviam *tokens* por todos os fluxos de saída. Um desvio paralelo espera que cheguem *tokens* de todos os seus fluxos de entrada, e só depois prosseguir com um processo.
 desvio inclusivo	Desvios inclusivos avaliam todas as condições de seus fluxos de saída e enviam um *token* para cada caminho que apresentar uma condição verdadeira. Desvios inclusivos sabem quantos *tokens* devem esperar antes a partir de quantos fluxos de entrada foram ativados anteriormente

Fonte: Adaptado de (OMG, 2013)

3.2.1 Exemplo de Modelo de Processo em BPMN

A Figura 3.4 apresenta um modelo BPMN de um processo de matrícula em uma universidade fictícia do ponto de vista de um candidato. O processo inicia com o candidato sendo aceito pela universidade. Ao ser aceito, o candidato tem duas opções, levar os documentos necessários para a matrícula presencialmente até a secretaria ou enviar por meio online. Após entregar os documentos, o candidato deve efetuar o pagamento da matrícula. Com o pagamento efetuado, o candidato deve escolher as disciplinas que deseja cursar, e, por norma da universidade, participar de um clube como atividade extracurricular. Como a inscrição no clube e a escolha de disciplinas são atividades independentes, elas podem ser realizadas em qualquer ordem. Após inscrito em um clube e escolhido as disciplinas, a matrícula do candidato é efetivada e o processo termina.

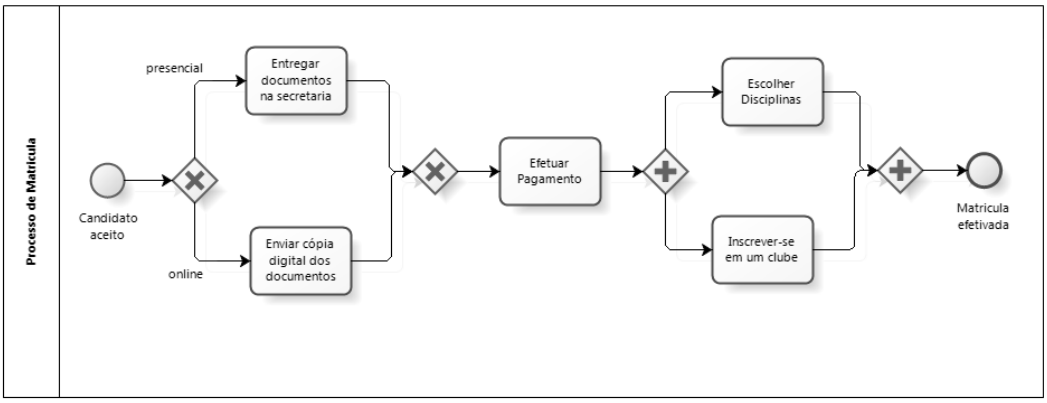
São consideradas boas práticas na criação de modelos BPMN a nomeação de atividades no padrão verbo-objeto, apresentar apenas um evento de fim, e que desvios sejam apenas

Figura 3.3 – Elementos básicos usados no modelo exemplo

Evento de inicio	Eventos de inicio são "acontecimentos" que geram uma nova instância do processo.
Evento de fim	Eventos de fim são "acontecimentos" que indicam o termino do processo.
Atividade Tarefa	Atividades representam trabalhos realizados no processo. Existem diversos tipos de atividade, dntre eles a tarefa indica o menor nível de abstração no modelo (Não pode ser dividida em outras tarefas)
Fluxo de sequência	Fluxos de sequência indicam a ordem em que os objetos de fluxo (eventos, atividades e desvios) ocorrem dentro do processo.

Fonte: Adaptado de (OMG, 2013)

Figura 3.4 – Processo de matrícula modelado em BPMN



Fonte: Modelado na ferramenta Bizagi

divergentes ou convergentes e apresentem pares de correspondência divergente-convergente (MENDLING; REIJERS; AALST, 2010), como apresentadas no processo da Figura 3.4.

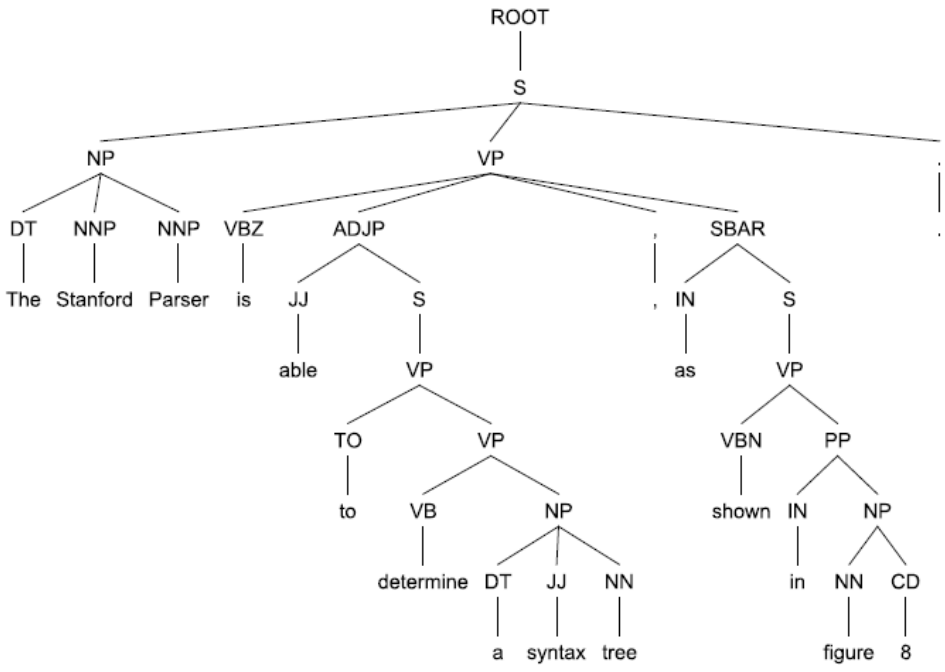
3.3 Extração de Código a partir de Linguagem Natural

Métodos na área de Linguística Computacional e Processamento de Linguagem Natural (NLP, do inglês *Natural Language Processing*) tentam analisar e extrair informações de textos e

discursos em linguagem natural (FRIEDRICH, 2010). Dois conceitos podem ser vistos como fundamentais na maioria dos trabalhos que utilizam técnicas de NLP são análise sintática e análise semântica (LEI et al., 2013), (FRIEDRICH; MENDLING; PUHLMANN, 2011), (PANDITA et al., 2012) e (CARVALHO et al., 2014).

A análise sintática trata da identificação morfo-sintática (POS, do inglês *part-of-speech*) das palavras de uma frase, e o reconhecimento da estrutura sintática, ou seja, como as palavras formam uma frase e suas respectivas relações (FRIEDRICH, 2010). Um *parser* de linguagem natural é um software que identifica as estruturas gramaticais de uma sentença. A anáçise sintática é realizada por ferramentas chamadas *parsers* (normalmente traduzido para analisador sintático, mas o uso de *parser* é mais frequente). Existem diversos *parsers* diferentes que realizam a identificação morfo-sintática em um texto como o *UIUC shallow parser* usado em Lei et al. (2013), o *Stanford Parser* usado em Pandita et al. (2012), Friedrich, Mendling e Puhlmann (2011), e outros como NLTK e o OpenNLP. As classificações e relações sintáticas normalmente são representadas pelas ferramentas na forma de árvores, denominada árvore sintática, como demonstra a Figura 3.5.

Figura 3.5 – Uma árvore sintatica gerada pelo *Stanford Dependency Parser*.



Fonte: (FRIEDRICH, 2010, p. 19)

Existem três categorias gerais de análise sintática em NLP: busca por palavras chave, afinidade lexical e métodos estatísticos (CAMBRIA; WHITE, 2014).

A busca por palavras chave classifica o texto em categorias baseadas na presença de

palavras sem ambiguidade. A principal limitação dessa técnica é que ela necessita da presença de palavras que podem ser óbvias, e por isso omitidas em um contexto, como uma busca pela palavra “carro” em uma revista automotiva, que pode nunca ser encontrada porque todos os carros são referidos através da marca e modelo.

A afinidade lexical designa uma “afinidade” probabilística por uma categoria particular a palavras arbitrárias. Apesar de superar a busca por palavras chave na maioria dos casos, as probabilidades das afinidades lexicais tendem a um gênero particular de texto, ditado pela fonte do corpora linguístico, o que pode dificultar o desenvolvimento de modelos reusáveis e independentes de domínio.

Métodos estatísticos dependem de modelos baseados em algoritmos de aprendizado de máquina, como estimativa por máxima verossimilhança, campos randômicos condicionais e máquinas de vetores de suporte. Com um grande volume de textos anotados como entrada para um algoritmo de aprendizado de máquina, é possível aprender a valência de palavras chave (como na abordagem de busca por palavras chaves), como também a valência de palavras arbitrárias (como na afinidade lexical), pontuação, e frequência da coocorrência das palavras. Por se tratarem de estimativas estatísticas, apresentam baixa precisão no nível de palavras individuais, sendo mais precisos a níveis de sentenças e superior.

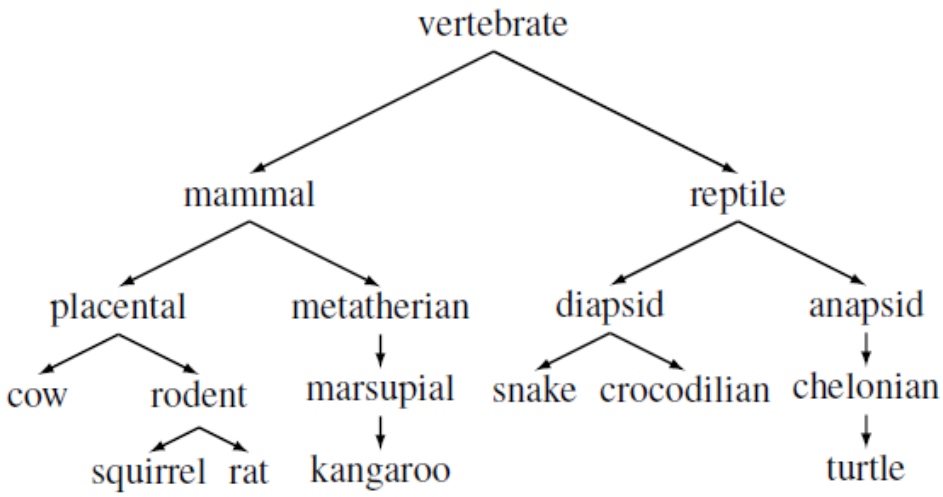
A análise semântica trata do significado das palavras. Exemplos de ferramentas que realizam análise semântica são o *WordNet*, que apresenta uma base de dados com o significado de mais de 150.000 palavras, e o *FrameNet*, que associa o significado das palavras a um contexto, possuindo mais de 135.000 frases para elaboração desses significados (FRIEDRICH, 2010). A Figura 3.6 demonstra um exemplo de análise semântica através de uma árvore taxonômica.

Abordagens NLP baseadas em semântica podem ser, de maneira generalizada, agrupadas em duas categorias: técnicas que utilizam conhecimento externo, como NLP taxonômica e NLP noética, ou métodos que exploram a semântica intrinsica de documentos (NLP endógena) (CAMBRIA; WHITE, 2014).

NLP endógena envolve o uso de técnicas de aprendizado de máquina para executar a análise semântica de um corpus através da construção de estruturas que aproximam conceitos a partir de um grande volume de documentos, ou seja, ele depende apenas do conhecimento endógeno destes documentos. As vantagens dessa abordagem são a efetividade, economia em termos de habilidades humanas, e portabilidade para outros domínios.

NLP taxonômica inclui iniciativas que visam construir taxonomias universais ou ontologias na Rede para compreender a hierarquia semântica associada a expressões em linguagem natural. Essas taxonomias geralmente consistem de conceitos (ex., pintor), instâncias (ex., "Leonardo da Vinci"), atributos e valores (ex., "A data de nascimento de Leonardo é 15 de abril de 1452), e relações (ex., "Mona Lisa foi pintada por Leonardo"). De modo geral, há diversas tentativas de construir recursos taxonômicos e incluem tanto recursos criados por especialistas humanos

Figura 3.6 – Parte de uma árvore taxonômica retirada do *WordNet*.



Fonte: (BANSAL et al., 2014, p. 1)

ou esforços de comunidades, como o *WordNet* e o *Freebase*, quanto bases de conhecimento geradas automaticamente.

NLP noética envolve as abordagens em NLP que visam compensar a falta de adaptividade de domínio e e inferência semântica implicita de algoritmos tradicionais. Essa abordagem visa coletar conhecimento idiossincrático sobre objetos, ações, eventos e pessoas. Além disso, é capaz de gerar resultados dependentes de contesto e descobrir novos padrões semânticos que não estão explicitados na base de dados.

Não existe um consenso ou padrão na forma como os métodos de análise sintática e análise semântica são aplicados para geração de código. Como podemos ver na Capítulo 2, cada autor tem um método especifico voltado para seu objetivo. Mas de modo geral, os rótulos obtidos na análise sintática serão usados para extrair os elementos com a propriedade desejada, e através dos significados obtidos na análise semântica destes, um codigo será "traduzido"de acordo com regras especificadas por cada abordagem. O capítulo a seguir demonstra a metodologia deste trabalho, com a forma como essas técnicas foram aplicadas para geração de código proposto neste trabalho.

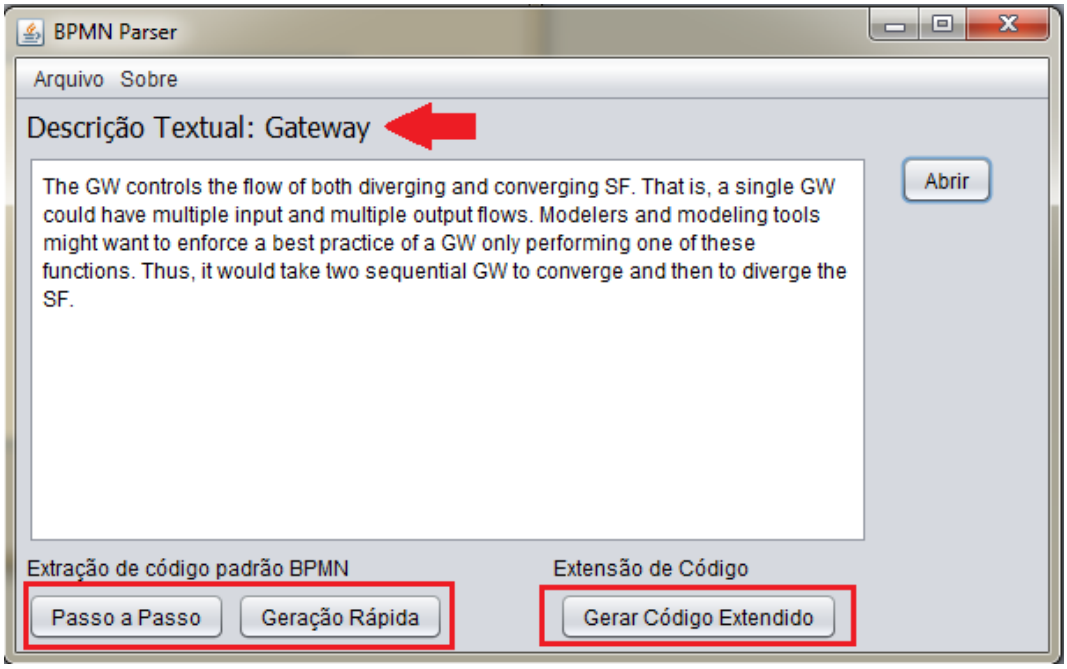
4 GERAÇÃO DE CÓDIGO XML A PARTIR DE ELEMENTOS NOTACIONAIS DA BPMN

Este capítulo apresenta a metodologia utilizada para a realização deste trabalho: *parser* a ser utilizado, estrutura da sentença, análise semântica e geração de código. Descreve também a implementação do protótipo criado para geração de código.

O protótipo segue os seguintes passos para geração de código: análise sintática da descrição textual com uso de um *parser*, extração de estruturas SVO e SVC a partir da análise sintática, análise semântica das SVOs e SVCs e "tradução" das mesmas para código XML.

A primeira etapa para geração de código é obter a descrição textual presente em OMG (2013) sobre um dos elementos notacionais de interesse, como demonstra a Figura 4.1.

Figura 4.1 – Demonstração do Protótipo – Arquivo Aberto



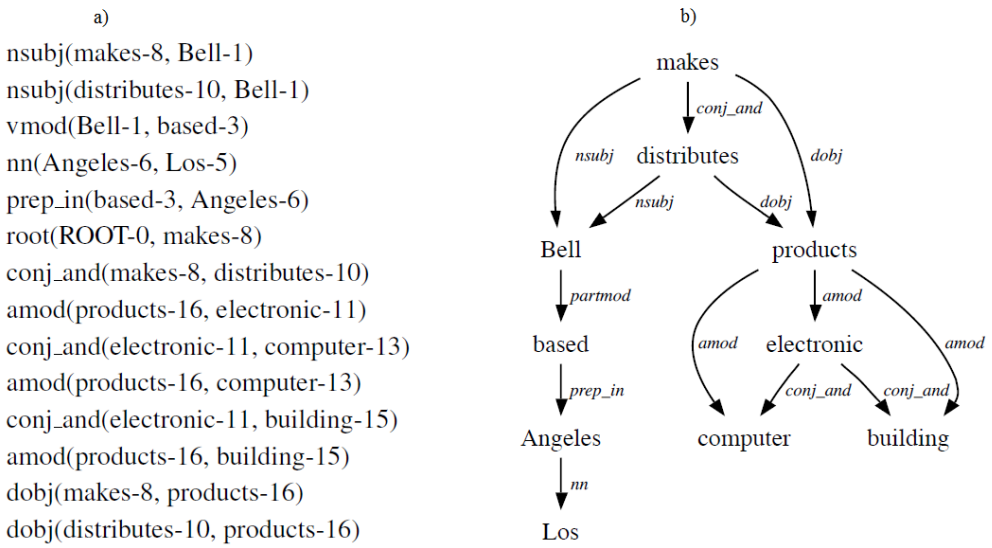
Descrição textual do elemento *gateway* apresentado na área de texto.

4.1 Stanford Parser

Parsers (ver seção 3.3) probabilísticos usam o conhecimento linguístico adquirido de sentenças analisadas manualmente para produzir a análise estatisticamente mais significativa para novas sentenças (MARNEFFE; MACCARTNEY; MANNING, 2006). O *Stanford Parser*, além de realizar a identificação morfo-sintática como outros *parsers* probabilísticos, também produz as *typed dependencies*, o que o levou a ser escolhido na implementação deste trabalho. As

typed dependencies foram criadas para prover uma descrição simples das relações gramaticais em uma senteça de forma a serem compreendidas sem um conhecimento linguístico de alto nível(MARNEFFE; MANNING, 2015). Enquanto a análise sintática de estruturas de frase representam o aninhamento de constituintes com multíplas palavras, uma análise de dependências representa relações entre palavras individuais (MARNEFFE; MACCARTNEY; MANNING, 2006). A Figura 4.2 mostra um exemplo das dependências resultantes da análise de um frase.

Figura 4.2 – *Typed Dependencies* adquiridas do *parsing* de uma frase.



Análise da frase "*Bell, based in Los Angeles, makes and distributes electronic, computer and building products.*". Na figura a) tem-se um dos tipos de saída gerada pelo *Stanfod Parser* (apresentado neste exemplo por ser a utilizada neste trabalho). Na figura b) tem-se uma representação gráfica das dependências. Fonte: adaptado de (MARNEFFE; MANNING, 2015, p. 1-2)

O primeiro elemento da relação gramatical é chamado de governante (gov) enquanto que o segundo elemento é chamado dependente (dep). Essas relações gramaticais fornecem uma camada de abstração para a árvore sintática pura e contêm informações sobre a função sintática de todos os elementos (FRIEDRICH, 2010). Marneffe e Manning (2015) definem uma lista com as 55 relações de dependências em uma sentença. Neste trabalho foram utilizadas as seguintes relações para formação de estruturas Sujeito-Verbo-Objeto (SVO):

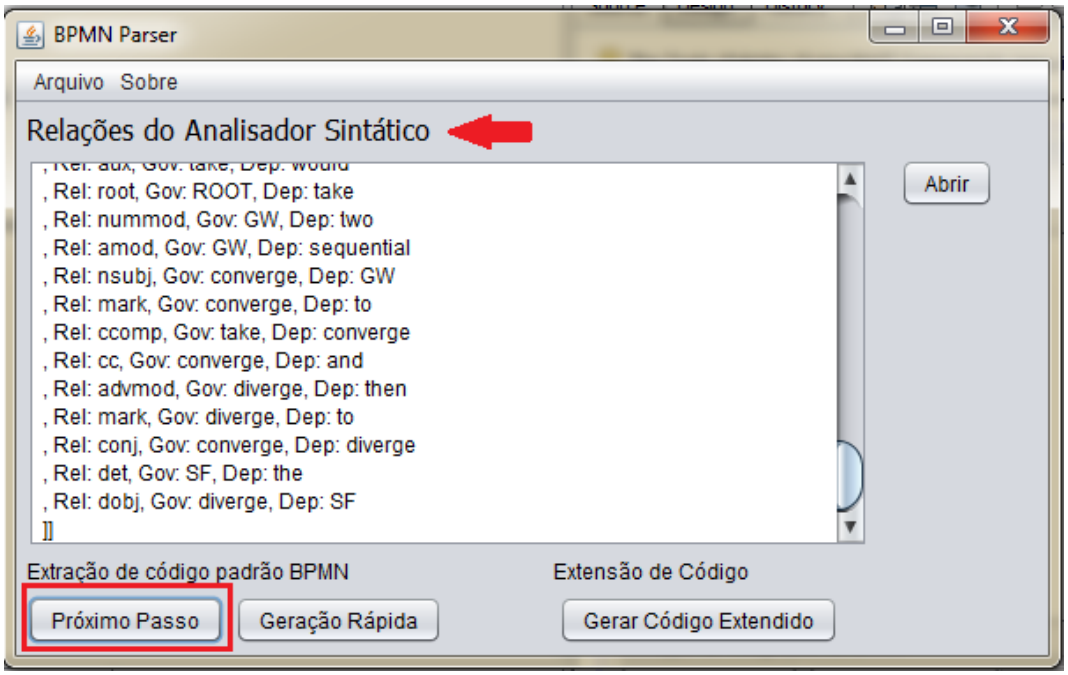
- **advcl**: Relação entre uma frase adverbial que modifica uma frase verbal ou sujeito.
- **advmod**: Relação entre um advérbio, ou frase iniciada por um advérbio, que modifica o significado de uma palavra.
- **amod**: Frase adjetiva que modifica o significado de uma frase nominal.
- **aux**: Relação de verbos secundários com o verbo principal.

- **auxpass**: Relação de um verbo principal com um verbo secundário que mantém a informação passiva da frase.
- **case**: Relação usada para qualquer preposição em inglês. Em inglês, conjunções subordinadas introduzindo frases normalmente são preposições.
- **cc**: Relação entre um elemento da conjunção com a própria conjunção de coordenação. Normalmente o primeiro elemento da conjunção é o governador da relação.
- **conj**: Relação entre dois elementos conectados por uma conjunção de coordenação como *and* ou *or*.
- **compound**: Frases ou palavras associadas por uma conjunção. Quando existem vários compostos, todos são associados à palavra mais a direita da frase.
- **det**: Relação entre o elemento principal de uma frase nominal e seu determinante.
- **dobj**: Frase nominal que é o objeto direto do verbo.
- **iobj**: Substantivo principal de uma frase nominal que é o objeto indireto do verbo.
- **mwe (*multi-word expression*)**: relação usada entre palavras múltiplas que se comportam como uma única palavra. Ex: "*in case*".
- **neg**: Relação entre uma palavra de negação e a palavra que ela modifica.
- **nmod**: Frase nominal que modifica o significado de outra frase nominal.
- **nsubj**: Sujeito sintático ativo de uma frase nominal.
- **nsubjpass**: Sujeito sintático passivo de uma frase nominal.
- **rcmod**: Relação de uma frase relativa que modifica uma frase nominal. A relação aponta do substantivo principal da frase nominal para o elemento principal da frase relativa, normalmente um verbo.

Nas seções a seguir apresenta-se como essas relações são utilizadas para obtenção das estruturas definidas para extração de código.

A Figura 4.3 demonstra as dependências extraídas da descrição utilizada como resultado da análise sintática.

Figura 4.3 – Demonstração do Protótipo – Análise Sintática



Dependências extraídas da descrição utilizada como resultado da análise sintática.

4.2 Estrutura Sujeito-Verbo-Objeto

A extração de estruturas Sujeito-Verbo-Objeto (SVO) é utilizada porque uma frase em inglês é usualmente composta por estes três elementos (JAROSIEWICZ, 2003). Diferente do português, o inglês não apresenta sujeito oculto, o que facilita a extração deste tipo de estrutura.

Como em Yang et al. (2011), neste trabalho, são procuradas relações sujeito-verbo através de dependências **nsubj** e relações verbo-objeto através de dependências **dobj**, mas além dessas também são identificados sujeitos passivos através de **nsubjpass**, e objetos indiretos através de **iobj**. Esta primeira extração constitui as palavras base das SVOs. Após a identificação desses elementos, os modificadores associados a cada um são extraídos através de relações **amod**, **compound**, **det** e **rcmod** para sujeitos e objetos, **aux** e **auxpass** para verbos, e **advmod** e **neg** para os três.

Com exceção da relação **compound**, os modificadores obtidos são adicionados ao elemento associado (sujeito, verbo ou objeto), tornando o elemento em uma lista de palavras, com a palavra base no início. A relação **compound** apresenta a associação de várias palavras através de conjunções, então, para cada uma é gerada uma estrutura SVO. Além disso, de acordo com a conjunção encontrada nas relações **conj** ou **cc**, uma nova SVO pode ser criada com um elemento base composto.

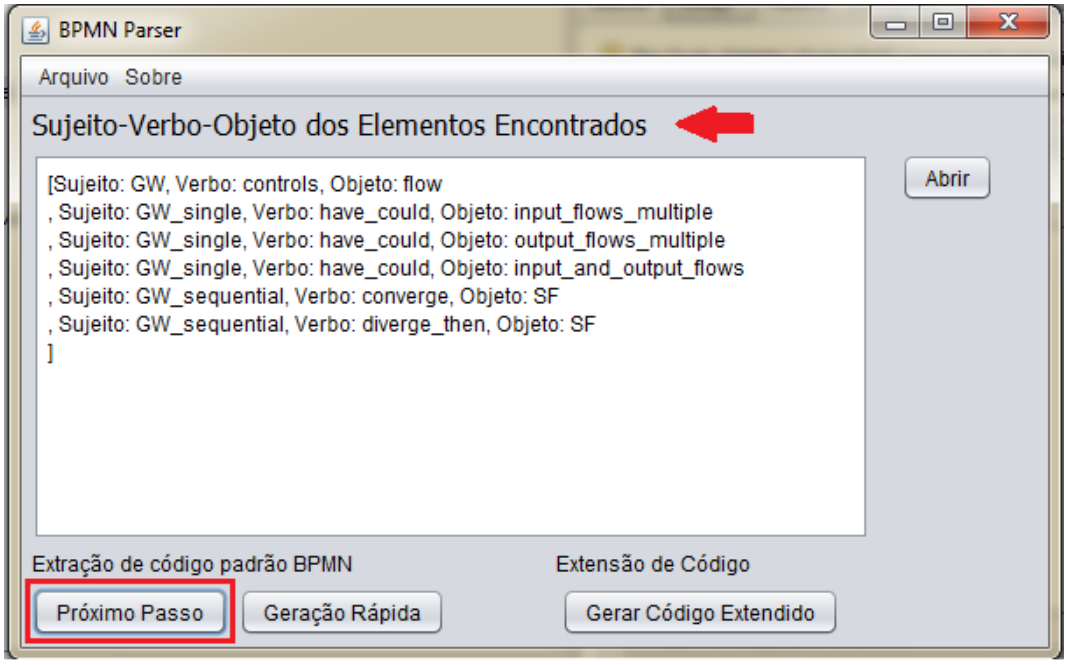
Aqui têm-se um exemplo do uso dessa estrutura com a frase: "*a single gateway could have*

multiple input and multiple output flows.". Este trabalho extrai como base : Sujeito – *gateway*, Verbo – *have*, Objeto – *input* e Objeto – *output*. Por se tratar de objetos ligados por uma conjunção *and*, um terceiro objeto composto, Objeto – *input_and_output*, é gerado. Neste caso, extraímos da frase três SVOs base (S: *gateway*, V: *have*, O: *input*; S: *gateway*, V: *have*, O: *output*; S: *gateway*, V: *have*, O: *input_and_output*). Depois são extraídos os modificadores e como resultado teremos: S: *gateway_single*, V: *have_could*, O: *input_flows_multiple*; S: *gateway_single*, V: *have_could*, O: *output_flows_multiple*; S: *gateway_single*, V: *have_could*, O: *input_and_output_flows_multiple*.

O trabalho de Yang et al. (2011) demonstra a força de expressividade das estruturas SVOs utilizando-as para formação de descrição de vídeos, isto é, a partir de ações identificadas no vídeo são extraídas estruturas SVO e uma frase descritiva é formada tomando-a como base, por exemplo, identificando-se <*person, ride, horse*> pode-se formar a descrição "*A person is riding a horse.*". Krishnamoorthy et al. (2013) Usa uma abordagem semelhante, mas com uma quádrupla (sujeito, verbo, cena, preposição), para descrever o que é realizado e em que local a partir da imagem. A utilização do componente "cena" em Krishnamoorthy et al. (2013) é semelhante a função que "objeto" apresenta na análise semântica deste trabalho (neste trabalho o interesse é saber quem "faz" o que a quem, e para eles quem faz o que, onde).

A Figura 4.4 apresenta as SVOs (apresentadas na área de texto) obtidas a partir das relações sintáticas encontradas no texto.

Figura 4.4 – Demonstração do Protótipo – SVOs



SVOs obtidas através das dependências encontradas.

4.2.1 Estrutura Sujeito-Verbo-Caso

A estrutura Sujeito-Verbo-Caso (SVC) foi criada para extrair informações de frases subordinadas que indicam uma "condição" em relação a outra frase. Elas são utilizadas para geração do codigo extendido proposto em Santos, Thom e Fantinato (2015). São assim chamadas porque são primariamente identificadas através da dependência **case**. As estruturas SVC são colocadas em conjunto com estruturas SVO das frases ao qual são subordinadas. Outra dependência importante na formação desse conjunto é **advcl**, uma vez que os modificadores adverbiais desta relação expressam consequência, condição, propósito, entre outros, e através dela podemos adquirir os verbos relacionados ao qual precisamos extrair a SVO.

Aqui têm-se um exemplo do uso dessa estrutura com a frase: "*If and only if none of the conditions evaluates to true, the token is passed on the default sequence flow*.". Encontra-se aqui a relação **advcl**(*passed*, *evaluates*) que indica que "a passagem" é uma consequência da "avaliação". Também obteremos a relação **case(*none*, *if*)**, que como é um caso de interesse será extraída junto com seus modificadores (que é realizado da mesma maneira das SVOs). No caso, o resultado final desta etapa da extração para essa frase vai resultar em: (S: *conditions_only_none*, V: *evaluates*, C: *true* – S: *token*, V: *passed*, O: *[sequence flow]_default*) (Obs: *sequence flow* está entre colchetes pois é realizado um pré-processamento que o reduz a uma palavra, como será explicado na seção 4.5).

4.3 Análise Semântica

A análise semântica das SVOs e SVCs se utiliza de métodos de NLP taxonômica, e para isso, neste trabalho foi criada manualmente uma tabela de significados para os elementos notacionais, verbos e modificadores encontrados no texto. Por se tratar de um vocabulário de domínio específico, muitas palavras apresentam um significado próprio no contexto do texto que não pode ser adquirido por bases de dados tradicionais. Um exemplo é a palavra *path*, que, de modo geral, também significa "caminho" para um processo, mas que deve ser interpretada como sinônimo de *sequence flow* neste contexto de domínio, o que não vai ser obtido por bancos de dados genéricos. Carvalho et al. (2014) também definem novas semânticas aos verbos através da criação de mais "papéis temáticos" em sua gramática dos casos para melhor se adaptar ao seu contexto especifico da aviação.

Parte da análise semântica ocorre junto com a formação de SVOs e SVCs, uma vez que só são formadas estruturas que apresentam um sujeito, objeto ou caso de interesse (no caso, elementos que podem ser encontrados na tabela semântica). Pandita et al. (2012) também mantém uma lista de palavras para casos especiais em sua implementação, mas que no caso são removidas da solução.

Assim como em Lei et al. (2013), onde certos verbos como *contains* e *consists* são bons indicadores de "frases chave", alguns verbos são definidos como indicadores para a criação de

atributos, como *control* e *have*.

Abaixo temos a representação de algumas palavras na tabela semântica:

(control, create)	(controls, create)
(identified, check)	(identify, check)
(have, check)	(has, check)
(input, incoming flow)	(output, outgoing flow)
(diverge, multiple outgoing flow)	(diverging, multiple outgoing flow)
(converge, multiple incoming flow)	(converging, multiple incoming flow)
(could, optional)	(optionally, optional)

Pode-se ver que mesmo os verbos indicadores não apresentam a mesma semântica. Neste trabalho, "*control*" possui uma indicação mais forte da possibilidade de um novo atributo do que o verbo indicador"*have*". Apesar da tabela usar a semântica "*create*", ainda assim são analisadas se a semântica das outras SVOs encontradas apresentam realmente valores ao atributo indicado, e se nenhuma correspondência for encontrada, o atributo é descartado.

A análise semântica não produz uma estrutura própria como a análise sintática. Nesta fase diversas comparações são feitas com os significados de cada SVO e SVC extraídas do texto ao mesmo tempo em que são interpretadas pelo gerador de código.

4.4 Geração de Código

Como discutido na seção 3.3, cada autor apresenta um método próprio para geração de código de acordo com seus propósitos. O gerador de Pandita et al. (2012) utiliza um mapeamento predefinido de classes semânticas dos predicados para construções de programação na produção de contratos de código válidos. Lei et al. (2013) geram um *parser top-down* a partir da gramática extraída por ser uma gramática simples. Carvalho et al. (2014) determinam as variáveis e seus tipos através dos "papéis" *Patient* e *Condition Patient*.

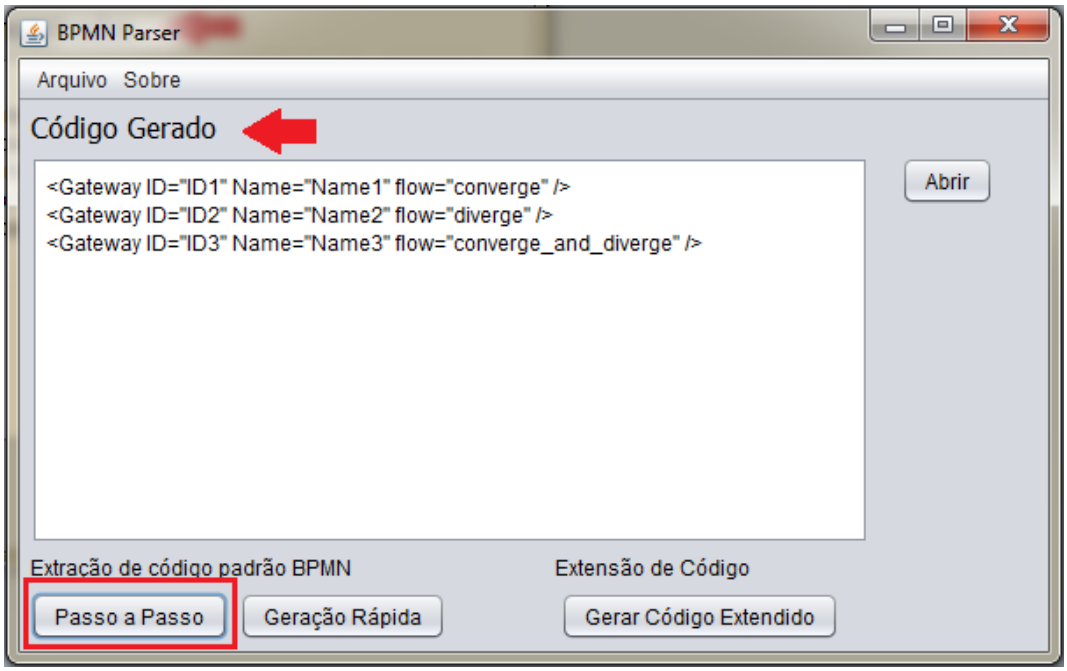
Neste trabalho, a geração de código é centrada na busca de ações e relações dos desvios com fluxos de sequência (uma vez que a função principal dos desvios é definir o comportamento de seus fluxos de entrada e saída).

Aqui têm-se um exemplo de uma interpretação das SVOs, S: *gateway_single*, V: *have_could*, O: *input_flows_multiple*; e, S: *gateway*, V: *converge*, O: *flow*. De maneira geral, a primeira SVO pode ser interpretada como "*one gateway optionally have multiple incoming sequence flows*" que é semânticamente equivalente a "*gateway converge flow*". Estas podem ser, por sua vez, relacionadas a SVOS: *gateway*, V: *controls*, O: *flow*. O verbo *controls* é um dos indicadores de um possível atributo já que "controlar" tem o sentido de "definir comportamento", junto

com o objeto *flow*, que é um sinônimo reduzido para fluxo de sequência, um dos elementos de interesse, podemos ver que um dos modos de *control flow* é *converge*. Com essas informações podemos extrair o código XML *<Gateway ID="GW1" name="Gateway1" flow="converging" />* (ID e *name* são atributos herdados de acordo com a especificação da BPMN (OMG, 2013)). As comparações não são sempre tão diretas como nesse exemplo, mas, de modo geral, são procuradas frases semanticamente equivalentes e as relações entre essas frases com outras que apresentam verbos indicadores.

Na Figura 4.5 é possível ver o resultado final da geração código a partir da análise semântica dos elementos das SVOs.

Figura 4.5 – Demonstração do Protótipo – Código Gerado



Resultado final geração de código.

Nestes exemplos foi utilizada a descrição textual de *gateway*, que apesar de ser um elemento abstrato, a geração de seus atributos é necessária pois os outros desvios herdam os mesmos.

4.5 Implementação do Protótipo

O protótipo desenvolvido para extração de código XML da descrição textual dos elementos notacionais selecionados (AND, OR, XOR) foi implementado na linguagem JAVA. JAVA é uma linguagem puramente orientada a objetos, de modo que se aplicados os conceitos de orientação a objetos, facilita a criação de um código coeso e modular facilitando a manutenção,

extensibilidade e reuso de classes implementadas. O *Stanford Parser* possui uma API para JAVA, o que influenciou na sua escolha.

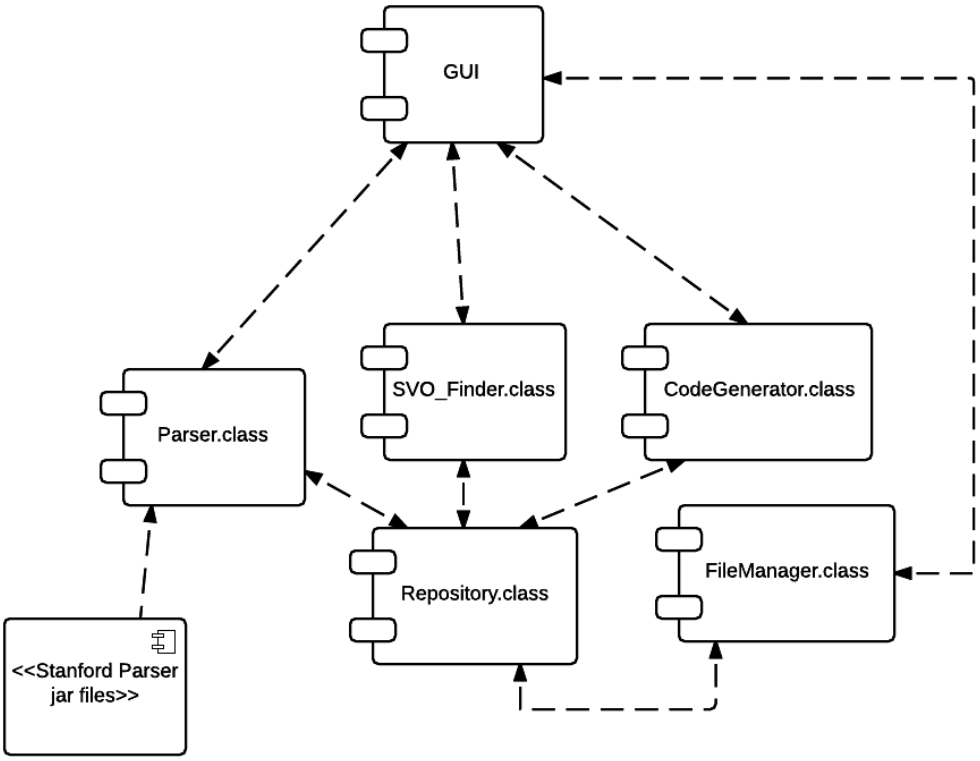
Duas estruturas foram criadas para comportar os dados básicos utilizados no programa. A **struct Relation**, e a **struct SVO**.

A **struct Relation** apresenta os dados de uma relação de dependência: uma **String** para a relação (**rel**), uma **String** para o governante (**gov**) e uma **String** para o dependente (**dep**). As dependências de uma frase são colocadas em uma lista de **Relation** (é usada a interface **List** para que novos componentes não necessitem usar sempre um tipo específico que implementa **List**). As dependências são armazenadas em um lista própria para que o programa não seja dependente do *Stanford Parser*, e qualquer *parser* que possa extrair dependências do texto possa ser utilizado.

A **struct SVO** possui uma **String** para sujeito (**subj**), uma **String** para o verbo (**verb**) e uma **String** para o objeto (**obj**).

O programa é organizado como demonstra a Figura 4.6.

Figura 4.6 – Diagrama de Componentes do protótipo implementado



A classe **Repository** armazena todos as estruturas de dados utilizadas pelos demais componentes e recebe as estruturas geradas pelos mesmos.

A classe **FileManager** lê os arquivos textos e retorna uma **String** com o conteúdo do arquivo (posteriormente será estendida para gravar arquivos com os resultados).

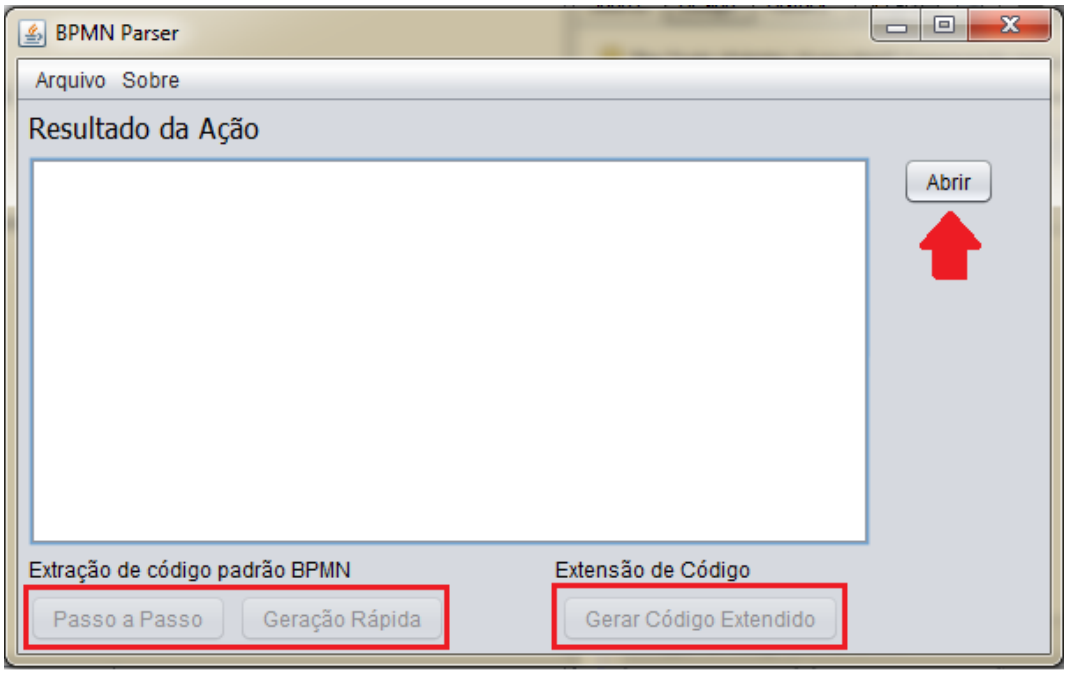
A classe **Parser** utiliza a API do *Stanford Parser* para a análise sintática do texto e retorna uma **List<List<Relation»**, onde cada **List<Relation>** representa as relações de dependência de uma frase do texto.

A classe **SVO_Finder** faz duas passagens pelas dependências encontradas, primeiro são adquiridos os elementos principais do sujeito, do verbo e do objeto, e depois seus modificadores, como explicado na seção 4.2. A classe **SVO_Finder** se utiliza da tabela semântica para filtrar os elementos de interesse, eliminando SVOs que não apresentem sujeitos de interesse. **SVO_Finder** retorna uma **List<SVO>** com todas as SVOs de interesse encontrados.

A classe **CodeGenerator** utiliza a **List<SVO>** e a tabela semântica para geração do código XML como explicado na

Como mostrado na Figura 4.6, os componentes se comunicam apenas com o repositório e a interface gráfica (GUI). A GUI possui uma comunicação em duas vias porque os componentes "avisam" quando terminaram suas funções e disponibilizaram a estrutura de dados gerada, habilitando a função da GUI que disponibiliza o uso do componente sem que este tente utilizar de recursos inexistentes. Como pode-se ver pela Figura 4.7, os botões para geração de código estão desabilitados enquanto não há nenhum arquivo aberto para que não se tenha uma exceção de arquivo inexistente, com o único botão habilitado sendo o botão "Abrir".

Figura 4.7 – Demonstração do Protótipo – Tela Inicial



Botões desabilitados enquanto não há dados para serem utilizados.

5 VALIDAÇÃO

Para validação da solução proposta, os resultados obtidos foram comparados com a codificação especificada em OMG (2013).

A Figura 5.1 mostra os atributos definidos pela OMG (2013) para o elemento abstrato *gateway*. A Figura 5.2 demonstra que os atributos obtidos pela solução proposta foram muito próximos dos definidos na especificação. Os atributos "ID" e "name" são herdados de classes mais genéricas, e por isso não aparecem na Figura 5.1, que só apresenta os atributos próprios de *gateway*. O atributo encontrado "*flow*" é equivalente ao atributo definido "*gatewayDirection*", e os três valores encontrados para o atributo (*converge, diverge e converge_and_diverge* existem na especificação (*converging, diverging e mixed*, respectivamente). O valor "*unspecified*" (Figura 5.1) presente no atributo "*gatewayDirection*" na definição da BPMN não foi obtido porque os outros três valores já contemplam todas as possibilidades descritas no texto (apenas multíplas entradas, apenas multíplas saídas, multíplas entradas e saídas) e, em nenhum momento, é explicitamente descrito que não é necessário definir como um *gateway* controla o fluxo.

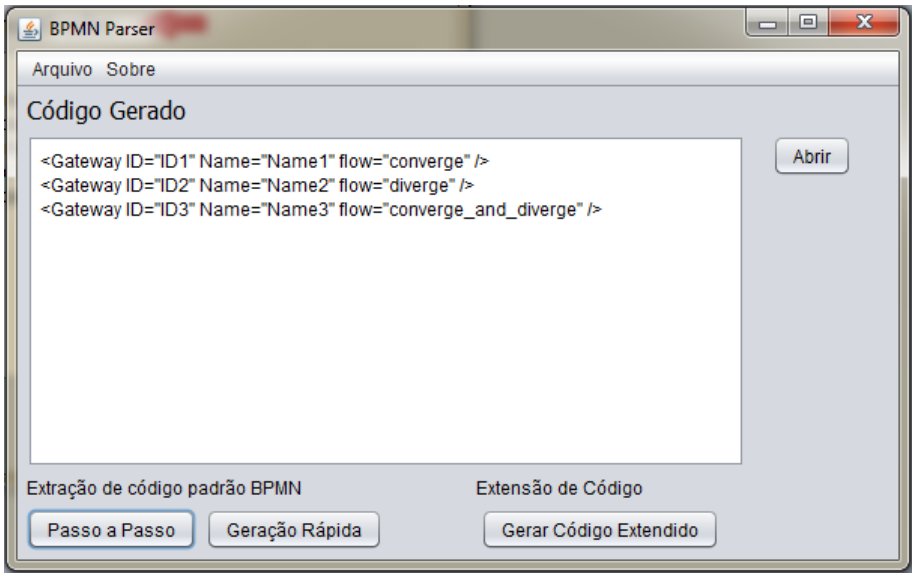
Figura 5.1 – Atributos de *Gateway* – De acordo com a especificação da BPMN

Nome do Atributo	Descrição/Uso
gatewayDirection: GatewayDirection = Unspecified { Unspecified | Converging | Diverging | Mixed }	An attribute that adds constraints on how the **Gateway** MAY be used. • **Unspecified**: There are no constraints. The **Gateway** MAY have any number of *incoming* and *outgoing* **Sequence Flows**. • **Converging**: This **Gateway** MAY have multiple *incoming* **Sequence Flows** but MUST have no more than one (1) *outgoing* **Sequence Flow**. • **Diverging**: This **Gateway** MAY have multiple *outgoing* **Sequence Flows** but MUST have no more than one (1) *incoming* **Sequence Flow**. • **Mixed**: This **Gateway** contains multiple *outgoing* and multiple *incoming* **Sequence Flows**.

Atributos de *gateway*. Adaptado de (OMG, 2013)

A Figura 5.3 apresenta o atributo adicional presente em *inclusive gateways* e *exclusive gateways*. Como percebesse pela Figura 5.4 e pela Figura 5.5, o atributo foi encontrado como de acordo com a especificação (aqui denominado "*defaultFlow*" ao invés de apenas *"default"*). As figuras mostram todas as permutações dos atributos do elemento abstrato pai "*gateway*" com o novo atributo encontrado, demonstrando sua característica de opcionalidade (os atributos presentes em "*defaultFlow*" são os mesmos especificados para o elemento notacional *sequence*

Figura 5.2 – Atributos de *Gateway* – Obtidos pela solução proposta



Resultado da primeira etapa de geração de código.

flow e são predefinidos em nesta implementação porque a extração de código do mesmo não é contemplada neste trabalho).

Figura 5.3 – Atributos de *Exclusive (Inclusive) Gateway* – De acordo com a especificação da BPMN

Nome do Atributo	Descrição/Uso
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such `Expression` SHALL be ignored.

Atributos de *exclusive (inclusive) gateway*. Adaptado de (OMG, 2013)

Por não apresentar nenhum atributo próprio, os resultados obtidos para o *parallel gateway* são iguais aos obtidos para o elemento pai, como demonstra a comparação das figuras 5.2 e 5.6.

5.1 Extensão do Protótipo e Aplicabilidade em outras Notações

Como descrito no capítulo anterior, o protótipo foi projetado de modo a ser extensível e reusável. As formas como ele pode ser extendido para outras notações ou linguagens de programação são:

- Criação de outras tabelas semânticas que contemplem elementos da notação desejada.

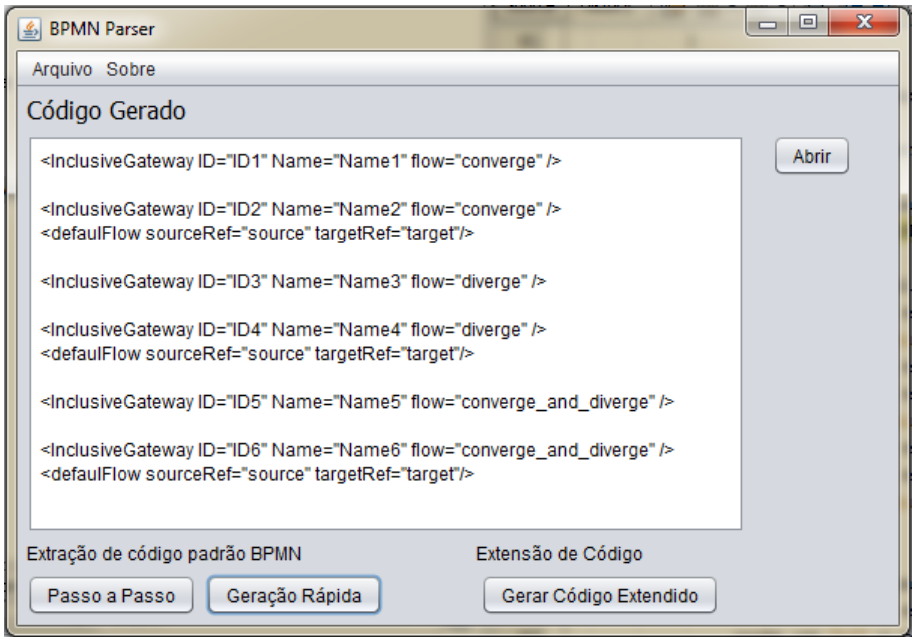
Figura 5.4 – Atributos de *Exclusive Gateway* – Obtidos pela solução proposta

Atributos de *exclusive gateway* obtidos pela de geração de código da solução proposta.

- Implementação do componente de geração de código para a linguagem de programação desejada.

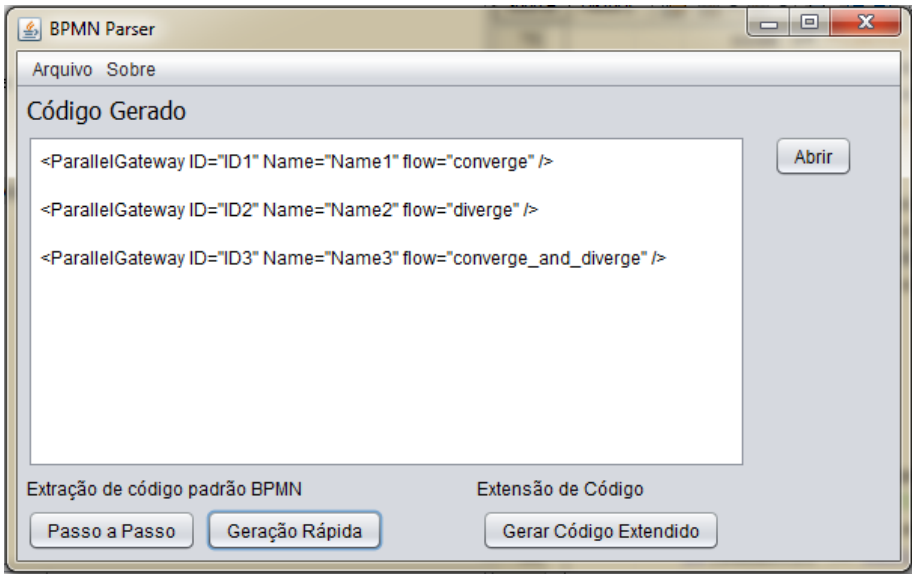
Uma das grandes vantagens da arquitetura de repositório aplicada neste *software* é justamente a modularidade. Como o repositório é uma "base de dados" e é o único ponto com o qual os outros componentes interagem, a adição de novos componentes ou dados não interfere nos componentes implementados e nem altera dados já incluídos no repositório, podendo-se utilizar todas as funcionalidades já disponíveis sem nenhuma alteração.

Figura 5.5 – Atributos de *Inclusive Gateway* – Obtidos pela solução proposta



Atributos de *inlusive gateway* obtidos pela de geração de código da solução proposta.

Figura 5.6 – Atributos de *Parallel Gateway* – Obtidos pela solução proposta



Atributos de *parallel gateway* obtidos pela de geração de código da solução proposta.

6 CONCLUSÃO

Neste trabalho foi apresentado um método para extração de de código XML a partir da descrição textual de elementos notacionais da BPMN. Para atingir este objetivo, técnicas de NLP tanto para análise sintática, quanto para análise semântica foram combinadas para a obtenção de estruturas SVO do texto e a respectiva interpretação das mesmas a um código XML.

Como analisados nos capítulos anteriores, foram obtidos resultados semelhantes com a extração de código em relação aos atributos e valores definidos na especificação da BPMN.

As principais contribuições deste trabalho são:

- Uma metodologia pra extração de código XML a partir de textos em linguagem natural.
- Apresentação de uma nova estrutura (SVC) como base para análise das relações semânticas entre frases.
- Um protótipo para extração de código a partir de linguagem natural.

Apesar do protótipo apresentado extrair apenas código XML associado a elementos notacionais da BPMN, a forma como foi implementado permite a adição de componentes para estender a outras linguagens de programação e/ou outras notações sem a necessidade de alteração dos componentes existentes. Para isso, basta acrescentar ao repositório dados relevantes a linguagem/notação desejada e implementar novos componentes que utilizem esses dados do repositório, podendo também usar todos os dados existentes ou produzidos pelos componentes já presentes no protótipo.

Algumas das limitações que podem ser encontradas na solução proposta são:

- Por se tratar de uma aplicação com a necessidade de interpretação de palavras em um domínio especifico implementado com uma semântica própria, a reusabilidade em outras notações não apresenta uma portabilidade direta.
- No seu estágio atual, o protótipo não apresenta a interpretação semântica de todos os elementos notacionais da BPMN, podendo gerar resultados errôneos quando aplicado a outros elementos que não os apresentados neste trabalho.
- Esta solução não trata da análise de anáforas (interpretação de pronomes), uma vez que apesar de descrita em linguagem natural, por ser uma especificação de uma notação, a BPMN apresenta um certo grau de formalidade, sendo os elementos notacionais sempre referidos explicitamente para diminuir possíveis ambiguidades.

- Pelo fato da interpretação semântica estar integrada junto à classe de geração de código XML, ela não pode ser diretamente aplicada para a geração de código em outras linguagens (Diversas funções dentro da classe apenas tratam da análise semântica, então elas poderiam ser reutilizadas em uma nova classe).

Trabalhos futuros incluem alguns pontos que podem ser melhorados e extensões possíveis como:

- Inclusão da semântica de mais elementos notacionais da BPMN, para extração de código dos mesmos.
- Integração com bases de conhecimentos como *WordNet* ou *Freebase*, para aumentar a amplitude do domínio da extração de código.
- Extração de código para outras linguagens além de XML.
- Suporte a validação das regras de extração propostas em Santos, Thom e Fantinato (2015).

REFERÊNCIAS

BANSAL, M. et al. Structured learning for taxonomy induction with belief propagation. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*. [s.n.], 2014. p. 1041–1051. Disponível em: <http://aclweb.org/anthology/P/P14/P14-1098.pdf>. Citado na página 22.

CAMBRIA, E.; WHITE, B. Jumping NLP curves: A review of natural language processing research. *IEEE Computational Inteligence Magazine*, v. 9, n. 2, p. 48–57, 2014. Disponível em: <http://dx.doi.org/10.1109/MCI.2014.2307227>. Citado 2 vezes nas páginas 20 e 21.

CARVALHO, G. et al. Nat2test$_{\mathrm{SCR}}$: Test case generation from natural language requirements based on SCR specifications. *Sci. Comput. Program.*, v. 95, p. 275–297, 2014. Disponível em: <http://dx.doi.org/10.1016/j.scico.2014.06.007>. Citado 4 vezes nas páginas 14, 20, 28 e 29.

CHINOSI, M.; TROMBETTA, A. BPMN: an introduction to the standard. *Computer Standards & Interfaces*, v. 34, n. 1, p. 124–134, 2012. Disponível em: <http://dx.doi.org/10.1016/j.csi.2011.06.002>. Citado 2 vezes nas páginas 11 e 17.

CORREIA, A.; ABREU, F. B. e. Adding preciseness to bpmn models. *Procedia Technology*, v. 5, p. 407–417, 2012. Citado 2 vezes nas páginas 11 e 17.

DUMAS, M. et al. *Fundamentals of Business Process Management*. Springer, 2013. ISBN 978-3-642-33142-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-33143-5>. Citado 4 vezes nas páginas 10, 15, 16 e 17.

FRIEDRICH, F. *Automated Generation of Business Process Models*. Dissertação (Master of Science in Information Systems) — School of Business and Economics, Humboldt-Universität zu Berlin, Berlim, Alemanha, nov 2010. Citado 3 vezes nas páginas 20, 21 e 24.

FRIEDRICH, F.; MENDLING, J.; PUHLMANN, F. Process model generation from natural language text. In: *Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011, London, UK, June 20-24, 2011. Proceedings*. [s.n.], 2011. p. 482–496. Disponível em: <http://dx.doi.org/10.1007/978-3-642-21640-4_36>. Citado 2 vezes nas páginas 15 e 20.

JAROSIEWICZ, E. *Natural Language Parsing and Representation in XML*. Dissertação (Master of Science) — Graduate School of the University of Florida, 2003. Citado na página 26.

KRISHNAMOORTHY, N. et al. Generating natural-language video descriptions using text-mined knowledge. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. [s.n.], 2013. Disponível em: <http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6454>. Citado na página 27.

LEI, T. et al. From natural language specifications to program input parsers. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. [s.n.], 2013. p. 1294–1303. Disponível em: <http://aclweb.org/anthology/P/P13/P13-1127.pdf>. Citado 4 vezes nas páginas 14, 20, 28 e 29.

LYAZIDI, A.; MOULINE, S. Formal verification of bpmn models using petri nets. In: *Maroc 2013 - The 1st International Workshop on Models and Algorithms for Reliable and Open Computing*. Tetouan, Morocco: [s.n.], 2013. Citado na página 11.

MARNEFFE, M.-C. de; MACCARTNEY, B.; MANNING, C. D. Generating typed dependency parses from phrase structure parses. In: *LREC 2006 - The fifth international conference on Language Resources and Evaluation, 22-28 may*. Genoa, Italia: [s.n.], 2006. Citado 2 vezes nas páginas 23 e 24.

MARNEFFE, M.-C. de; MANNING, C. D. *Stanford Typed Dependencies Manual*. Stanford, CA 94305-9010, 2015. Disponível em: <http://nlp.stanford.edu/software/dependencies_manual.pdf>. Citado na página 24.

MENDLING, J.; REIJERS, H. A.; AALST, W. M. P. van der. Seven process modeling guidelines (7PMG). *Information & Software Technology*, v. 52, n. 2, p. 127–136, 2010. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2009.08.004>. Citado na página 19.

OBJECT MANAGEMENT GROUP. *Business Process Model and Notation (BPMN) v2.0.2*. [S.l.], 2013. Citado 10 vezes nas páginas 10, 11, 12, 17, 18, 19, 23, 30, 33 e 34.

PANDITA, R. et al. Inferring method specifications from natural language API descriptions. In: *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. [s.n.], 2012. p. 815–825. Disponível em: <http://dx.doi.org/10.1109/ICSE.2012.6227137>. Citado 4 vezes nas páginas 13, 20, 28 e 29.

SAEKI, M.; HORAI, H.; ENOMOTO, H. Software development process from natural language specification. In: *Proceedings of the 11th International Conference on Software Engineering, Pittsburg, PA, USA, May 15-18, 1989*. [s.n.], 1989. p. 64–73. Disponível em: <http://doi.acm.org/10.1145/74587.74594>. Citado na página 13.

SANTOS, C. F. H. dos; THOM, L. H.; FANTINATO, M. Investigating completeness of coding in business process model and notation. In: *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 3, Barcelona, Spain, 27-30 April, 2015*. [s.n.], 2015. p. 328–333. Disponível em: <http://dx.doi.org/10.5220/0005464603280333>. Citado 6 vezes nas páginas 11, 12, 15, 18, 28 e 38.

von ROSING, M.; SCHEER, A.-W.; von SCHELL, H. *Complete Business Process Handbook*. first. 225 Wyman Street, Waltham, MA 02451, USA: Morgan Kaufmann, 2015. I. Citado 4 vezes nas páginas 10, 15, 16 e 17.

WESKE, M. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012. ISBN 978-3-642-28615-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28616-2>. Citado 4 vezes nas páginas 10, 15, 16 e 17.

YANG, Y. et al. Corpus-guided sentence generation of natural images. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*. [s.n.], 2011. p. 444–454. Disponível em: <http://www.aclweb.org/anthology/D11-1041>. Citado 2 vezes nas páginas 26 e 27.