UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

ESCOLA DE INFORMÁTICA APLICADA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Um Framework Genérico para Geração de Texto em Linguagem Natural a partir de Modelos de Processo de Negócio

Autor:

Raphael de Almeida Rodrigues

Orientadores:

Leonardo Guerreiro Azevedo

Henrik Leopold

Geração de Texto em Linguagem Natural a partir de Modelos de Processo de Negócio em Português

Raphael de Almeida Rodrigues

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Rio de Janeiro - RJ

Abril/2013

Geração de Texto em Linguagem Natural a partir de Modelos de Processos de Negócio em Português

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

Prof. Leonardo Guerreiro Azevedo, D.Sc. (UNIRIO)

_____

Henrik Leopold, M.Sc. (Humboldt-Universiät zu Berlin)

_____

Prof. Kate Cerqueira Revoredo, D. Sc. (UNIRIO)

_____

Prof. Márcio de Oliveira Barros, D.Sc. (UNIRIO)

O autor deste Projeto autoriza a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, ____ de _____ de _____

_____

Raphael de Almeida Rodrigues

Agradecimentos

Agradeço primeiramente a minha família, em especial a minha mãe, que sempre me ajudou durante o meu desenvolvimento pessoal e profissional, me dando apoio e incentivo nos momentos mais difíceis e estressantes, possibilitando chegar nesse ponto da minha vida.

Agradeço ao meu orientador, Leonardo Azevedo, pela sua dedicação, paciência e capacidade que foram essenciais para o desenvolvimento desse trabalho, e pela sua enorme contribuição tanto na minha formação acadêmica quanto na minha formação profissional.

Agradeço ao Henrik Leopold pela brilhante palestra cujo tema e *framework* apresentados serviram como motivação e base para a formulação deste projeto e pelas inúmeras ajudas prestadas durante todo o período de elaboração desta monografia.

Agradeço o corpo docente da UNIRIO pelo conhecimento transmitindo sempre se esforçando para proporcionar o melhor aprendizado possível. Uma lembrança especial aos professores Alexandre Correa, Marcio Barros, Sean Siqueira e Gleison Santos por algumas das melhores aulas que já tive e por sempre extraírem o melhor dos seus alunos.

Agradeço aos meus colegas de graduação com os quais dividi momentos de felicidade e angústia durante quase cinco anos de estudo, em especial Amannda Misael, André Gouvêa, Juliana Carvalho, Bruna Christina e Marina Vinhaes.

Nunca considere o estudo como uma obrigação, mas sim como uma oportunidade para penetrar em um belo e maravilhoso mundo do saber.

(Albert Einstein).

SUMÁRIO

LISTA DE ABREVIATURAS

BPMN – Business Process Model and Notation (Notação de Modelo de Processos de Negócio)

DSyn – Deep-syntatic tree (Árvoré sintática de busca)

GLN - Geração de Texto em Linguagem Natural

NLG – Natural Language Generation

NLU – Natural Language Understanding (Entendimento de Linguagem Natural)

OMG – Object Management Group

RPST – Refined Process Structure Tree (Árvore de Processo Estruturada e Refinada)

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMO

Modelagem de processos é um conceito amplamente utilizado para entender, documentar e também redefinir as operações das organizações. A validação e uso de modelos de processos é, entretanto, afetada pelo fato de que estão voltados para o entendimento por analistas de negócio e não por especialistas do negócio. Em outras palavras, analistas de negócio têm facilidade para ler os modelos mas não entendem profundamente o negócio. Por outro lado, especialistas do negócio detêm profundo conhecimento sobre o domínio, mas em geral não detêm a técnica para leitura de modelos de processos de negócio. Para estes é mais simples a leitura do texto que representa o modelo de processo ao invés do próprio modelo. Portanto é importante ter tanto o processo de negócio modelado como também o texto que representa o processo para que tanto analistas como especialistas do negócio tenham facilidade para interagirem. Para contornar este problema, Leopold *et al.* [2012] propuseram um *framework* que transforma automaticamente modelos de processo desenhados com a notação BPMN em textos em linguagem natural em inglês. Este trabalho apresenta a generalização do framework elaborado por Leopold *et al.* [2012] para ser capaz de tratar novos idiomas para geração de texto em linguagem natural a partir de modelos de processos em BPMN. Esta generalização foi realizada através da abstração do código para criação de interfaces, cujas operações são apresentadas e documentadas. Duas avaliações foram realizadas no trabalho: avaliação da generalização do framework e avaliação do uso do framework. A primeira avaliação foi realizada através da implementação das extensões necessárias para tratar o idioma português. A segunda avaliação baseou-se na geração de texto a partir de um conjunto de modelos de processo BPMN escritos em português e inglês. Como resultado foi demonstrado que textos em linguagem natural podem ser gerados a partir de modelos de processo escritos em português e inglês de uma maneira confiável e de fácil compreensão.

Palavras-chave: Geração de linguagem natural, modelos de processo de negócio, *framework* para geração de texto a partir de modelos de processos de negócio, Geração de texto a partir a de modelos de processos de negócio.

Capítulo 1: INTRODUÇÃO

1.1 Contextualização e Motivação

Organizações possuem uma sequência, normalmente estruturada, de suas atividades, de modo a realizar uma determinada tarefa. Estas atividades devem ser executadas nas ordens certas e possuem papéis específicos, que são responsáveis por executá-las. Cada tarefa dentro de uma empresa pode ter inúmeras atividades e inúmeros responsáveis. Por este motivo existe a necessidade de uma documentação detalhada sobre a estrutura dos processos que a mesma executa. Segundo Davies *et al* [2004], esta documentação é necessária, sendo cada vez mais crescente em grandes empresas. Kautz *et al.* [2004] corrobora esta afirmação e a justifica pelo fato de grandes empresas possuírem projetos grandes e complexos, sendo necessária uma abordagem mais detalhada para suas atividades.

Um modelo é uma simplificação da realidade. Ele é construído para ajudar na compreensão das atividades, para ajudar na documentação e servem como guias [Larman, 2007]. Aproveitando-se das vantagens de um modelo, a modelagem de processos foi criada para ajudar na especificação e documentação dos processos organizacionais. Através destas características podemos destacar uma das principais importâncias de um modelo de processo: permitir uma visualização gráfica, sequencial e simples de um processo executado pela empresa, bem como documentar todas as atividades e as informações relacionadas com cada uma delas [Larman, 2007]. Por exemplo, atividades de um modelo de processo são executadas por atores. Em modelo, as atividades executadas por um mesmo ator ficam em um mesmo eixo ou raia (horizontal ou vertical), tornando fácil a visualização de todas as atividades a ele relacionadas dentro do processo especificado OMG [2006].

Apesar das vantagens de um modelo de processo, esta abordagem também possui algumas desvantagens. Dentre elas, a principal, é o conhecimento prévio da notação utilizada na modelagem, necessário para a correta compreensão do modelo [Ko *et al.*, 2008]. Os modelos são elaborados conforme um padrão de modelagem de processos especificado, mais usualmente o padrão BPMN [OMG, 2011]. Para que alguém possa, de fato, compreender todas as informações que o modelo tenta

representar, a pessoa que está visualizando-o deve entender todos os componentes presentes no modelo, como gateways, eventos, atores, desvios etc. Para este fim, muitas vezes se faz necessário o treinamento dos funcionários no padrão escolhido para representação de seus modelos de processo. Todo treinamento custa tempo e dinheiro, e ambos os recursos são essenciais para qualquer empresa.

Este projeto visa exatamente à solução deste problema, fazendo com que qualquer pessoa, mesmo sem um conhecimento prévio de modelagem de processos, possa compreender o cenário descrito pelo modelo. Tal compreensão é possível graças a geração de um texto, em linguagem natural, que descreve o cenário modelado. Esta geração de texto a partir do modelo é chamada de verbalização. Por utilizar a língua escrita, não é necessário conhecimento prévio sobre a notação utilizada para elaboração dos modelos de processo [Leopold *et al.*, 2012]. Um exemplo de cenário é sua utilização para validação de modelos, ajudando a diminuir a distância de diálogo entre especialistas do domínio e analistas de sistemas [Frederiks e Weide, 2004].

Leopold *et al.* [2012] elaboraram um framework para lidar com este problema. Através da combinação de diversos algoritmos de mineração de texto [Konchady, 2006] e outras técnicas, eles criaram uma ferramenta que, dado um modelo de processo elaborado no padrão BPMN, gera como saída um texto gramaticalmente correto e estruturado, de fácil compreensão para os usuários. Porém, o projeto foi elaborado e implementado exclusivamente para o inglês. Para lidar com as particularidades e com a morfologia da língua inglesa na citada aplicação, se fez necessário o uso de uma biblioteca gramatical digital, diretamente acessível e de fácil integração com o framework elaborado com a linguagem Java. Esta biblioteca é conhecida como WordNet [WordNet, 2013]. WordNet é um banco de dados léxico online para ser utilizado em conjunto com aplicações, desenvolvido e registrado sob os direitos autorais da universidade de Princeton. Esta biblioteca é capaz de fazer ligações entre adjetivos, advérbios, pronomes e substantivos em inglês para o conjunto de sinônimos, que por sua vez, estão interligados através de relações semânticas que determinam a definição de cada palavra [Miller, 1995]. Devido a esta característica, ela é capaz de qualificar qualquer palavra em inglês em sua respectiva categoria, como verbo, substantivo, adjetivo, etc. Além disso, possui outras categorizações como, por exemplo, identificar o tempo do verbo e seu modo.

O trabalho de Leopold *et al.* [2012] é base para este trabalho, sendo evoluído para lidar com a língua portuguesa. Tal proposta tem seu maior desafio no fato de não existir uma vasta biblioteca gramatical digital em português, como a WordNet para a língua inglesa. Até a conclusão deste trabalho, não foi possível encontrar uma biblioteca digital da língua portuguesa que atendesse aos seguintes requisitos: Ser totalmente aberta, disponível para download e modificável [Rademaker e Paiva, 2012]. Algumas abordagens alternativas (que não atendem a todos os requisitos citados) estão disponíveis, como a WordNet.PT [WordNet.PT, 2013] e a OpenWN-PT disponível no endereço: https://github.com/arademaker/wordnet-br. Dessa forma, para português é necessária uma abordagem capaz de contornar o problema da ausência de uma biblioteca como a WordNet. Consequentemente, há necessidade da criação de novos componentes para auxiliar no tratamento gramatical e morfológico do texto.

1.2 Objetivo do trabalho

Este trabalho tem o objetivo de generalizar o framework proposto por Leopold *et al.* [2012] para permitir a adição padronizada de novos idiomas para geração de texto em linguagem natural a partir de modelos de processos em BPMN. Além disso, este trabalho implementou as classes necessárias para tratar modelos de processos em português. Para este fim foram elaborados novos algoritmos para tratar e mapear as especificidades da gramática portuguesa.

1.3 Contribuições

Este trabalho tem as seguintes contribuições:

- Generalização do framework de Leopold *et al.* [2012] para tratar idiomas derivados do latim;
- Documentação detalhada do código (diagrama de classes e JavaDoc), da organização estrutural em módulos (diagrama de pacotes), dos algoritmos do framework de Leopold *et al.* [2012] e da sequência das operações executadas (diagrama de sequencia).
- Implementação das classes necessárias para geração de texto a partir de processos em português;

- Especificação em pseudocódigo dos principais algoritmos para esta monografia;
- Criação de um repositório semelhante ao Wordnet para obtenção de informações linguísticas a partir do corpus Floresta[1];
- Criação de heurísticas para tratar as características envolvidas com a gramática portuguesa como, por exemplo, mapeamento de plural para singular, definição dos artigos corretos para os substantivos, bem como seus respectivos gêneros.

1.4 Estrutura do trabalho

Este trabalho está dividido em seis capítulos. O Capítulo 1 corresponde à presente introdução. O Capítulo 2 apresenta os principais conceitos relacionados a este trabalho. O Capítulo 3 apresenta o processo para geração de texto em linguagem natural. O Capítulo 4 apresenta a proposta deste trabalho, apresentando detalhes de implementação, enquanto que o Capítulo 5 apresenta a avaliação da proposta. Finalmente, no Capítulo 6 são apresentadas as conclusões do trabalho e propostas de trabalho futuro.

[1] Floresta é um corpus (uma coleção grande ou completa de artigos ou escritas em geral) composta por cerca de 95.000 frases (cerca de 1.600.000 palavras) retiradas do início dos corpora **CETENFolha** (textos do jornal brasileiro Folha de São Paulo de 1994) e **CETEMPúblico** (diário português PÚBLICO, de 1991 a 1998). Todo o corpus foi analisado automaticamente pelo analisador sintático PALAVRAS [Floresta].

Capítulo 2: PRINCIPAIS CONCEITOS

Este capítulo define e apresenta de forma detalhada os conceitos que estão diretamente relacionados a este trabalho.

2.1 BPMN

Cada vez mais as organizações precisam de processos de negócios bem definidos e, ao mesmo tempo, flexíveis. Além disso, os sistemas de informação precisam acompanhar o progresso das organizações. Para este fim se faz necessária a utilização de uma notação padrão para a modelagem dos diversos processos executados pelas organizações.

Existem várias notações para modelagem de processos de negócio, dentre elas cabe destacar: BPEL (*Business Process Execution Language*) [WS-BPEL, 2007], UML AD (UML *Activity Diagrams*) [UML, 2012], BPDM (*Business Process Definition Metamodel*) [OMG, 2008], EPC (*Event-driven Process Chain*) [Scheer, 2000], BPQL (*Business Process Query Language*) [OMG 2008] e YAWL (*Yet Another Workflow Language*) [YWAL, 2007]. A escolhida para ser utilizada neste trabalho foi BPMN [OMG, 2011]. A escolha deste padrão se baseia na avaliação dos diversos padrões para a modelagem de processos realizado por Ko *et al.* [2008]. Os autores apontam dois padrões como sendo os principais da atualidade: o Diagrama de atividades da UML e BPMN. Segundo Ko *et al.* [2008]¸estes padrões são atualmente os dois mais expressivos, com um nível fácil de integração e execução e possivelmente os que mais influenciarão no futuro próximo. Uma das vantagens do BPMN sobre o Diagrama de Atividades da UML é a capacidade de mapeamento diretamente para código executável (por exemplo, BPEL). BMPN está crescentemente se consolidando como o padrão definitivo para modelagem de modelos de processo pelas organizações [Ko *et al*., 2008]. Por este motivo, BPMN foi escolhida como o padrão para confecção dos modelos utilizados pela ferramenta.

A Notação de Modelo de Processos de Negócio (BPMN) é uma linguagem padrão de modelagem de processos que provê uma notação gráfica para especificação de processos de negócio e tem como objetivo prover um conjunto de símbolos

específicos para modelagem de processos. A BPMN está atualmente na versão 2.0, publicada pela OMG [OMG, 2011]. A BPMN funciona como o meio-campo entre as áreas interessadas no planejamento estratégico da organização, diminuindo a distância entre o mapeamento de processos da organização e a implementação técnica destes processos. Para alcançar tal alinhamento, é necessário que a linguagem BPMN seja compreendida por analistas de negócio, técnicos e usuários. Além disso, é possível automatizar a interpretação de modelos de processos através de sistemas. Por exemplo, Leopold *et al.* [2012] utilizam uma interface que é capaz de ler um modelo e interpretá-lo extraindo as informações relevantes e desejadas, da mesma forma que uma pessoa faria.

Em geral, notações gráficas são mais fáceis para usuários do negócio (não técnicos) entenderem e utilizarem. Comparadas com os padrões de nível de execução como BPEL, padrões gráficos revelam, por exemplo, padrões dentro de um processo, falhas em ciclos e gargalos de um processo de negócio [Ko *et al.*, 2008].

Devido a sua notação gráfica, um diagrama desenhado com base na BPMN tende a ser compreendido mais facilmente. Porém, para isso é necessário que todos os envolvidos possuam um conhecimento do que se trata BPMN e do significado dos elementos utilizados em sua notação gráfica. Uma boa documentação da forma como a notação será utilizada, descrevendo os principais componentes BPMN e como eles devem ser utilizados. Além disso, exemplos de modelos, que podem ser utilizados como guias pelos desenvolvedores de modelos de processos, são importantes para um bom andamento do desenho dos processos.

O conjunto de símbolos de um modelo BPMN cobre quatro tipos de elementos: objetos de fluxo (atividades, eventos, caminhos), objetos de conexão (sequência e fluxo de mensagem, associações), raias (*pool* e *lanes*) e artefatos (objetos e anotações) [OMG, 2011].

A Tabela 1 apresenta um resumo dos principais componentes da BPMN utilizados neste trabalho.

A Figura 1 apresenta um exemplo de modelo de processo descrito em BPMN. O processo inclui quatro atores e as diversas atividades são agrupadas de acordo com seus respectivos atores responsáveis, em quatro diferentes raias. As atividades executadas pelos diferentes atores são exibidas como caixas com quinas

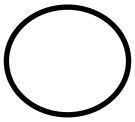
arredondadas. O caminho que possui uma forma de diamante (*gateway*) define a estratégia de roteamento das atividades. O sinal de mais (+) em um caminho especifica uma execução em paralelo, o círculo uma escolha inclusiva (um ou mais dos possíveis caminhos podem ser executados) e o xis (x) uma escolha exclusiva (somente um dos caminhos pode ser executado). O processo pode ser descrito da seguinte forma:

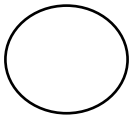
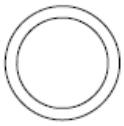
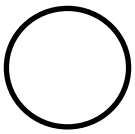
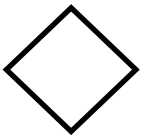
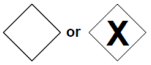
O processo começa quando o gerente de quarto recebe um pedido. Isso ativa três atividades paralelas:

- Caso bebidas alcoólicas sejam solicitadas, o gerente de serviço de quarto entrega o pedido para o *barman*. Concluída esta etapa, uma ou ambas atividades são executadas: o *barman* pega o vinho da adega e/ou o *barman* prepara as bebidas;
- Em paralelo a esta etapa, o gerente designa o pedido para o garçom. Subsequentemente o garçom prepara a nota fiscal;
- Além destas, o gerente submete um pedido para a cozinha. Neste momento, a cozinha começa a preparar a comida;

Uma vez que todas as três atividades paralelas tenham sido executadas, o garçom realiza a entrega do pedido no quarto do hóspede. Posteriormente, ele retorna para o serviço de quarto. Finalmente, o garçom faz o débito na conta do hóspede.

Tabela 1 – Resumo dos principais componentes BPMN utilizados nos modelos de processos de negócio elaborados [OMG, 2008].

Elemento	**Descrição**	**Símbolo**
Evento	Um evento é algo que 'acontece' durante um processo. Um evento afeta o percurso do processo e normalmente possui uma causa (gatilho) ou um impacto (resultado). Eventos são representados como círculos com centros abertos que permitem marcadores internos para diferenciar os diferentes tipos de resultados ou gatilhos. Existem três tipos de eventos, baseados no momento em que atuam sobre o percurso: Inicio, Intermediário e Fim.	
Tipo de evento: Início	Como o nome sugere, o evento inicial indica onde um processo começa.	*Início*

		
Tipo de Evento: Intermediário	Eventos intermediários ocorrem entre um evento de início e um evento de fim. Eles afetam o fluxo do processo, mas nunca irão começar ou terminar (diretamente) o processo.	
Tipo de Evento: Final	Como o nome sugere, o evento final indica onde um processo termina.	
Atividade	Uma atividade é um termo genérico para um trabalho executado em um processo. Uma atividade pode ser atômica ou não atômica. Os tipos de atividades que são parte de um modelo de processo são subprocessos e tarefas, que são retângulos com cantos arredondados.	
Gateway	Um *gateway* é usado para controlar a convergência e divergência de fluxos de sequência em um processo. Consequentemente, determina ramificações (divisões) e uniões dos diversos caminhos presentes no modelo. Marcadores internos indicam o tipo do gateway.	
Tipos de Gateways	Ícones dentro da forma de diamante do gateway indicam o tipo de comportamento de controle de fluxo. Os tipos de controles incluem: • Gateway para decisão exclusiva. Pode ser exibido com ou sem o marcador X. Este símbolo é utilizado para indicar que somente um dos próximos caminhos que estão conectados ao gateway, pode ser executado. Ao escolher um deles, os outros caminhos não poderão ser executados. A escolha de um dos caminhos é obrigatória.	*Gateway Exclusivo* 

	• Gateway de decisão inclusiva. Este símbolo é utilizado para indicar que um ou mais dos caminhos presentes no modelo podem ser executados. A escolha de ao menos um dos caminhos é obrigatória. • Gateway paralelo para união. Este símbolo é utilizado para indicar que todos os caminhos que estão conectados ao gateway serão iniciados ao mesmo momento. Todas as atividades serão iniciadas em paralelo.	*Gateway Inclusivo* *Gateway Paralelo*
Fluxo de sequência	É usado para mostrar a sequência com que as atividades serão executadas em um processo	
Pool	Representa um participante em um processo. Um participante pode ser uma **entidade de negócio** (exemplo: uma empresa) ou pode ser um **papel** (de negócio), como por exemplo: vendedor, comprador ou fabricante.	Nome
Lane	Lane é uma subdivisão dentro de um Pool usada para organizar e categorizar as atividades. Uma *lane* representa uma função de negócio ou um papel de negócio.	Nome Nome Nome

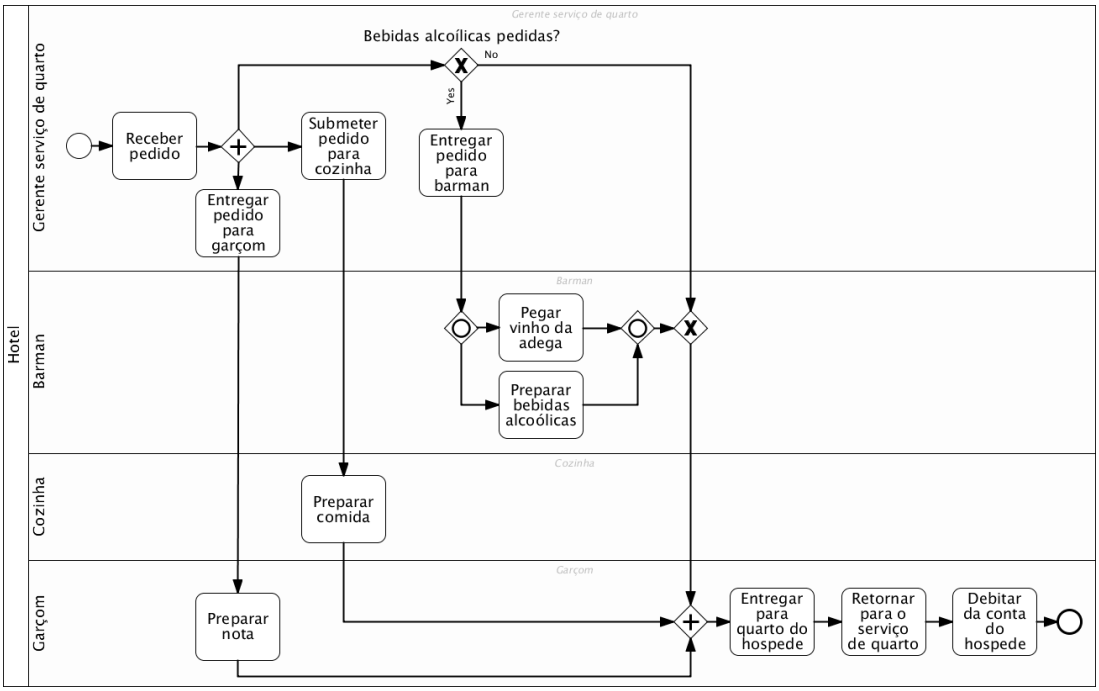


Figura 1- Exemplo de um modelo de processo de negócio (BPMN) (Adaptado de Leopold *et al.* [2012])

2.2 Geração de Linguagem Natural (GLN)

Geração de Linguagem Natural (GLN ou NLG da sigla em inglês Natural Language Generation) é a tarefa de gerar textos em linguagem natural a partir de uma representação em um sistema de máquina, como base de conhecimento ou numa forma lógica [Reiter e Dale, 2000]. Psicolinguistas preferem o termo produção de linguagem quando dadas representações formais são interpretadas como modelos para representação mental. Reiter e Dale definem GLN como: "GLN é um campo da inteligência artificial e linguística computacional que se preocupa com a construção de sistemas de computadores que são capazes de produzir textos em inglês ou em qualquer outro idioma a partir de uma representação não linguística da informação". Um sistema que utiliza GLN pode ser entendido como um tradutor que converte uma representação baseada em computador em uma representação em linguagem natural. Porém, os métodos para produzir a linguagem final são muito diferentes dos utilizados por um compilador devido a inerente expressividade de linguagens naturais. GLN pode ser visto como o oposto de Entendimento de Linguagem Natural (NLU – Natural Language Understanding), pois enquanto que no entendimento de linguagem natural o sistema precisa retirar a ambiguidade da sentença que foi submetida para produzir a representação em máquina, os sistemas GLN precisam tomar decisões sobre como representar um conceito através de palavras [Ovchinnkova, 2012].

Sistemas GLN combinam conhecimento sobre o idioma e sobre o domínio onde a aplicação será utilizada para produzir, de maneira automática, relatórios, explicações, mensagens de ajuda e outros tipos de textos [Reiter e Dale, 2000].

Os exemplos de sistemas GLN mais simples (e talvez mais triviais) são os que geram textos para cartas [Portet et. al, 2008]. Estes sistemas tipicamente não lidam com regras gramaticais, mas podem criar uma carta para um consumidor, por exemplo, relatando que o limite do cartão de crédito está prestes a ser alcançado.

Sistemas GLN mais complexos criam textos dinamicamente para atingir objetivos de comunicação. Como em outras áreas de processamento de linguagem natural, isto pode ser feito usando modelos explícitos de linguagens (por exemplo,

gramáticas) e o domínio. Neste caso, domínio faz alusão à área que o texto a ser elaborado pertence. Por exemplo, se for área médica, o texto deverá corresponder ao jargão utilizado pelos médicos e funcionários da área. Estes jargões serão diferentes dos utilizados pela área de engenharia.

Outra forma de criar textos dinamicamente é a utilização de modelos sintáticos derivados a partir da análise de textos escritos por pessoas. Este último se baseia na avaliação prévia de um texto corrido redigido por alguém com conhecimento adequado dentro do domínio específico, por exemplo, um médico, caso o domínio seja a área de medicina. O modelo sintático pode ser criado de maneira automática, através de uma pequena aplicação que tome como entrada um texto exemplo (*template*) e identifique as palavras chaves, ou seja, o sistema aprende a gerar novos textos.

O uso mais comum da tecnologia GLN é criar sistemas de computadores que são capazes de apresentar a informação para as pessoas de uma maneira que seja de fácil compreensão, sem ser necessário um treinamento prévio na tecnologia que foi utilizada para gerar tal informação. Por exemplo, em muitos casos, informações contidas em planilhas de contabilidade, bancos de dados de companhias aéreas, modelos de processo de negócio e outros, requerem um conhecimento considerável na área para serem interpretados corretamente. Isso significa que, frequentemente, existe uma necessidade de sistemas que podem apresentar tais informações de forma compreensível para usuários não especialistas. Devido a esta necessidade frequente, a tecnologia GLN tem crescido notoriamente [Reiter e Dale, 2000]. Por exemplo, técnicas GLN podem ser utilizadas para gerar previsões de tempo em forma textual a partir de mapas gráficos de tempo, resumir dados estatísticos extraídos de um banco de dados ou planilha e explicar informações médicas de um modo amigável para o paciente [Reiter e Dale, 2000].

A tecnologia GLN também pode ser utilizada para construção de sistemas que ajudam pessoas à criarem documentos rotineiros. Muitas pessoas utilizam boa parte de seu tempo para produção de documentos, frequentemente em situações onde esta produção de documentos não é sua responsabilidade principal. Um médico, por exemplo, pode usar uma parte significativa de seu tempo escrevendo cartas de referência, resumos para liberação de paciente (alta médica) e outros documentos rotineiros. Ferramentas GLN podem ajudar a produzir rapidamente documentos de

boa qualidade, melhorando, consequentemente, produtividade e alocação dos profissionais. Outros exemplos são: auxiliar na escrita de cartas consumidores de uma empresa; ajudar engenheiros a produzir resumos contendo informações sobre as opções que eles consideram como as melhores para realizar determinado projeto; ajudar gerentes de departamentos para produzir descrições de trabalho; e, ajudar autores técnicos para produzir instruções para a utilização de um software [Reiter e Dale, 2000].

2.3 RPST – *Refined Process Structure Tree*

Uma RPST é uma árvore contendo a hierarquia de subgrafos derivada a partir do grafo original [Vanhatalo *et al*. 2009, Polyvyanyy *et al.* 2011]. A RPST é baseada na observação que todo grafo que representa um fluxo pode ser decomposto em uma hierarquia de subgrafos logicamente independentes tendo uma única entrada e uma única saída. Tais subgrafos, com uma única entrada e uma única saída, são denominados fragmentos. Em uma RPST quaisquer uns destes dois fragmentos são ou interligados ou independentes. A hierarquia resultante pode ser vista como uma árvore onde a raiz é a árvore completa e as folhas são os fragmentos com um único arco. Um exemplo de árvore RPST é apresentado na Figura 2.

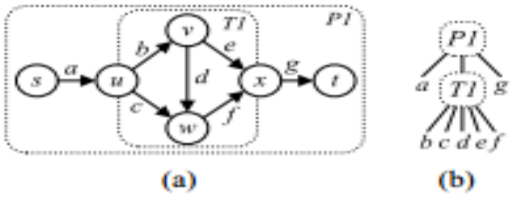


Figura 2 - (a) Um grafo e seus fragmentos canônicos, (b) a árvore RPST que representa o grafo (a).

Existem quatro tipos diferentes de classes de fragmentos: fragmentos triviais (T), *bonds* (B), polígonos (P) e rígidos (R). Fragmentos triviais consistem de dois nós conectados com um arco simples. Um *bond* representa um conjunto de fragmentos dividindo dois nós comuns. Em modelos de processo BPMN isso geralmente se aplica para casos de divisão e de união, incluindo estruturas de divisão e de união mais complexas como *loops*. Polígonos capturam sequência de outros fragmentos. Qualquer sequência em um modelo de processo é representada por um respectivo fragmento do tipo polígono. Se um fragmento não pode ser representado como os

tipos descritos, ele é classificado como rígido. Apesar da versão original da RPST ser baseada em grafos tendo apenas uma única entrada e um único ponto de saída, esta técnica pode ser facilmente estendida para computar uma árvore RPST para modelos de processos arbitrários, já que um modelo de processo pode ser representado facilmente através de uma estrutura em grafo (eventos e atividades são nós do grafo e suas ligações se tornam arestas). A Figura 3 e a Figura 4 ilustram os conceitos usando uma versão abstrata do processo do hotel, apresentado na Figura 1, e sua correspondente árvore RPST.

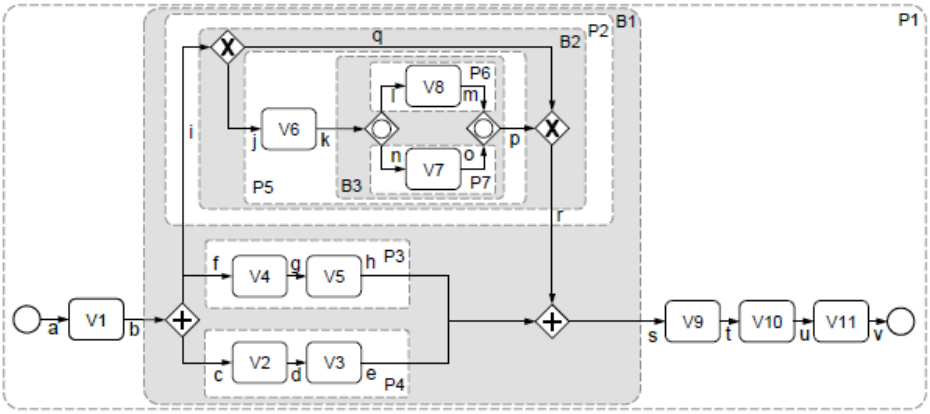


Figura 3 - Versão abstrata do processo apresentado na Figura 1 (adaptado de Leopold *et al.* [2012]).

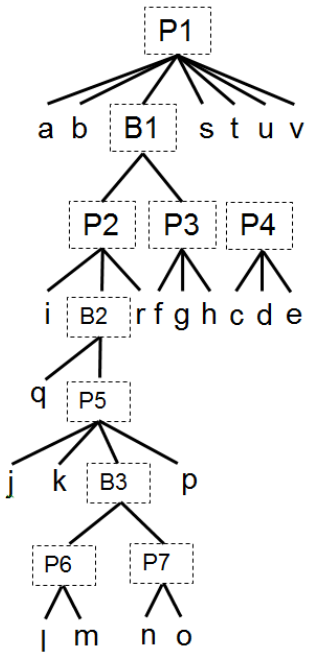


Figura 4 - RPST correspondente ao modelo de processo apresentado na Figura 1(adaptado de Leopold *et al.* [2012]).

2.4 DSynt – *Deep-Syntatic Tree*

Uma árvore DSynt (*deep-syntatic tree*) é uma representação de dependência introduzida pelo *framework* Meaning Text Theory [Mel'cuk, 1987].

Numa árvore DSynt cada nó é rotulado com um lexema semanticamente completo. Isto significa que lexemas tais como conjunções ou verbos auxiliares são excluídos. Além disso, cada lexema possui informações gramaticas sobre eles mesmos (*metadados*), chamados de gramemas (*grammemes*). *Grammemes* incluem, por exemplo, voz e conjugação dos verbos ou número e definições de substantivos. As vantagens do uso de árvores DSynt são: ela é rica, mas ainda mutável; ela representa sentenças; e existem ferramentas que, dada uma árvore DSynt como entrada, são capazes de transformá-la diretamente em uma sentença gramatical correta. A Figura 5 apresenta um exemplo de uma árvore DSynt, baseada na atividade "Receber pedido" do modelo de processo utilizado como referência (Figura 1). O exemplo apresentado (Figura 5) é somente uma ilustração conceitual, abstraindo-se aspectos técnicos. O procedimento detalhado para gerar uma árvore DSynt será explicado mais a frente, na seção 4.2.2.1. Neste exemplo, a árvore DSynt representa: (i) a ação da sentença: "Receber" identificada como "verbo"; (ii) o sujeito da sentença: "Gerente do serviço de quarto" identificado como "substantivo comum" e que tem um artigo definido associado a ele; (iii) objeto de negócio da atividade "Pedido" que é identificado como um substantivo comum e tem um artigo definido associado a ele.

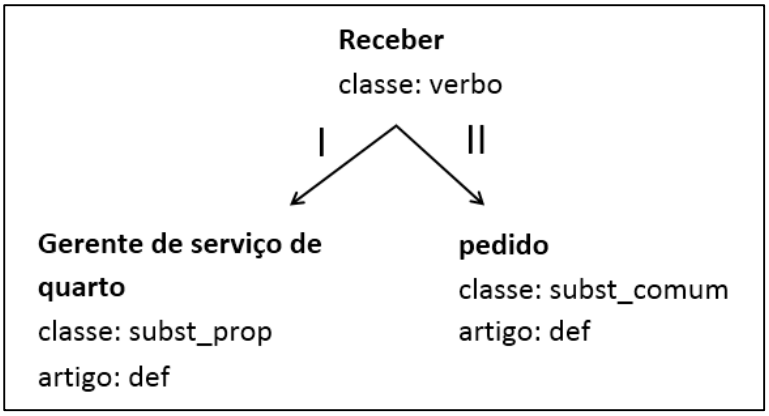


Figura 5 - Exemplo de uma árvore DSynt.

2.5 Resumo do capítulo

O presente capítulo teve como objetivo a apresentação dos principais conceitos e estruturas relacionadas com este trabalho. Estes conceitos serão utilizados durante todo o processo de geração de texto em linguagem natural a partir de um determinado modelo de processo. O capítulo seguinte visa detalhar, de maneira conceitual (abstraindo-se aspectos tecnológicos) e genérica, os passos necessários para a extração de informações linguísticas de um artefato de máquina (neste caso, um modelo de processo) para gerar um texto em linguagem natural que seja de fácil compreensão.

Capítulo 3: GERAÇÃO DE TEXTO EM LINGUAGEM NATURAL

Este capítulo apresenta a arquitetura de Reiter e Dale [1997] para geração de texto em linguagem natural. Esta arquitetura é genérica e amplamente utilizada para elaboração de ferramentas cujo objetivo é lidar com manipulação de textos em linguagem natural.

3.1 Técnicas para traduzir artefatos de máquina

Existem três técnicas principais para traduzir artefatos de máquina, por exemplo, para traduzir um modelo de processo de negócio para um texto em linguagem natural. Estas técnicas são: técnicas não linguísticas, *mail-merge* e técnicas linguísticas.

As técnicas mais simples, normalmente denominadas de não linguísticas, são baseadas em textos canônicos ou em *templat*es pré-definidos. No primeiro caso, o dado de entrada é diretamente mapeado para uma frase pré-definida. Por exemplo, um sistema traduzindo dados sobre previsão climática do tempo para texto em linguagem natural poderia usar a sentença *O tempo vai ser bom hoje* para representar um dia quente e ensolarado. Um pouco mais avançado, é usar *templates* onde algumas informações extras são inseridas na frase pré-definida. Por exemplo, no *template Hoje existe uma probabilidade de X% de chover*, o X poderia ser substituído pela probabilidade de chuva recuperada de uma base de dados [Leopold *et al*., 2012]. Reiter [1995] apresenta que estes tipos de técnicas não são consideradas verdadeiramente linguísticas, já que a manipulação do texto é baseada na manipulação de caractere(s) dentro de uma frase previamente definida.

As técnicas de *mail-merge* são mais complexas do que as técnicas *não linguísticas* e são encontradas no Microsoft Word e outros editores de texto populares. Neste caso, existem técnicas que inserem dados de entrada em espaços pré-definidos em um documento padrão, com técnicas que são essencialmente linguagens de programação que permitem que o texto de saída varie de modo arbitrário, de acordo com os dados de entrada [Reiter e Dele, 2000]. Para

exemplificação, a Figura 6 exibe um documento Word que terá sua saída segundo o conteúdo do arquivo utilizado como fonte de dados. Cada espaço pré-definido, como por exemplo, "<<CONTACT_FIRSTNAME>>" e "<<CONTACT_LASTNAME>>" será substituído pelo respectivo valor definido no arquivo utilizado como entrada. Suponha que o arquivo de entrada seja uma planilha Excel cujas colunas são: FirstName, LastName e etc. Através do emprego de técnicas de *mail-merge*, os dados de cada coluna podem ser mapeados para os espaços predeterminados no documento Word, de acordo com o mapeamento definido pelo usuário (por exemplo, coluna FirstName corresponde à <<CONTACT_FIRSTNAME>>).

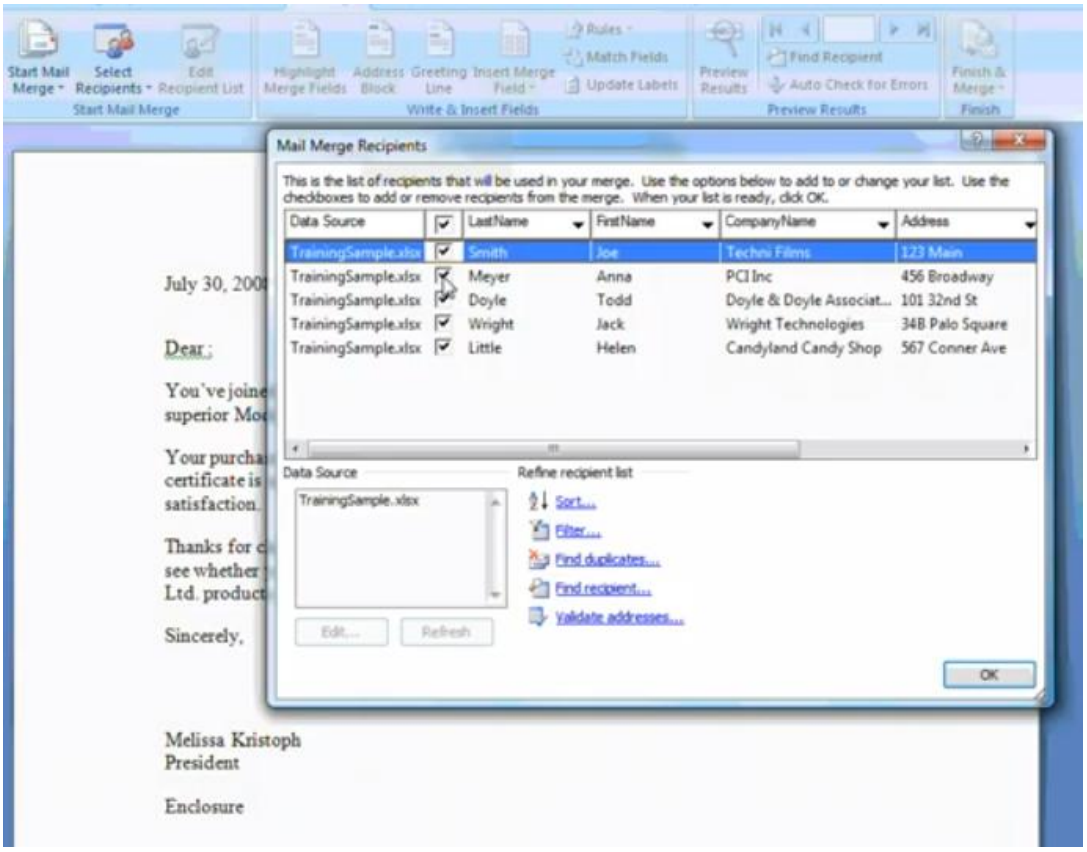


Figura 6 – Exemplo do uso da técnica *mail-merge* no Microsoft Word.

Técnicas linguísticas ou de geração real de linguagem natural usam estruturas intermediárias para obter uma representação mais detalhada do texto. Estas estruturas normalmente especificam os principais lexemas para cada sentença. Lexema é um conjunto de palavras de mesma classe morfológica que se distribuem de forma complementar e diferem morfologicamente entre si unicamente por sufixos e flexão. As palavras que compõem um lexema são chamadas de flexões do lexema [Jurafsky e Martin, 1999]. Por exemplo, as palavras "cantora" e "cantar" não compõem um

lexema porque não pertencem a mesma classe morfológica, substantivo e verbo, respectivamente. Já as palavras "cantor", "cantora", "cantores" e "cantoras" compõem um lexema porque pertencem à mesma classe morfológica, a dos substantivos, e diferem entre si unicamente por sufixos flexivos (morfema[2] zero, -a, es, -as). Além disso, agrega informações adicionais, definindo, por exemplo, a voz e o modo do verbo.

O núcleo de todas estas arquiteturas é o uso de uma estrutura intermediária para armazenar as mensagens antes de serem transformadas em sentenças de linguagem natural. A vantagem deste procedimento é o ganho significativo em manutenção e flexibilidade. Em um sistema baseado em *templates*, cada *template* deve ser manualmente modificado se for necessária uma mudança no texto a ser gerado como saída. Numa abordagem baseada em linguística, o texto gerado como saída pode ser alterado mudando-se um parâmetro na estrutura intermediária. Por exemplo, a sentença *O tempo vai ser bom hoje* pode ser facilmente transformada em *O tempo é bom hoje* modificando o tempo do verbo principal contido na estrutura intermediária.

As abordagens baseadas em *templates* e textos canônicos são consideradas como sendo inferiores às abordagens linguísticas em termos de manutenabilidade, qualidade do texto de saída e variação do texto. Reiter e Dale [2000] afirmam que sistemas que utilizam abordagens baseadas em *templates* são mais difíceis de manter e modificar, e que eles produzem saídas mais rígidas e mais pobres do que sistemas GLN. Busemann e Horacek [1998] afirmam que sistemas que utilizam tal abordagem não são capazes de incorporar aspectos linguísticos importantes [Deemter *et al.*, 2005].

Por outro lado, abordagens baseadas em templates e textos canônicos também possuem suas vantagens. Dentre elas podemos destacar que sistemas baseados em *templates* necessitam de menos tempo para serem elaborados e que não precisam estar totalmente completos para seu correto funcionamento, podendo ser ampliados através da inserção de novos *templates*. Outra vantagem é a adaptação flexível a um novo domínio. Quando se aplica um novo domínio ao sistema baseado em *templates*,

[2] **Morfema:** Fragmento mínimo capaz de expressar significado. Uma letra ou algumas letras acrescentadas a uma palavra para indicar flexões de gênero, número, pessoa, modo tempo etc. Morfemas são as desinências. *Morfema zero* é a ausência de um morfema para indicar a flexão. Neste caso, a flexão é representada por outros recursos. Por exemplo, analisando a palavra mar, que está singular, o morfema é zero porque não há nenhuma desinência que indique o singular.

muitos dos *templates* deverão ser reescritos, porém o mecanismo utilizado para a geração dos textos irá requerer pouca ou nenhuma modificação. O fato de que *templates* podem ser especificados manualmente é vantajoso em relação á GLN quando boas regras linguísticas ainda não estão disponíveis ou possuem condições muito específicas para sua utilização [Deemter *et al.*, 2005].

Como consequência, Reiter e Mellish [1993] propõem uma análise de custo benefício, buscando uma abordagem que faça uso das vantagens de uma abordagem para eliminar desvantagens da outra e, consequentemente, apresentando um maior custo benefício em sua utilização. Como resultado, muitos sistemas de geração de linguagem natural usam abordagens híbridas onde técnicas linguísticas são combinadas com textos canônicos e *templates* [Galley *et al*, 2001; Reiter *et al.*, 1992]

3.2 Arquitetura de sistemas de GLN

Reiter e Dale [1997] apresentam que muitos sistemas de geração de linguagem natural utilizam uma abordagem de pipeline baseada em três passos (Figura 7): (i) Planejamento do texto; (ii) Planejamento das sentenças; e, (iii) Realização das sentenças. Estes três passos são apresentados a seguir sendo suas execuções ilustradas com o processo apresentado na Figura 8.

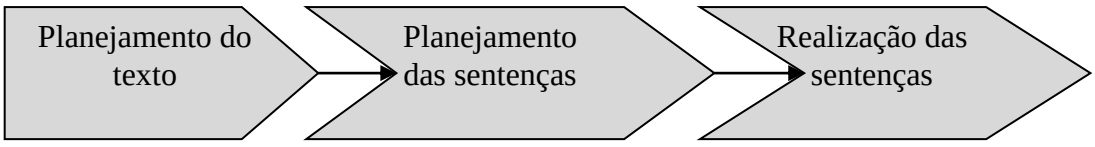


Figura 7 - Arquitetura de sistemas de GLN em *pipeline*

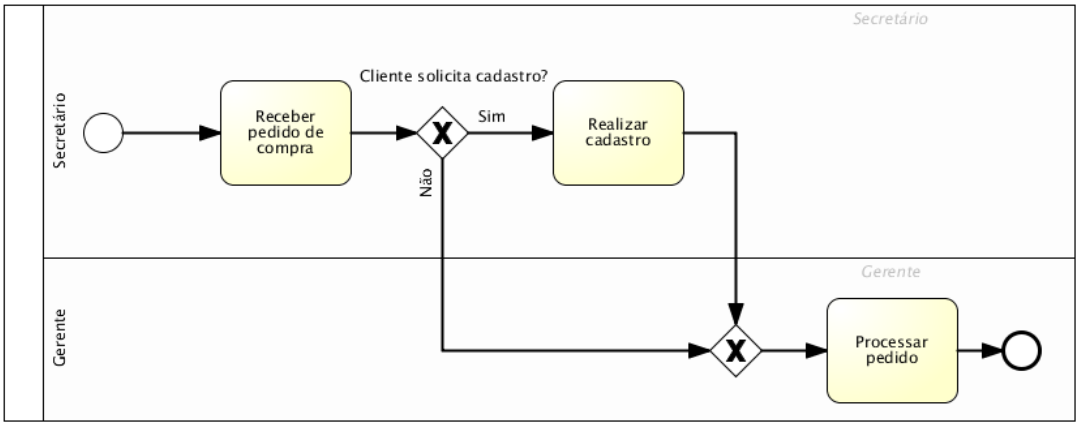


Figura 8 – Processo exemplo para ilustração da abordagem de pipeline.

3.2.1 Planejamento do texto

O "Planejamento do texto" é responsável por determinar a informação que o texto quer expressar e a ordem em que esta informação será representada.

Considerando como exemplo o processo apresentado na Figura 8, inicialmente é feita a identificação da função sintática de cada palavra correspondente às atividades do processo. No caso da primeira atividade "Receber pedido de compra", o resultado obtido é "receber = verbo" e "pedido de compra = substantivo".

Em seguida, é definida a ordem das atividades de modo que o texto produzido exiba as atividades em uma ordem sequencial correta, por exemplo:

- 1ª Atividade: Receber pedido de compra
- 2ª Atividade (opcional): Realizar cadastro
- 3ª Atividade: Processar pedido

Ao final deste passo, sabemos quais informações o modelo de processo quer transmitir e a ordem em que elas devem ser exibidas.

3.2.2 Planejamento das sentenças

O "Planejamento das sentenças" é responsável por escolher palavras específicas para expressar a informação obtida determinada no passo anterior. Neste passo, se for possível, mensagens são agregadas e pronomes são introduzidos a fim de obter uma maior variedade.

Neste passo, as informações determinadas no passo "Planejamento do texto" são combinadas com mapeamentos previamente construídos em relação aos componentes da notação BPMN quanto ao idioma específico do modelo, neste caso a língua portuguesa. Tais expressões são mensagens padrões previamente cadastradas no módulo de localização no dicionário do respectivo idioma, que são inseridas para representar algumas das notações específicas do modelo de processo. Exemplos de mapeamentos criados são: mapeamento do *gateway* exclusivo para expressões como “se então, senão, caso”; mapeamento do *gateway* paralelo para expressões como “em paralelo, paralelamente, concorrentemente”. Em outras palavras, sempre que um *gateway* é encontrado, este é mapeado através de expressões que o representam textualmente.

Além disso, há a adição de marcadores de discursos para maior flexibilização do texto e para obter uma leitura mais coerente e natural. Os marcadores são inseridos no começo de novas frases que possuem um relacionamento sequencial bem definido. Por exemplo, a mensagem “O processo começa quando” previamente cadastrada no módulo de localização é inserida sempre no início do texto. De maneira análoga, mensagens como: “Em seguida”, “Posteriormente”, “Finalmente o processo termina” são inseridas em locais específicos do texto.

Logo, ao final deste passo chega-se ao seguinte texto: “O processo começa quando secretário receber pedido de compra, caso cliente solicitar cadastro então ela realizar cadastro. Posteriormente, gerente inicia processar pedido”.

3.2.3 Realização das sentenças

A "Realização das sentenças" é responsável por transformar as mensagens em sentenças gramaticalmente corretas. Neste passo, a mensagem a ser transmitida é refinada gramaticalmente como, por exemplo, incluindo artigo definido antes de secretário (em “o secretário”).

Logo, a ferramenta receberia como entrada o modelo de processo apresentado na Figura 8 e produziria como saída o texto “O processo começa quando o secretário recebe um pedido de compra. Caso o cliente solicite cadastro então ele realiza o cadastro. Posteriormente, o gerente inicia o processamento do pedido”.

3.3 Geração de texto a partir de modelos de processo de negócio

Existe um conjunto de etapas para a geração automática de texto a partir de modelos de processos. Leopold *et al.* [2012] apresentam as etapas (Tabela 2) considerando os três passos da abordagem de Reiter e Dale [1997] e atividades relacionadas para geração automática de texto, além de considerarem também o desafio de flexibilidade.

Tabela 2 – Etapas para a geração automática de texto (adaptado de Leopold *et al.* [2012]).

1. Planejamento de textos
    a. Extração de informações linguísticas [Polyvyanyy *et al*., 2011].
    b. Linearização do modelo [Vanhatalo, 2009].
    c. Estruturação do texto [Meteer, 1992].
2. Planejamento de sentenças
    a. Lexicalização [Dalianis, 1999]
    b. Refinamento de mensagens [Stede, 1994]
3. Realização das mensagens [Lavoie e Rambow, 1997]

No passo "Planejamento de texto" existem três grandes desafios relacionados às atividades envolvidas neste passo.

- Extrair informações linguísticas dos elementos contidos no modelo de processo. Por exemplo, a atividade *Receber pedido de compra* tem que ser automaticamente dividida na ação *receber*, no objeto de negócio *pedido* e no complemento *de compra*. Sem essa separação, não seria claro qual das duas palavras define o verbo que representa a ação da atividade. A análise de rótulos (ou nomes) de tarefas do modelo de processo se torna ainda mais complicada devido ao pequeno tamanho de rótulos contidos em modelos de processo [Dixon, 2008];
- Linearizar modelos de processo em uma sequência de frases. Modelos de processos raramente consistem apenas de uma sequência de tarefas. Eles também incluem ramificações paralelas e pontos de decisão;
- Estruturar e formatar texto, como parágrafos e marcadores.

O passo "Planejamento de sentenças" engloba as tarefas de lexicalização e refinamento de mensagens e tem os seguintes desafios:

- O aspecto de lexicalização refere-se ao mapeamento de modelos de processo BPMN para palavras específicas. Neste caso, o desafio é a necessidade de integração de informações linguísticas extraídas de elementos de modelos de processo com os de estruturas de controles, de divisão e de união de modo que o processo seja descrito em uma maneira de fácil compreensão;
- O aspecto de refinamento de mensagens refere-se à construção do texto. O desafio relacionado envolve a agregação de mensagens, como a introdução de

expressões de referência (por exemplo, pronomes) e também a inserção de marcadores de discurso como, por exemplo, *depois disso* e *subsequentemente*. A fim de consolidar as mensagens de uma maneira adequada, a opção de agregação tem que ser identificada primeiramente e então decidir onde pode ser aplicada para melhorar a qualidade do texto. A introdução de expressões de referência requer o reconhecimento automático de tipos de entidades. Por exemplo, o ator 'Cozinha' deve ser referenciado pelo pronome 'ela' enquanto que o ator 'Garçom' deve ser referenciado pelo pronome 'ele'. A inserção de marcadores de discurso deve aumentar a facilidade de leitura e a variedade do texto. Por este motivo, os marcadores de discurso[3] devem ser inseridos nas posições adequadas.

No contexto de "realização de mensagens", a atividade de geração de sentenças gramaticalmente corretas é realizada e ela inclui, dentre outras tarefas:

- Determinação de uma ordem adequada das palavras;
- Classificação do gênero;
- Introdução de artigos;
- Introdução de pontuação;
- Transformação das palavras em maiúsculas ou minúsculas de acordo com o contexto em que estão inseridas (por exemplo, começo de frase, substantivo próprio e etc.).

Além das atividades núcleo de geração de linguagem natural, a flexibilidade também é um recurso importante. Como não esperamos que modelos que serão utilizados como entrada sejam definidos de acordo com uma convenção específica, temos que lidar com características diferenciadas de cada artefato utilizado como entrada. Cenários diferentes devem ser cobertos com implicações diferenciadas para o texto de saída. Por exemplo, se um modelo usa raias e consequentemente provê uma descrição dos atores, a sentença pode ser representada na voz ativa como, por exemplo, *O Gerente verifica o sistema*. Se não forem definidos atores para a tarefa especificada, a descrição deve vir na voz passiva, como, por exemplo, *O sistema é verificado.*

[3] Marcadores de discurso, neste contexto, são palavras e expressões utilizadas para correlacionar de maneira explícita as frases em um texto (por exemplo, "Paralelamente", "em seguida", "consequentemente", etc.).

3.4 Resumo do capítulo

O presente capítulo apresentou, de maneira detalhada, porém genérica, a área da inteligência artificial dedicada à extração de informações linguísticas a partir de artefatos de máquina para geração de textos em linguagem natural. Foram explicados os diversos passos que são executados para gerar um texto em linguagem natural. Porém, foram abstraídos os aspectos tecnológicos como: linguagem de programação a ser utilizada, representação dos conceitos através de estruturas de dados, definição de operações e/ou classes, divisão em módulos e etc.

O capítulo seguinte tem como objetivo detalhar os aspectos tecnológicos relacionados com cada um dos passos. Em outras palavras, são explicadas como e quais técnicas foram utilizadas para o mapeamento dos conceitos apresentados para o correto funcionamento de um software capaz de executar os passos necessários para gerar um texto em linguagem natural a partir de um modelo de processo.

Capítulo 4: Framework Genérico para Geração de Texto a Partir de Modelos de Processos

Este capítulo apresenta os componentes do *framework* para geração de texto em linguagem natural a partir de modelo de processo de negócio em BPMN. Inicialmente é apresentada uma visão geral da sequência de atividades executadas pelo *framework* para geração de texto em linguagem natural. Em seguida, são detalhadas as implementações para tratar cada uma das etapas do pipeline de três fases para geração de texto em linguagem natural apresentado na Figura 9, a qual representa uma visão geral dos seus seis componentes principais.

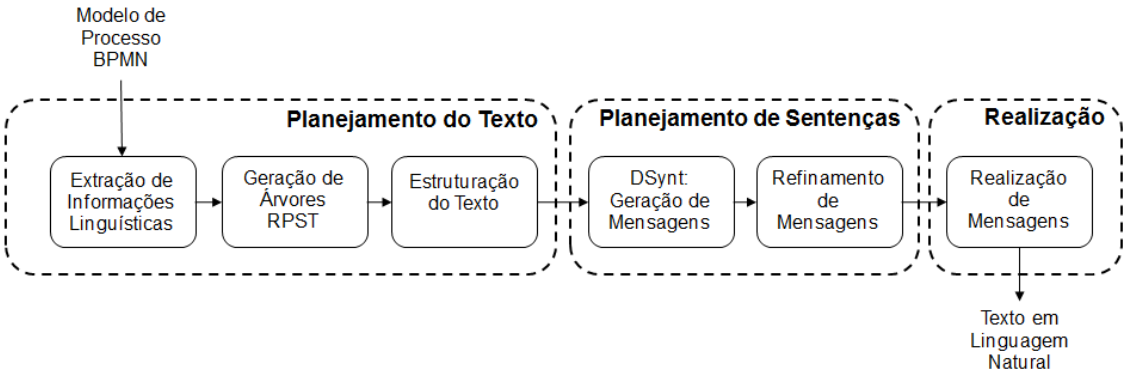


Figura 9 - Visão geral dos seis componentes para GLN (Adaptado de Leopold *et al.* [2012]).

4.1 Sequência de atividades para geração de texto em linguagem natural

Abstraindo alguns componentes intermediários auxiliares, a Figura 10 apresenta a sequência das atividades executadas pelo sistema durante o processo de geração de texto em linguagem natural a partir de um modelo de processo de negócio. Os passos referentes a este diagrama são detalhados a seguir.

1º Passo: A sequência de atividades é disparada quando um usuário, de posse de um arquivo que representa um modelo de processo, solicita a leitura do mesmo. Neste trabalho, foi escolhido JSON [JSON, 2013] como a estrutura para representação do modelo de processo. Neste momento, um objeto da classe JSONReader é criado, cuja principal responsabilidade é ler e transformar a

informação contida em um arquivo .json em uma estrutura adequada de um modelo de processo. O objeto retorna um modelo de processo condizente com o arquivo lido previamente. Este modelo retornado é armazenado temporariamente, para futura manipulação, em memória através de uma classe que utiliza as estruturas de dados adequadas para o mapeamento de todos os elementos presentes em um processo de negócio;

2º Passo: De posse do modelo, o usuário solicita a transformação deste modelo na sua respectiva forma em linguagem natural. Faz isso por intermédio de uma classe estática responsável por chamar sequencialmente todos os componentes do pipeline da arquitetura GLN;

3º e 4º passos: O idioma do modelo é obtido através de um atributo do modelo do processo que identifica o idioma em que o mesmo foi confeccionado. Assim que a linguagem é detectada, o componente linguístico específico é acionado (3º passo), instanciando os objetos com as respectivas implementações para tratar o idioma em questão, bem como o ajuste do componente de localização (4º passo). O componente de localização é um módulo responsável pela tradução de palavras e sentenças específicas, cadastradas como chaves genéricas do dicionário. As chaves são cadastradas como um conjunto de enumerações (estrutura de dados 'enum'), onde cada item deve possuir sua respectiva tradução para o idioma que está sendo utilizado no momento. Para isso, basta ter cadastrado as traduções das chaves fornecidas em um arquivo externo. O módulo recebe como requisição uma determinada chave, cuja tradução deve ser localizada no dicionário do idioma correspondente. Por exemplo, considerando o idioma português, ao detectar a chave "PROCESS_BEGIN_WHEN" a mensagem traduzida "O processo começa quando" é retornada;

5º Passo: Em seguida, é necessário transformar a estrutura atual do modelo de processo em uma estrutura intermediária de modelo baseada em grafos, onde cada arco é representado por uma aresta do grafo e cada nó um elemento do processo como: evento, *gateway*, atividade, etc;

6º Passo: Com o modelo de processo baseado em grafo, a árvore RPST é criada contendo toda a informação linguística essencial para gerar o texto;

7º Passo: Em seguida, o nó raiz da árvore é utilizado como ponto de partida para geração de todas as árvores DSynt que representarão, em modo textual, as diversas atividades e componentes (*gateways*, eventos) do modelo de processo. Terminado este passo, já temos a representação textual do modelo de processo. Porém, esta ainda não é adequada para leitura devido a problemas gramaticais e redundâncias;

8º Passo: A primeira etapa do tratamento do texto é a agregação das diversas mensagens. Este componente retorna mensagens devidamente agrupadas segundo critérios como, por exemplo, atividades feitas pelo mesmo ator em uma sequência direta;

9º Passo: A segunda etapa corrige alguns problemas resultantes da agregação através de um componente de inserção de expressões de referência. Por exemplo, ele executa a substituição da repetição do nome do ator em uma mesma sentença por seu respectivo pronome pessoal, de modo a obter uma maior legibilidade;

10º Passo: A terceira etapa é responsável por adicionar conectivos sequenciais às diversas sentenças, de modo a conectá-las semanticamente no texto. Por exemplo, as sentenças "Garçom entregar pedido" e "Cozinha preparar comida" seriam conectadas através da expressão 'em seguida': "Garçom entregar pedido Em seguida Cozinha preparar comida". Note que o uso de artigos, pontuação e conjugação dos verbos ainda está incorreto. Isto será tratado na etapa seguinte;

11º Passo: A quarta e última etapa tem como objetivo corrigir os erros gramaticais presentes nas sentenças resultantes das transformações de modo a gerar um texto fluido e coerente. Neste componente, ocorre, por exemplo, a transformação da sentença "Garçom entregar pedido. Em seguida, cozinha preparar comida" em "O garçom entrega o pedido. Em seguida, a cozinha prepara comida.". Após a correção de todas as mensagens contidas nas árvores DSynt, o componente realiza a concatenação das mensagens e retorna um texto corrido para o usuário, representando o modelo de processo original em linguagem natural.

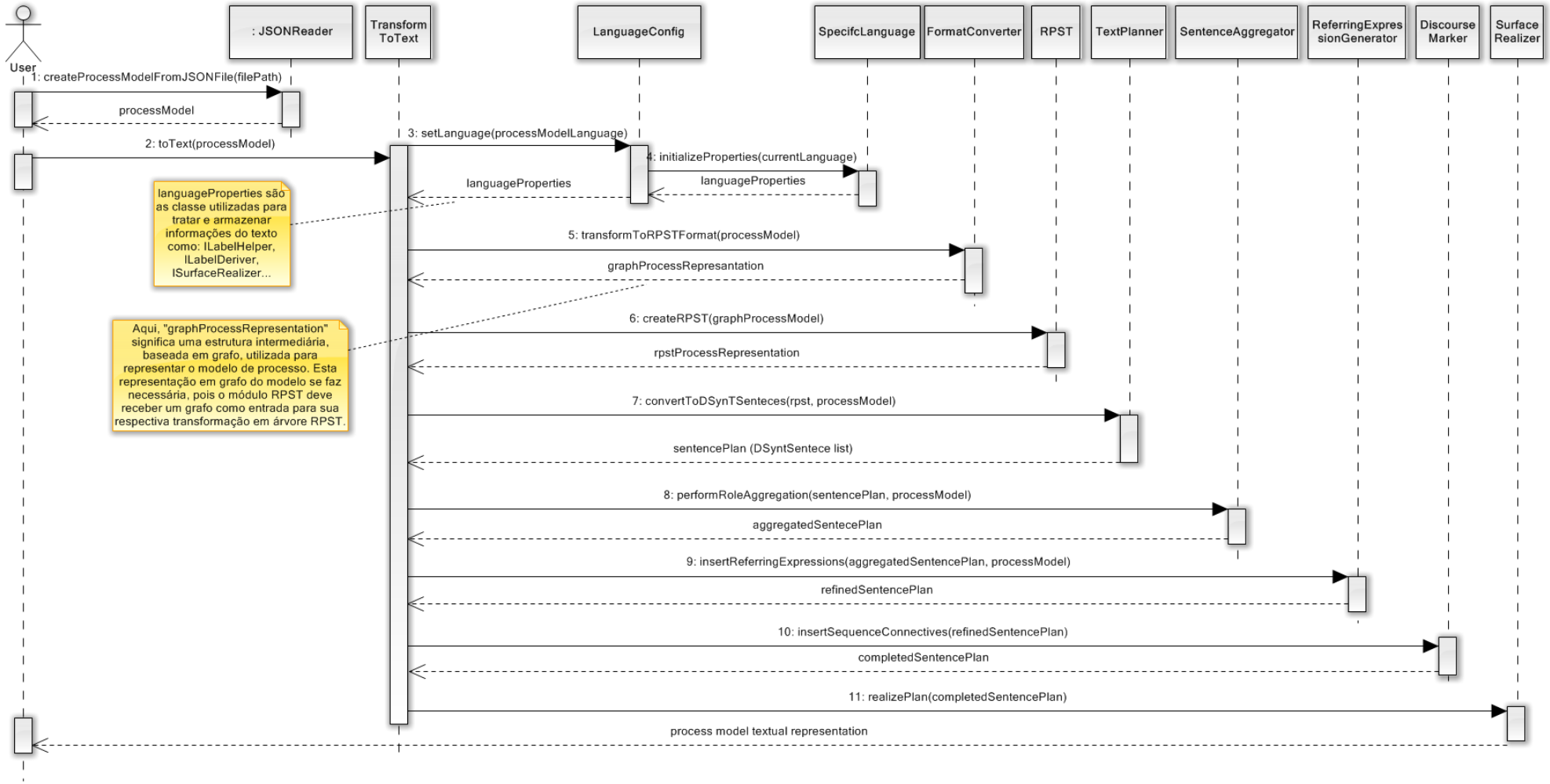


Figura 10 - Diagrama de sequência da execução da aplicação

4.2 Pipeline de três fases para geração de texto em linguagem natural

Nesta seção são apresentados detalhes da implementação realizada para o *framework* genérico para geração de texto a partir de modelos de processos de negócio, considerando as etapas apresentadas na Figura 9. Também são apresentadas as principais classes utilizadas no processo GLN. Por possuírem funcionalidades relacionadas, podemos agrupá-las facilmente em módulos (ou pacotes) bem definidos, apresentados na Figura 11, os quais são descritos a seguir.

- **LabelAnalysis**: Este pacote contém todas as classes responsáveis pela manipulação dos rótulos (*labels*) presentes em um modelo de processo, para descrição de seus diversos elementos. Definem também as funções linguísticas, como classificação das palavras (substantivo, verbo, adjetivo, etc.) utilizadas;
- **Fragments**: Este pacote contém as classes responsáveis pela representação de uma sentença em alto nível. Tem sua origem na leitura de uma árvore DSynt e sua respectiva simplificação de modo que os dados importantes, como o objeto de negócio, verbos, complemento da atividade, sejam obtidos mais facilmente;
- **DSynt**: Este pacote contém as classes responsáveis pela representação da estrutura de dados em árvores DSynt, que tem sua origem na transformação de uma árvore RPST, com seus diversos nós, em diversas árvores DSynt para futura manipulação pelo módulo de realização de mensagens;
- **Localization**: Este pacote contém as classes responsáveis pela execução da lógica de localização de palavras bem como o armazenamento dos diversos dicionários utilizados para cada idioma. Também é responsável pela definição das chaves que serão utilizadas como entradas para obtenção da tradução correspondente ao idioma utilizado no modelo de processo;
- **ISurfaceRealizer**: Interface responsável por definir as operações necessárias para a tarefa de realização de mensagens;

- **<Language>Realizer**: Corresponde a pacotes contendo as classes responsáveis pela implementação concreta das interfaces e classes abstratas definidas no pacote "GeneralLanguageCommon" para cada idioma. Por exemplo, EnglishRealizer, PortugueseRealizer etc. As classes correspondentes a estes pacotes definem os diversos algoritmos necessários para a correta execução do processo GLN para o idioma em questão;
- **LanguageConfig**: Esta classe é responsável por, dado um determinado idioma, instanciar as interfaces com suas respectivas implementações referentes ao idioma.

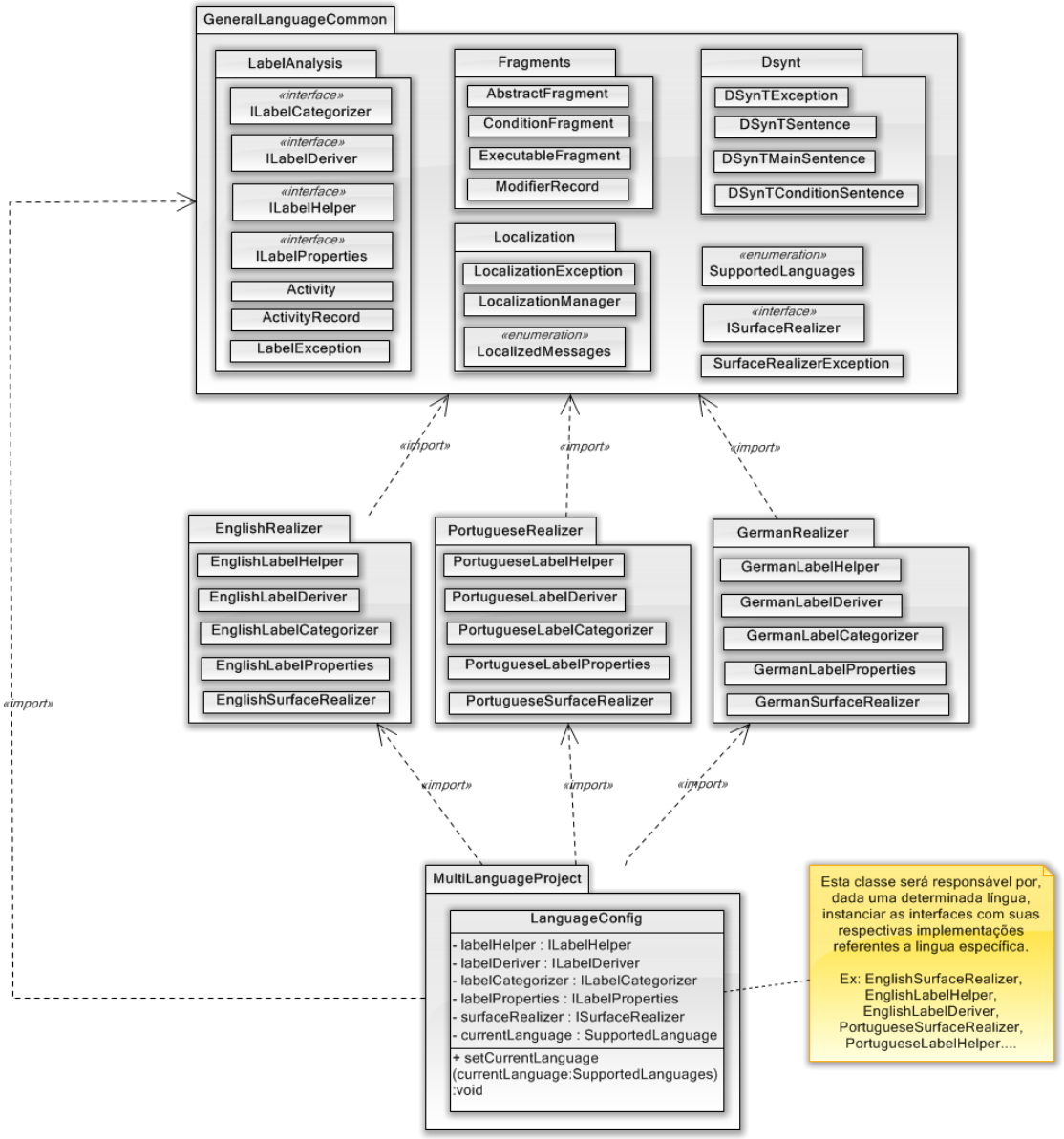


Figura 11 – Diagrama de pacotes contendo as principais classes utilizadas no processo GLN.

4.2.1 Planejamento do texto

Para tratar o passo “Planejamento do texto”, as seguintes implementações foram consideradas.

4.2.1.1 Extração de informações linguísticas

A principal meta do componente "Extração de informações linguísticas" é a inferência adequada das informações linguísticas de todos os rótulos (nome das atividades, descrição de condições e atores) de um modelo de processo de negócio.

Por isso, é importante ser capaz de lidar com diferentes tipos de representações linguísticas, denominadas estilos de rótulos[4]. Neste trabalho, foi adaptada a proposta de Leopold *et al.* [2010], a qual é capaz de realizar a extração de ações e objetos do negócio a partir de rótulos de atividade e eventos, bem como classificá-los corretamente em seus respectivos estilos. A adaptação consistiu em, basicamente, identificar os trechos dos algoritmos que foram confeccionados especificamente para o inglês. Tais trechos foram abstraídos através de métodos cujos nomes fossem claros e representativos da funcionalidade que executam. Todos estes métodos necessários foram movidos para a definição de uma interface a qual passou a especificar quais métodos devem ser implementados para cada novo idioma. Desta maneira, quando uma decisão deve ser tomada levando em consideração aspectos linguísticos, as operações definidas nas interfaces são invocadas e a classe que implementa o método da interface para o idioma sendo tratado naquele momento executa o algoritmo específico para o idioma.

Nesta etapa, são utilizadas implementações específicas, baseadas no idioma do modelo de processo, dos componentes genéricos definidos no pacote "GeneralLanguageCommon" (Figura 11). Neste momento, as principais operações executadas para extração e inferência de informações linguísticas são:

- Verificação se uma palavra é um verbo (Algoritmo 1)
- Verificação se uma palavra é um substantivo (Algoritmo 2)
- Verificação se o rótulo possui uma conjunção (Algoritmo 3)
- Verificação se uma palavra é um artigo definido (Algoritmo 4)
- Remoção de artigos de um objeto de negócio (Algoritmo 5)
- Obtenção das preposições utilizadas no idioma do modelo (Algoritmo 6)

[4] Estilos de rótulos são as diferentes formas utilizadas para descrição de uma atividade em um modelo de processo. Neste contexto existem três formas conhecidas: AN (Action-Noun label, como por exemplo, "Ativos de manutenção"), VOS (Verb-Object label, como por exemplo, "Validar pedido") e DES (Descriptive-Style label, como por exemplo, "Raphael atende chamada").

```
1:   isVerb(String word)
2:   isVerb = false;
3:   if getPos(word) == PortugueseLabelHelper.POS_VERB then
4:      isVerb = true;
5:   else
6:      isVerb = VerbManager.containsVerb(word);
7: return isVerb;
```

Algoritmo 1 - Verificação se uma palavra é um verbo: Operação isVerb da classe PortugueseLabelHelper.

```
1:   isNoun(String word)
2:   isNoun = false;
3:   if getPos(word) == PortugueseLabelHelper.POS_NOUN then
4:      isNoun = true;
5: return isNoun;
```

Algoritmo 2 - Verificação se uma palavra é um substantivo: Operação isNoun da classe PortugueseLabelHelper

```
1:   checkForConjunction(Label label, ILabelProperties props)
2:   splitLabel = label.split(“ ”);
3:   if label.contains(“ e ”) then
3:      props.setIndexConjunction(label.indexOf(“ e ”))
4:      props.setHasConjunction(true);
5:      for all word ∈ splitLabel.words do
6:      if word == “e” then
7:         props.setIndexConjunction(word);
8:         break for;
9:   if label.contains(“ / ”) and
        label.indexOf(“ / ”) < props.getIndexPrep) then
10:      props.setIndexConjunction(label.indexOf(“ / ”))
11:      props.setHasConjunction(true);
12:      for all word ∈ splitLabel.words do
13:      if word == “/” then
14:         props.setIndexConjunction(word);
15:         break for;
16:   if label.contains(“,”) then
17:      props.setIndexConjunction(label.indexOf(“,”))
18:      props.setHasConjunction(true);
19:      for all word ∈ splitLabel.words do
20:      if word.contains(“,”) then
21:         props.setIndexConjunction(word);
22:         break for;
```

Algoritmo 3 - Verificação se o rótulo possui uma conjunção: Operação checkForConjunction da classe PortugueseLabelHelper

```
1:   isDefArticle(String word)
2:   isDefArticle = false;
3:   if LanguageData.defArticles.contains(word) then
4:      isDefArticle = true;
5: return isDefArticle;
```

Algoritmo 4 - Verificação se uma palavra é um artigo definido: Operação isDefArticle da classe PortugueseLabelHelper

```
1:   removeArticleFromBO(String bo)
2:   splittedBo = bo.split(“ ”);
3:   if splittedBo.length > 1 and isArticle(splittedBo.words[0]) then
4:      bo = bo.substring(splittedBo.words[0].length);
```

Algoritmo 5 - Remoção de artigos de um objeto de negócio: Operação removeArticleFromBO da classe PortugueseLabelHelper

```
1:   getPrepositions()
2:   if prepositions == null then
3:      prepositions = LanguageData.noArticlePrepositions U
       LanguageData.articlePrepositions;
4:   return prepositions;
```

Algoritmo 6 - Obtenção das preposições utilizadas no idioma do modelo: Operação getPrepositions da classe PortugueseLabelProperties

Além disso, o trabalho de Leopold *et al.* [2012] foi utilizado como base para inclusão de caminhos e rótulos de decisão para cobrir todos os elementos relevantes de um modelo de processo. Baseando-se na estrutura de rótulos (descrição textual relacionada com os elementos) de modelos de processo, pode-se extrair ações, objetos de negócio e possivelmente modificar atributos de qualquer rótulo presente no modelo de processo, independentemente da representação linguística atual. Esta extração utiliza os métodos definidos nos anexos listados anteriormente. Por exemplo, tomando a atividade “Receber pedido” do modelo de processo do hotel (Figura 1), o algoritmo classificaria as palavras “Receber” e “pedido” como verbo (Ação da atividade) e substantivo (objeto de negócio da atividade), respectivamente.

Para que o sistema atue de forma genérica e independente da representação linguística atual, ou seja, permita evoluções futuras para outros idiomas de forma mais simples, as implementações de Leopold *et al.* [2012], que são específicas para a língua inglesa, foram generalizadas.

Um projeto central foi criado (GeneralLanguageCommon, Figura 11) para conter as interfaces necessárias para geração de texto em linguagem natural. As classes de cada idioma devem prover os métodos definidos nestas interfaces. Além disso, há um mecanismo de localização (Pacote “Localization” do GeneralLanguageCommon, Figura 11) onde todas as chaves definidas devem conter

suas respectivas traduções, por meio de um arquivo texto, para o idioma desejado. Chaves são mensagens a serem exibidas no texto final. Por exemplo, considerado o idioma português, ao detectar a chave "PROCESS_BEGIN_WHEN" a mensagem traduzida "O processo começa quando" é retornada. Desta maneira, a adição do suporte de novos idiomas se torna simples e rápida, sem que seja necessária a alteração dos demais componentes externos ao projeto central.

Uma vez que o algoritmo tenha sido executado para todas as atividades de um modelo de processo que possuam algum rótulo descrevendo-as, as informações extraídas dão origem a uma simplificação do modelo de processo original, definido através de uma estrutura em grafo para ser compatível e facilmente manipulado pelo módulo seguinte. O algoritmo utilizado para transformação de um modelo de processo em sua respectiva estrutura de grafo é o mesmo elaborado por Leopold *et. al* [2012], que utiliza uma estrutura de dados baseada em grafos, elaborada por Polyvyanyy *et al.* [2011].

4.2.1.2 Geração de árvores RPST

Este módulo é responsável por criar a representação em árvore RPST (Seção 2.3) do modelo de processo que serviu como entrada para o sistema, de modo a servir como uma base para descrever o processo passo a passo. Este módulo recebe como entrada a estrutura em grafo do modelo de processo gerada no passo anterior de extração de informações linguísticas do modelo (Seção 4.2.1.1). Os algoritmos utilizados neste módulo são os mesmos utilizados e disponibilizados por Leopold *et al.* [2012], ou seja, não houve ajustes no *framework* original para generalização deste módulo.

Alguns detalhes importantes deste módulo devem ser apresentados. Polyvyanyy *et al*. [2011] propõe um algoritmo de geração de árvores RPST que não ordena os fragmentos de acordo com o fluxo de controle. Porém, este problema foi abordado e contornado por Leopold *et al.* [2012] cuja implementação foi empregada neste trabalho. Neste caso, para cada nível na árvore RPST, a ordem pode ser determinada através das organizações dos fragmentos de acordo com suas aparições no modelo de processo. Considerando o exemplo apresentado na Figura 4, o primeiro nível começa com o fragmento trivial *a*, conectando o evento inicial e o vértice *V1*.

De maneira semelhante, o fragmento trivial *b*, o *bond* B1 e os fragmentos triviais *s*, *t*, *u* e *v* seguem em sequência.

Além da introdução de uma ordem adequada, informações linguísticas obtidas a partir do primeiro componente e também algumas meta-informações são adicionadas à árvore RPST. Por exemplo, o vértice *V1* do fragmento trivial *a* (Figura 4) é registrado com a ação *receber*, o objeto de negócio *pedido* e o ator *gerente de serviço de quarto* obtidos a partir do modelo de processo de negócio utilizado como exemplo (Figura 1) e extraídos na atividade anterior de extração de informações linguísticas do modelo (Seção 4.2.1.1). De maneira semelhante, o *bond B2* é salvo com sua respectiva ação *pedidas*, o objeto de negócio *bebidas* e o adjetivo *alcoólicas.* Além disso, o *bond* é classificado como um *XOR-gateway* com arcos *sim/não* do tipo *omissão.* O aspecto de omissão é derivado do fato de que uma ramificação está diretamente seguindo o *gateway* de *split* e provê a possibilidade de omitir a atividade da ramificação alternativa, já que esta não possui nenhuma atividade relacionada, ou seja, não há atividades que são executadas exclusivamente para a condição negativa de bebidas alcóolicas não serem pedidas.

4.2.1.3 Estruturação do texto

Para obter um texto de melhor qualidade e compreensão, através da utilização do algoritmo definido por Leopold *et. al* [2012], o texto é tratado com a inclusão de parágrafos e marcadores para estruturar as mensagens. Parâmetros editáveis para definição do tamanho dos parágrafos também são incluídos. Uma vez que um limite de tamanho do parágrafo tenha sido atingido, a mudança do ator responsável pela execução das tarefas ou a existência de um evento intermediário é usada para identificar a necessidade de se introduzir um novo parágrafo.

Considere, por exemplo, que o limite de tamanho do parágrafo tenha sido especificado para conter somente uma atividade e, considerando as atividades da Figura 12 como exemplo, ao detectar o *gateway* de paralelismo (AND) as propriedades “hasParagraph” das atividades “Submeter pedido para cozinha” e “Entregar pedido para garçom” (nó c e f da árvore RPST - Figura 4) são definidas, de maneira automática pelo sistema, como *true.* Esta propriedade é usada mais a frente pelo módulo de realização de mensagens (Seção 4.2.3) para saber se uma atividade específica deve ser exibida no início de um novo parágrafo. Tal propriedade foi

criada, como um atributo do nó da árvore RPST, especificamente para auxiliar a geração de um texto cuja estrutura visual é mais agradável para o leitor.

De maneira similar, são utilizados marcadores para configurar o recuo das mensagens adequadamente. Toda sequência de atividades realizada pelo mesmo ator que seja maior que um parâmetro definido é representada por uma lista com marcadores, onde cada frase possui o mesmo nível de recuo. Para ilustrar este cenário, considere as duas atividades "Submeter pedido para cozinha" e "Entregar pedido para garçom" executadas em paralelo pelo ator *Gerente serviço de quarto.* Tais atividades (nó f e c da árvore RPST - Figura 4) têm uma propriedade chamada "hasBullet" com o valor *true* e recebem o mesmo nível, através de atribuição do nível à uma propriedade chamada "senLevel", para fins de indentação futura. Assim como a propriedade "hasParagraph", estas propriedades também fazem parte do nó da árvore RPST. Desta maneira, no módulo de realização de mensagens (Seção 4.2.3), o algoritmo detecta que estas mensagens devem ser antecedidas por marcadores e no mesmo nível de indentação. Além disso, os ramos de qualquer divisão tendo mais de dois arcos de saída são apresentados como pontos de marcação. No caso de divisões aninhadas, o ponto de marcação é recuado adequadamente de modo que o leitor possa facilmente se situar dentro das estruturas aninhadas.

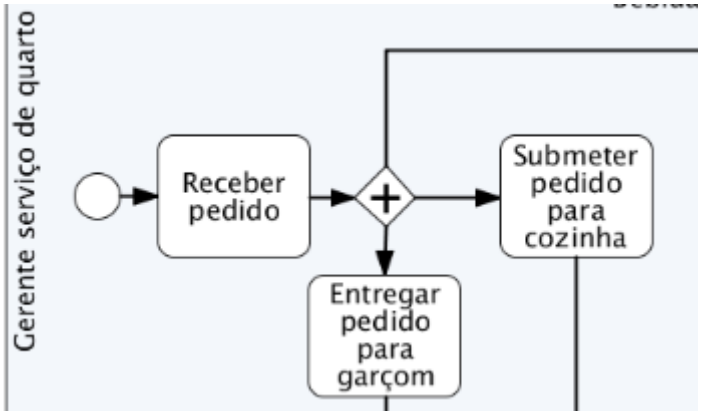


Figura 12 – Exemplo de atividades disparadas paralelamente por um mesmo ator.

4.2.2 Planejamento de sentenças

Para tratar o passo "Planejamento de sentenças", as seguintes implementações foram consideradas.

4.2.2.1 Geração de mensagens a partir de DSynt

O componente de geração de mensagens transforma a árvore RPST previamente criada em uma lista de mensagens intermediárias. Isto significa que a informação linguística do modelo não é diretamente mapeada para o texto final, mas sim para uma representação conceitual que ainda requer diversas modificações. Em particular, cada sentença é armazenada numa árvore DSynt (apresentada na Seção 2.4).

Baseado nesta representação intermediária, Leopold *et. al* [2012] desenvolveram um algoritmo que recursivamente percorre a anotação RPST e gera mensagens baseadas em árvores DSynt para fragmentos triviais e *bonds*.

A implementação de Leopold *et al.* [2012] havia sido elaborada de maneira específica para o inglês. Logo, neste trabalho foi necessária a identificação dos respectivos trechos de código específicos e criação de novas operações genéricas nas interfaces criadas para realizar a generalização. Além disso, as implementações das operações específicas para a língua portuguesa foram elaboradas. Algumas das operações utilizadas nesta etapa são:

- Verificação se um dado substantivo está no plural ou não (Algoritmo 7);
- Verificação se uma dada palavra é um artigo indefinido (Algoritmo 8);
- Obtenção de quantificadores para o idioma em questão (Algoritmo 9);
- Verificação se uma dada palavra é um verbo (Algoritmo 1).

```
1:   isPlural(String nounToBeChecked)
2:   isPluralNoun = false;
3:   for all pluralSuffix ∈ pluralSinguarMap.keySet do
4:       if nounToBeChecked.endsWith(pluralSuffix) then
5:           isPluralNoun = true;
6:           break for;
7:  return isPluralNoun;
```

Algoritmo 7 - Verificação se um substantivo está no plural ou não: Operação isPlural da classe auxiliar NounManager.

```
1:   isIndefArticle(String word)
2:   isIndefArticle = false;
3:   if LanguageData.indefArticles.contains(word) then
4:      isIndefArticle = true;
5: return isIndefArticle;
```

Algoritmo 8 - Verificação se uma palavra é um artigo indefinido: Operação isIndefArticle da classe PortugueseLabelHelper.

```
1:   getQuantifiers()
2:   return { "um", "o","a", "todo", "qualquer", "mais", "maioria",
             "nenhum", "algum", "tal", "um", "dois", "tres", "quatro",
             "cinco", "seis", "sete", "oito", "nove", "dez"};
```

Algoritmo 9 - Obtenção dos quantificadores do idioma: Operação getQuantifiers da classe PortugueseLabelHelper.

Os parágrafos seguintes introduzem os principais conceitos sobre como este mapeamento é feito no sistema.

Fragmentos triviais sempre consistem em duas atividades com uma conexão simples entre elas. Para gerar a mensagem, apenas a atividade origem é considerada, pois a atividade destino é analisada no fragmento subsequente como origem, já que o algoritmo percorre os nós de maneira sequencial, através do fluxo definido pelo modelo de processo. Usando a notação do RPST, a atividade em questão pode ser diretamente mapeada para uma árvore DSynt. Para ilustração deste procedimento, considere a primeira atividade do processo de exemplo (Figura 1). Usando a ação *receber* como o verbo principal, o ator *gerente de serviço de quarto* como sujeito e o objeto de negócio *pedido* como substantivo, nós podemos gerar uma árvore DSynt (Figura 5) representando a sentença "Gerente serviço de quarto Receber pedido". Note que a identificação dos objetos do modelo de processo, como verbo (Ação), ator (*Role* da atividade), etc. foi feita na etapa de extração de informação linguísticas (Seção 4.2.1.1).

A transformação de *bonds* é mais complexa. Para demonstrar este procedimento, examinaremos o *bond* B2 de nosso exemplo de processo. Este *bond* começa com um arco XOR rotulado com "*Bebidas alcoólicas pedidas?"*. Baseado nesta anotação, a cláusula condicional "*se bebidas alcóolicas pedidas"* é derivada. A

derivação consiste na adição, ao começo do rótulo do arco, de sentenças condicionais (pois este caso se trata de um arco do tipo XOR) e uso de chaves previamente cadastradas no dicionário do módulo de localização. Esta cláusula então é passada para a primeira atividade do arco *sim*, onde a condição e a cláusula principal são combinadas. Como resultado, uma árvore DSynt, sem considerar regras gramaticais, representando a sentença "Se bebidas alcóolicas pedidas gerente serviço de quarto entregar pedido para *barman*" é gerada (Figura 13). De modo semelhante, podemos realizar a transformação do arco de união. A cláusula de condição de união é então passada para a primeira atividade após o *bond* (*entregar para quarto do hóspede)* e é incorporada de acordo. O procedimento é usado para gerenciar *bonds* de diferentes tamanhos e tipos. No caso de um modelo de processo não prover nenhuma informação sobre a condição de um arco de decisão, utilizamos mensagens elaboradas a partir de um padrão adequado ao contexto. Este padrão é genérico, servindo para qualquer idioma. Posteriormente, ocorre a criação de árvores DSynt para incorporação das respectivas mensagens criadas automaticamente. Dentro de um *bond*, o algoritmo de transformação recursiva é executado de acordo, cobrindo todos os subgrafos previamente definidos.

O processo de geração de texto se baseia na criação de árvores DSynt com *grammemes* (classes que atuam como repositório dos dados morfológicos das palavras analisadas), capturando toda meta-informação necessária. Isso inclui atributos simples, como a classe palavra, mas também recursos mais sofisticados, como tempo e voz do verbo, a definição dos substantivos ou a posição de uma frase condicional (ver atributo ponto de partida na Figura 13).

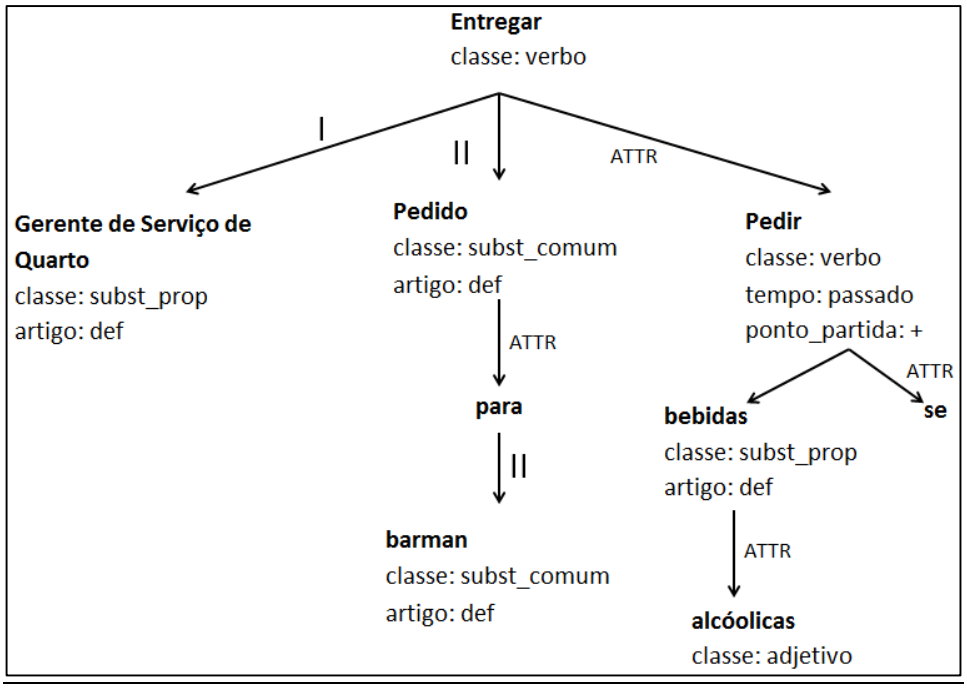


Figura 13 - Árvore DSynt com sentença condicional

4.2.2.2 Refinamento de mensagens

Dentro do componente de refinamento de mensagens, são utilizados algoritmos de Leopold *et al.* [2012] para tratar agregação de mensagens, geração de expressões de referência e inserção de marcadores de discurso.

A necessidade de *agregação de mensagens* normalmente surge quando o processo considerado contém sequências grandes. Neste caso, são utilizadas três técnicas de agregação: agregação por ator, agregação por objeto de negócio e agregação por ação.

Por exemplo, se duas atividades sucessivas são executadas pelo mesmo ator, as mensagens são unidas em uma única sentença. De maneira semelhante, atividades com ações ou objetos de negócio iguais são unidas em uma sentença, caso apareçam em duas atividades sequenciais. Para exemplificar, considere as atividades finais do modelo de processo (Figura 14). O algoritmo de Leopold *et al.* [2012] identifica que: (1) as atividades estão em um mesmo nível da árvore DSynt e estão em uma ordem sequencial; (2) as atividades são executadas pelo mesmo ator, pois se situam na mesma raia do modelo de processo. Após esta identificação, a partir da concatenação das árvores que representam as sentenças, o algoritmo cria uma nova árvore DSynt

para representar ambas as atividades. A nova sentença representada pela árvore seria: “Garçom retornar para o serviço de quarto e Garçom debitar da conta do hospede”.

Se ainda existirem mensagens adjacentes com o mesmo ator após a agregação, a descrição do ator na segunda mensagem é substituída por uma expressão de referência. Por exemplo, vamos utilizar o resultado da agregação das atividades da (Figura 14) exemplificado no parágrafo anterior, “Garçom retornar para o serviço de quarto e Garçom debitar da conta do hospede.”. A segunda ocorrência do ator (garçom) deve ser substituída por uma expressão de referência, neste caso, o pronome pessoal reto “ele”. Teríamos, então, a sentença “Garçom retornar para o serviço de quarto e ele debitar da conta do hospede.”. Note que o texto ainda não está adequado para leitura, considerando as regras gramaticais da língua portuguesa. Este problema será tratado pelo módulo de realização de mensagens.

Para detectar qual pronome seria o mais adequado para o contexto, observa-se o gênero do ator. Caso seja feminino, o pronome “ela” seria utilizado ao invés de sua forma masculina “ele”. Neste trabalho, foi necessária a generalização dos trechos de código que tomavam como premissa que o idioma utilizado no modelo de processo era o inglês. Após a generalização, através da criação de operações nas interfaces definidas no projeto *GeneralLanguageCommon* (Figura 11), suas respectivas implementações para a língua portuguesa foram elaboradas. Dentre elas, destacam-se:

- Verificação se uma dada palavra representa um pronome (Algoritmo 10);
- Identificação do gênero de um dado substantivo (Algoritmo 11);
- Obtenção dos pronomes utilizados pelo idioma (Algoritmo 12).

```
1:   isPronoum(String word)
2:   isPronoun= false;
3:   if LanguageData.pronouns.contains(word) then
4:      isPronoun = true;
5: return isPronoun;
```

Algoritmo 10 - Verificação se uma palavra é um pronome: Operação isPronoum da classe PortugueseLabelHelper.

```
1:   getGender(String noun, Boolean isPlural)
2:   gender = LanguageData.Gender_FEM;
3:   tempNoun = noun.toLowerCase();
4:   if tempNoun.contains(“ ”) then
5:      tempNoun = tempNoun.split(“ ”)[0];
6:   if isPlural then
7:      tempNoun = transformToSingularForm(tempNoun)
8:   if genderMap.containsKey(tempNoun) then
9:      if genderMap.get(tempNoun).equals(“f”) then
10:        gender = LanguageData.Gender_FEM;
11:     else
12:        gender = LanguageData.GENDER_MAS;
13:     return gender;
14:   else
15:      for all femSuffix ∈ LanguageData.feminineSuffixes do
16:         if tempNoun.endsWith(femSuffix) then
17:            return LanguageData.GENDER_FEM;
18:      for all mascSuffix ∈ LanguageData.masculineSuffixes do
19:         if tempNoun.endsWith(mascSuffix) then
20:            return LanguageData.GENDER_MAS;
```

Algoritmo 11 - Identificação do gênero de um dado substantivo: Operação getGender da classe auxiliar NounManager.

```
1:   getPronouns()
2:   if pronouns == ∅ then
3:      pronouns = LanguageData.demonstrativePronouns U
4:      LanguageData.pronouns;
5:   return pronouns;
```

Algoritmo 12 - Obtenção dos pronomes utilizados pelo idioma: Operação getPronouns da classe PortugueseLabelHelper.

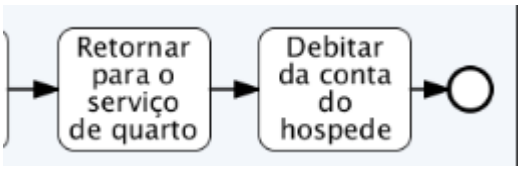


Figura 14 - Atividades finais do modelo de processo do hotel (Figura 1).

Para a introdução de marcadores de discurso, são identificadas mensagens aparecendo em uma sequência estrita. Usando um conjunto de conectores extensíveis (por exemplo: "consequentemente", "em seguida", "finalmente", "paralelamente" etc.), uma palavra adequada é randomicamente inserida. A inserção sempre ocorre no começo de sentenças que estão localizadas no mesmo nível da árvore, ou seja, são atividades sequenciais ou que possuem sua origem em um mesmo arco de decisão. Para ilustrar este caso, considere as atividades finais do modelo de processo (Figura 14). Para fornecer uma ideia de fluxo contínuo no texto, o marcador de discurso "em seguida" é adicionado. A sentença resultante é então "Garçom retornar para o serviço de quarto. Em seguida, ele debitar da conta do hospede. Finalmente, o processo termina.". Cabe lembrar que a frase "o processo termina" foi previamente criada na seção 4.2.2.1, correspondente ao 7º passo do diagrama de sequência (Figura 10), para representação do evento de fim do modelo de processo. Desta maneira, obtemos um texto que pode ser lido facilmente com variações. Nesta etapa, na implementação de Leopold *et al.* [2012] os marcadores de discurso adicionados eram oriundos do idioma inglês. Tais marcadores foram localizados e suas ocorrências substituídas por chaves do dicionário de localização que, em tempo de execução, buscam pela tradução da mensagem representada pela chave para o idioma correto do modelo de processo.

4.2.3 Realização de mensagens

A complexidade da tarefa de realização de mensagens levou ao desenvolvimento de ferramentas disponíveis publicamente, como TG/2 [Busemann, 1996] e RealPro [Lavoie, 1997].

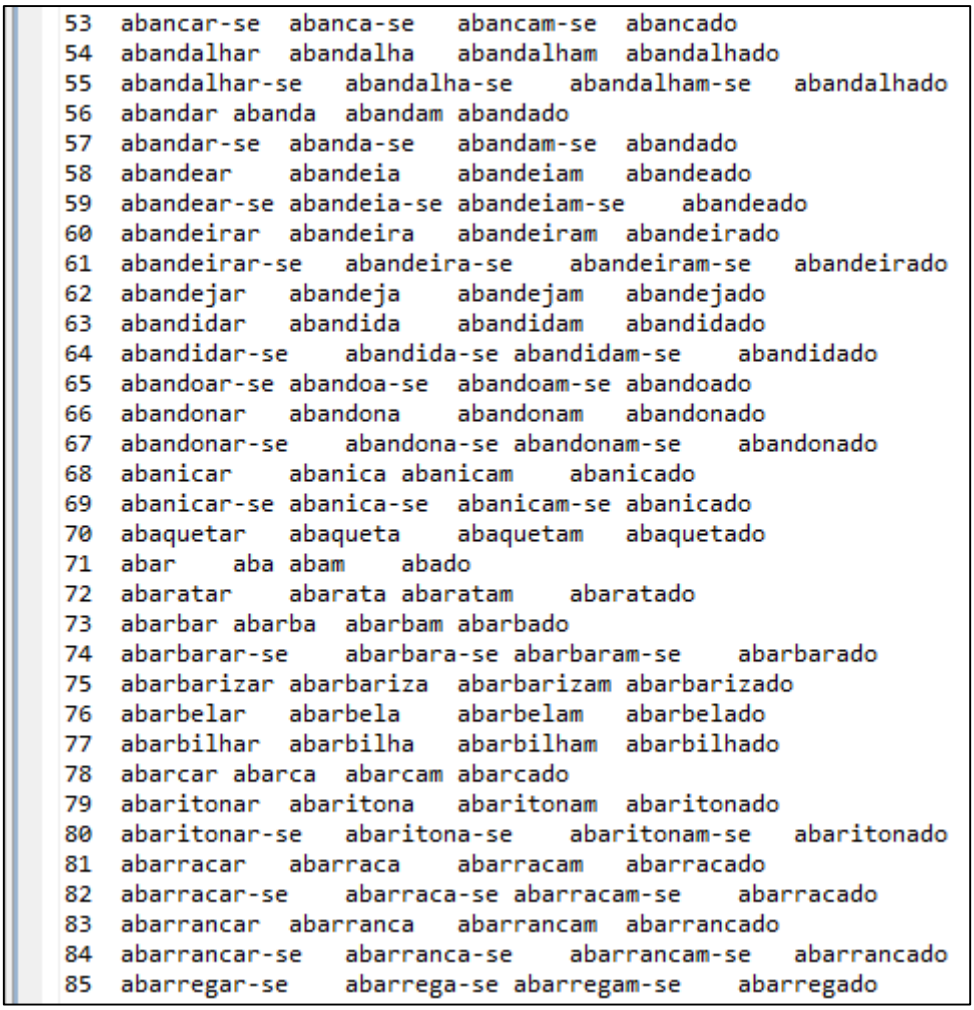
Devido à ausência de uma ferramenta adequada para realização de mensagens na língua portuguesa, algoritmos específicos para tratar as sentenças em português tiveram que ser elaborados.

Tomando como base o módulo implementado por Leopold *et al.* [2012] para a língua inglesa, neste trabalho foram observados os aspectos e operações comuns que seriam gerais para qualquer língua. Com base nessa observação foi criado um mecanismo genérico que aceita a implementação de um módulo de realização de mensagem em uma determinada língua sem que seja necessária a alteração dos demais módulos do sistema (Classe SurfaceRealizer do pacote GeneralLanguageCommon - Figura 11).

Após a identificação das operações necessárias para elaborar o módulo em português, bastou criar os algoritmos necessários para a correta execução das operações.

Primeiramente, foi criado um banco de dados de verbos (cuja estrutura é exemplificada na Figura 15), a partir da leitura de diversos arquivos obtidos através da ferramenta KonjugationsHase [5]da organização alemã Cactus2000 totalizando cerca de 15310 verbos distintos [Cactus2000]. Basicamente, foi feita uma leitura sequencial dos arquivos textos para centralização das informações dos diversos arquivos em um único banco de verbos. Os arquivos utilizados como base para a criação do banco de verbos tem a estrutura apresentada na Figura 16. As conjugações salvas destes verbos foram filtradas levando em consideração às necessidades de descrição de ações em modelos de processos, tais como: presente (3ª pessoa plural e 3ª pessoa singular), infinitivo, particípio e gerúndio [Leopold *et al.*, 2010].

[5] KonjugationsHase é uma ferramenta online que possui um vasto banco de verbos (de diversos idiomas) e oferece funcionalidades para manipulação dos mesmos. Por exemplo, mecanismos de busca e de conjugações do verbo (1ª pessoa presente para qualquer outra, como 3ª pessoa pretérito). Além disso, oferece recursos para exportação de todas as conjugações de um verbo específico, em um formato de texto (.txt).



```
53  abancar-se  abanca-se   abancam-se  abancado
54  abandalhar  abandalha   abandalham  abandalhado
55  abandalhar-se   abandalha-se    abandalham-se   abandalhado
56  abandar abanda  abandam abandado
57  abandar-se  abanda-se   abandam-se  abandado
58  abandear    abandeia    abandeiam   abandeado
59  abandear-se abandeia-se abandeiam-se    abandeado
60  abandeirar  abandeira   abandeiram  abandeirado
61  abandeirar-se   abandeira-se    abandeiram-se   abandeirado
62  abandejar   abandeja    abandejam   abandejado
63  abandidar   abandida    abandidam   abandidado
64  abandidar-se    abandida-se abandidam-se    abandidado
65  abandoar-se abandoa-se  abandoam-se abandoado
66  abandonar   abandona    abandonam   abandonado
67  abandonar-se    abandona-se abandonam-se    abandonado
68  abanicar    abanica abanicam    abanicado
69  abanicar-se abanica-se  abanicam-se abanicado
70  abaquetar   abaqueta    abaquetam   abaquetado
71  abar    aba abam    abado
72  abaratar    abarata abaratam    abaratado
73  abarbar abarba  abarbam abarbado
74  abarbarar-se    abarbara-se abarbaram-se    abarbarado
75  abarbarizar abarbariza  abarbarizam abarbarizado
76  abarbelar   abarbela    abarbelam   abarbelado
77  abarbilhar  abarbilha   abarbilham  abarbilhado
78  abarcar abarca  abarcam abarcado
79  abaritonar  abaritona   abaritonam  abaritonado
80  abaritonar-se   abaritona-se    abaritonam-se   abaritonado
81  abarracar   abarraca    abarracam   abarracado
82  abarracar-se    abarraca-se abarracam-se    abarracado
83  abarrancar  abarranca   abarrancam  abarrancado
84  abarrancar-se   abarranca-se    abarrancam-se   abarrancado
85  abarregar-se    abarrega-se abarregam-se    abarregado
```

Figura 15 – Exemplo da estrutura do arquivo que contém os verbos analisados.

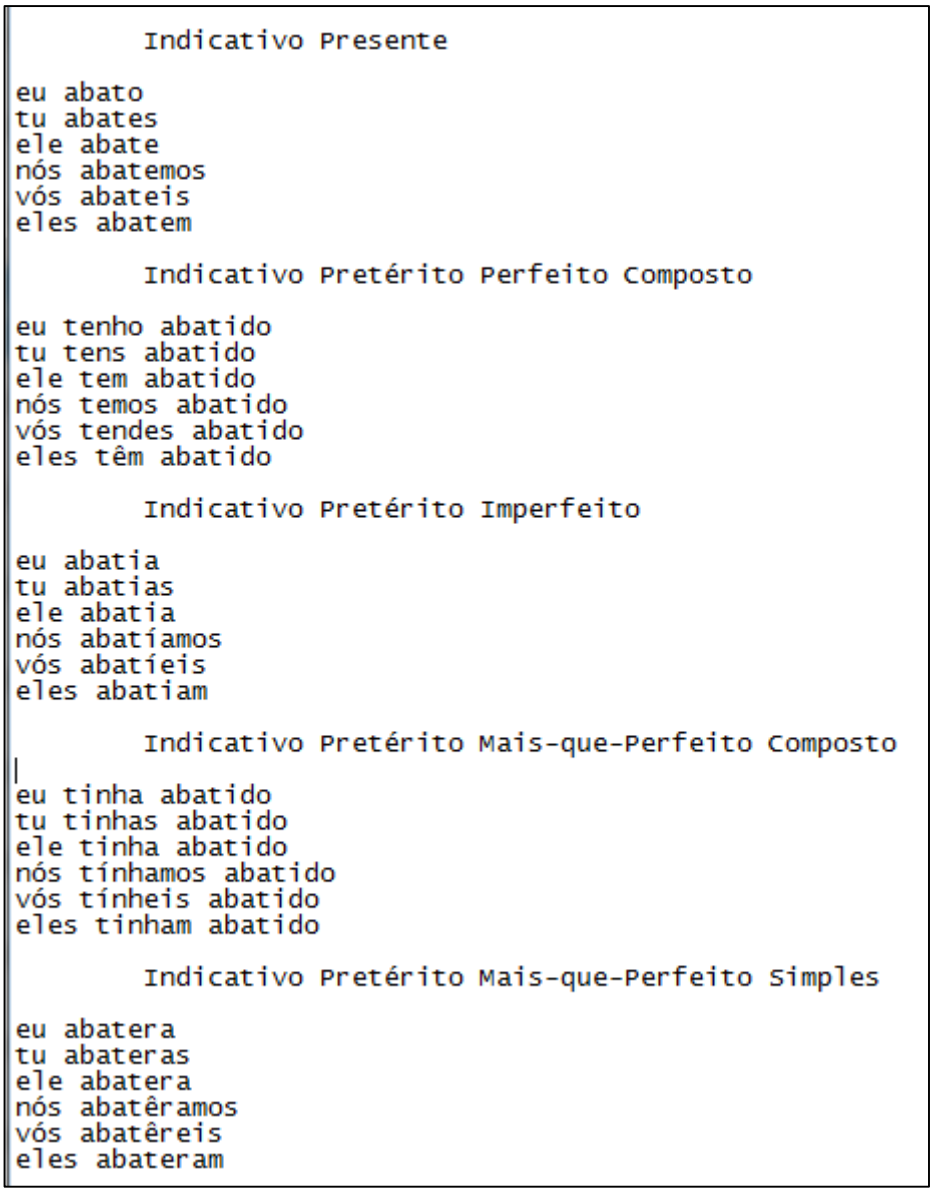


Figura 16 – Estrutura dos arquivos obtidos pela ferramenta KonjugationsHase [Cactus2000].

Em seguida, foi criado um banco de dados para identificação e manipulação das demais funções sintáticas que as palavras de um modelo de processo podem assumir, como: substantivos, adjetivos, pronomes, artigos e advérbios. Estas funções sintáticas podem tomar diferentes formas em um modelo de processo, podem exercer papel de um objeto de negócio, de um ator, expressar uma condição ou um simples complemento. Para criação do banco, foi utilizado o corpus Floresta [Floresta, 2009]. Através do arquivo referente ao corpus, foram extraídas as diversas informações sintáticas das palavras, totalizando cerca de um milhão e 640 mil palavras analisadas. Todas as palavras estavam devidamente categorizadas de acordo com categorias

predefinidas, como: artigo, substantivo, advérbio etc. Ao ler o arquivo, foram feitas pequenas alterações para adaptar o conteúdo necessário em uma forma mais simples e mais apropriada, de modo a obter um uso mais eficiente e direto das palavras. Estruturas de dados como mapas e tabelas *hash* foram empregadas para o mapeamento das palavras com suas respectivas categorias. Quanto maior for o banco de palavras com suas respectivas categorizações, maior será a cobertura dos modelos de processo que servirão de entrada para o algoritmo.

De posse de todos os dados necessários e já estruturados, os métodos necessários para gerar uma mensagem gramaticalmente correta foram elaborados. Alguns métodos se responsabilizam por classificar uma dada palavra arbitrária, extraída do modelo de processo, de acordo com sua respectiva função sintática, bastando para isso, uma busca no banco pela palavra em questão. Para ilustrar estas funcionalidades, considere a atividade "Entregar pedido para garçom" do modelo de processo (Figura 17). As operações de identificação se uma dada palavra é um substantivo ou um verbo já foram chamadas no módulo de extração de informações linguísticas, classificando a palavra "Entregar" como verbo (Ação da atividade) e a palavra "pedido" como substantivo (objeto de negócio da atividade). Porém, o conjunto de palavras restantes foram classificados, simplesmente, como complemento da atividade. Para gerar o texto final, devemos analisar este complemento, identificando as funções sintáticas das palavras e executando o tratamento adequado. Por exemplo, a palavra "Garçom" é identificada como um substantivo; logo, o texto final deve conter o artigo correto que se relaciona com o mesmo. Para obtenção do artigo correto, uma operação responsável por retornar um artigo correto para um dado substantivo é chamada (Algoritmo 20). As operações que representam tais funcionalidades são:

- Verificação se uma dada palavra é um advérbio (Algoritmo 13);
- Verificação se uma palavra é um verbo (Algoritmo 1);
- Verificação se uma palavra é um substantivo (Algoritmo 2);
- Verificação se uma palavra é um adjetivo (Algoritmo 14).

```
1:   isAdverb(String word)
2:   isAdverb = false;
3:   if getPos(word) == PortugueseLabelHelper.POS_ADV then
4:      isAdverb = true;
5: return isAdverb;
```

Algoritmo 13 - Verificação se uma palavra é um advérbio: Operação isAdverb da classe PortugueseLabelHelper.

```
1:   isAdjective(String word)
2:   isAdjective = false;
3:   if getPos(word) == PortugueseLabelHelper.POS_ADJ then
4:      isAdjective = true;
5: return isAdjective;
```

Algoritmo 14 - Verificação se uma palavra é um adjetivo: Operação isAdjective da classe PortugueseLabelHelper.

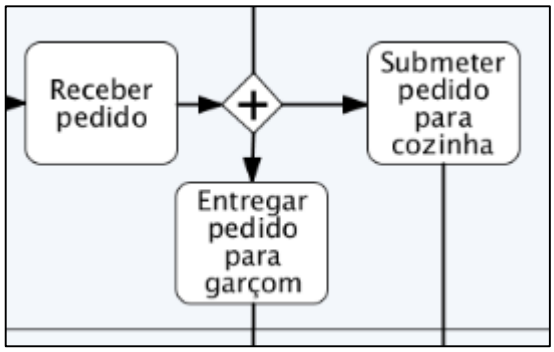


Figura 17 – Destaque de algumas atividades do modelo de processo

Outros métodos detêm a responsabilidade de transformar uma dada palavra em uma determina forma, por exemplo, transformação de um verbo que representa uma ação extraída de uma atividade do modelo no infinitivo por sua respectiva conjugação no presente da 3ª pessoa do singular. Outro exemplo é a transformação de gênero e forma de um substantivo, onde ocorre transformação de plural por sua respetiva forma em singular e vice-versa, bem como a mudança de gênero quando esta se faz necessária. A estratégia utilizada para transformação é basicamente um mapeamento elaborado previamente. Por exemplo, ao obter um verbo do banco, suas diversas conjugações são mapeadas para a sua forma no infinitivo. De forma análoga,

um substantivo é mapeado para sua forma no singular. Ilustrando estas funcionalidades, considere a atividade "Preparar nota" (Figura 18). O verbo "Preparar", já em sua forma no infinitivo, é fornecido para a operação responsável por retornar sua forma no presente da 3ª pessoa do singular "Prepara". Os métodos relacionados são:

- Obtenção da forma infinitiva do verbo (Algoritmo 15);
- Obtenção do particípio do verbo (Algoritmo 16);
- Obtenção da forma em presente do verbo (Algoritmo 17);
- Verificação se um verbo está na 3ª pessoa do singular (Algoritmo 18);
- Transformação de um substantivo para sua forma singular (Algoritmo 19).

Figura 18 – Atividade "Preparar nota" destacada do modelo de processo.

```
1:   getInfinitive(String conjugatedVerb)
2:   infinitiveForm = conjugatedVerb;
3:   if present3PP.containsValue(conjugatedVerb) then
4:      infinitiveForm = lookForKey(present3PP, conjugatedVerb)
5:   else if present3PS.containsValue(conjugatedVerb) then
6:         infinitiveForm = lookForKey(participle, conjugatedVerb)
7:   return infinitiveForm;
```

* **present3PP e present3PS:** Foi explicado na seção 4.2.3 que os verbos são obtidos e mapeados através de tabelas hash (present3PP, por exemplo), cuja chave é o verbo no infinitivo e o seu valor a conjugação no tempo e pessoa representados pela própria tabela (present3PP = presente 3ª pessoa do plural).

Algoritmo 15 - Obtenção da forma infinitiva do verbo: Operação getInfinitive da classe auxiliar VerbManager.

```
1:   getParticipleForm(String infinitive)
2:   participleForm = participle.get(infinitive.toLowerCase())
3:   if participleForm == ∅ then
4:      stemmedVerb = stemPrefix(infinitive)
5:      if stemmedVerb != ∅ then
6:         participleForm  = prefix + getParticipleForm(stemmedVerb)
7:   else
8:     participleForm = infinitive;
9:  return participleForm;
```

Algoritmo 16 - Obtenção do particípio do verbo: Operação getParticipleForm da classe auxiliar VerbManager.

```
1:   getPresentForm(String infinitive, Boolean bo_isSubject, Boolean
     bo_isPlural)
2:   presentForm = ∅;
3:   if bo_isSubject and bo_isPlural then
4:      presentForm = present3PP.get(infinitive.toLowerCase())
5:   else
6:      presentForm  = present3PS.get(infinitive.toLowerCase())
7:   if presentForm == ∅ then
8:      stemmedVerb = stemPrefix(infinitive)
9:      if stemmedVerb != ∅ then
10:         presentForm = prefix + getPresentForm(stemmedVerb)
11:   if presentForm == ∅ then
12:     presentForm = infinitive;
13:  return presentForm;
```

Algoritmo 17 - Obtenção da forma em presente do verbo: Operação getPresentForm da classe auxiliar VerbManager.

```
1:   is3PS(String word)
2:   is3PS = false;
3:   if present3PS.containsValue(word) then
4:      is3PS = true;
5: return is3PS;
```

Algoritmo 18 - Verificação se um verbo está na 3ª pessoa do singular: Operação is3PS da classe auxiliar VerbManager.

```
1:   transformToSingularForm(String pluralNoun)
2:   for all pluralSuffix ∈ pluralSinguarMap.keySet do
3:   if pluralNoun.endsWith(pluralSuffix) then
4:      pluralNoun = pluralNoun.replace(pluralSuffix,
       pluralSinguarMap.get(pluralSuffix));
5: return pluralNoun;
```

Algoritmo 19 - Transformação de um substantivo para sua forma no singular: Operação getSingularOfNoun da classe PortugueseLabelHelper.

Baseando-se nestes métodos, para cada DSynt (mensagem encapsulada do modelo de processo) recebida pelo módulo anterior, adiciona-se os artigos corretos aos respectivos substantivos, conjuga-se o verbo na forma adequada de acordo com o contexto e com o substantivo que está relacionado. Além disso, adiciona-se os conectivos necessários, como vírgulas, espaços e capitalizações de palavras. Como exemplo, considere a atividade "Preparar nota" do modelo (Figura 18). Ao receber a DSynt que a representa, uma série de operações são chamadas. A primeira é responsável por identificar o gênero do ator (*role* da atividade) e do objeto de negócio, identificando-os como sendo do gênero masculino e feminino, respectivamente. Em seguida, a operação para obtenção do artigo específico para o gênero e grau é chamada, retornando os artigos definidos "o" e "a". Para o tratamento adequado da ação, a operação responsável pela conjugação correta do verbo (Algoritmo 17) é chamada, retornando o verbo na forma da 3ª pessoa do singular "Prepara". Alguns dos métodos relacionados com esta etapa são:

- Obtenção do gênero de um dado substantivo (Algoritmo 11);
- Obtenção do artigo correto para um dado substantivo (Algoritmo 20).

```
1:   getArticle(String noun, Boolean isObject, Boolean isIndef,
     Boolean isPlural)
2:   article = ø;
3:   articleIndex = getGender(noun, isPlural)
4:   if isPlural then
5:      articleIndex++;
6:   if isObject and isIndef then
7:      article = LanguageData.indefArticles[articleIndex];
8:   else
9:      article = LanguageData.defArticles[articleIndex];
10:  return article;
```

Algoritmo 20 - Obtenção do artigo correto para um dado substantivo: Operação getArticle da classe PortugueseLabelHelper.

4.3 Resumo do capítulo

O presente capítulo abordou os aspectos tecnológicos envolvidos neste trabalho. Foram apresentadas as técnicas para a criação de um *framework* baseado nos conceitos envolvidos com as diversas etapas presentes no pipeline GLN. Destacaram-se as operações, algoritmos, classes e objetos, sequência em que as operações são chamadas durante a execução da aplicação e estruturas de dados utilizadas para a representação de artefatos como um modelo de processo em memória.

O capítulo seguinte tem como objetivo avaliar o framework quanto à capacidade de generalização e execução, destacando os resultados obtidos e relatando o seu comportamento diante de cenários distintos.

Capítulo 5: AVALIAÇÃO DA PROPOSTA

Este capítulo apresenta a avaliação dos resultados da execução da ferramenta de acordo com alguns exemplos confeccionados especificamente para o teste da mesma. Foram elaborados 10 modelos de processos, 5 para inglês e 5 para português, empregando a notação BPMN (Anexo IV). Seu comportamento foi avaliado e observamos como o sistema se comporta diante dos seguintes tipos de modelos de processos: triviais (atividades sequenciais sem nenhum tipo de gateway), intermediários (atividades com gateways de decisão exclusiva e inclusiva) e avançados (atividades com gateways de decisão, de união e paralelos).

Os modelos de processo intermediários e avançados também contam com a variação de algumas características importantes para a avaliação, as quais são:

- Atividades sendo executadas em sequência por um mesmo ator. Estas atividades, ao serem detectadas pelo *framework*, deveriam ativar o módulo de inserção de expressões de referência e o módulo de agregação de mensagens, definidos na seção 4.2.2.2. Por este motivo, a leitura de modelos de processo com tais atividades é essencial para a avaliação;
- Atividades definidas sem atores (raia vazia). Estas atividades se justificam pela necessidade de testar se o algoritmo responsável por tratar atividades sem atores mapeia corretamente a descrição da atividade para a voz passiva.

5.1 Avaliação do *framework*

Foram elaborados 5 modelos para o idioma inglês e 5 outros modelos equivalentes para o idioma português. Deste modo, pode-se testar se o *framework* é capaz de lidar com a mudança dinâmica de idioma do modelo de processo adequadamente em tempo de execução, inicializando as configurações necessárias para tratar o idioma do modelo atual automaticamente, de acordo com o atributo que define o idioma do modelo.

O *framework* teve uma avaliação geral positiva, gerando corretamente o texto em linguagem natural para todos os modelos de processo que serviram como entrada. Soube tratar e mapear corretamente todos os componentes presentes nos modelos, bem como suas respectivas notações e rótulos utilizados para sua descrição. O processo de pipeline GLN (Figura 7) foi eficientemente seguido, sem a detecção de nenhum erro que comprometesse a execução do processo. Porém, alguns problemas foram detectados:

1. O algoritmo responsável por ativar os módulos de agregação de mensagens e de inserção de expressões de referência (seção 4.2.2.2) não obteve sucesso em identificar mensagens sequenciais realizadas pelo mesmo ator, e, consequentemente, não acionou as devidas operações responsáveis por tratar tais atividades. Por exemplo, para o Modelo de Processo 6, utilizado como entrada podemos notar na saída produzida (Resultado Execução 1) que a palavra "garçom" é utilizada frequentemente no último parágrafo. Neste caso, o módulo de inserção de expressão de referência deveria ter substituído a palavra "garçom" pelo pronome "ele";
2. A cobertura atual de modelos de processo para o idioma português é menor do que para o inglês. Tal característica resulta do fato de que o idioma inglês é mais utilizado no mundo do que o português, e consequentemente, há mais recursos para manipulação linguística. Por exemplo, alguns recursos existentes para o inglês, que não foram encontrados para o português, são:
    a. Transformação da voz passiva para a voz ativa e vice versa;
    b. Transformação de uma sentença afirmativa para uma condicional;
    c. Identificação se uma palavra representa uma pessoa ou um simples objeto;
    d. Centralização de diversas operações linguísticas através de um único framework responsável por identificar as palavras (WordNet);
    e. Grande cobertura do vocabulário existente para o idioma, contando com palavras e termos técnicos de diversas áreas e domínios (medicina, engenharia, matemática, etc.).

Cabe ressaltar que o 1º problema identificado não afeta o correto entendimento do texto gerado para representação do respectivo modelo de processo: afeta somente a qualidade do texto gerado (vide Resultado Execução 6), já que a leitura de uma única sentença, que não seja demasiada grande, contendo diversas atividades ao invés de diversas sentenças sequenciais, se torna mais agradável e menos cansativa aos olhos do leitor. Como se pode observar no texto gerado em linguagem natural para representação do Modelo de Processo 6, é possível entender o texto perfeitamente e, consequentemente, entender toda a informação que o modelo transmite.

5.2 Avaliação da generalização para tratar novos idiomas

Outro aspecto levado em consideração para a avaliação do *framework* foi em relação a capacidade de extensão para novos idiomas, avaliando se o processo necessário para inclusão de um novo idioma para tratar modelos de processos é simples, rápido e claro.

A avaliação quanto ao suporte para novos idiomas levou em consideração a extensão do framework para tratar o idioma português. Os passos necessários para tratar idiomas distintos são apresentados a seguir.

Com a generalização do *framework* de modo que este suporte novos idiomas, abstraiu-se muitos aspectos complexos que são comuns a todos como, por exemplo, toda a fase de "Planejamento do Texto", (seção 4.2.1) e de "Planejamento de Sentenças", (seção 4.2.2) e que, consequentemente, não precisam ser modificados. Isto agiliza o processo e direciona o foco para implementação das operações definidas nas interfaces que tratam especificidades de cada idioma.

Notou-se, porém, que a qualidade do texto produzido é diretamente dependente da qualidade das operações implementadas para o idioma em questão. Como a língua portuguesa não possui uma biblioteca padrão para manipulação na linguagem Java, como a conhecida WordNet para o inglês, o texto produzido em português em comparação com o inglês, possui uma qualidade inferior e precisa de melhorias futuras. Para adição de suporte a um novo idioma, os seguintes passos devem ser executados:

1. Criação de um projeto responsável pela implementação das interfaces, bem como as classes auxiliares (internas ao projeto) que o desenvolvedor

julgue necessário. As classes auxiliares devem implementar as operações necessárias para manipulação do corpus do idioma. Neste trabalho, por exemplo, foram definidas classes para acessar o banco de verbos e de palavras do idioma português, enquanto que nenhuma classe adicional foi criada para o inglês, já que este conta com um completo framework de manipulação linguística (WordNet). Nenhuma classe precisa ser modificada ou criada no projeto principal (com exceção da classe de configuração dos idiomas, explicada mais adiante no 2º passo). As interfaces e classes que precisam ser implementadas (ou no caso do SurfaceRealizer, estendida) neste novo projeto são:

   a. ILabelDeriver
   b. ILabelHelper
   c. ILabelProperties
   d. SurfaceRealizer

   Neste *framework*, o projeto "RealizerPort", correspondente ao pacote "PortugueseRealizer" (Figura 11), foi elaborado para manipulação de modelos de processos de negócio confeccionados no idioma português;

2. Extensão do módulo de localização através da adição de um dicionário contendo as respectivas traduções para as chaves definidas no mesmo. Para isso, basta adicionar um arquivo texto com estrutura que se assemelhe a apresentada na Figura 19 contendo as chaves e suas traduções para o idioma em questão, no diretório do módulo de localização. O arquivo deve, por padrão, ter nome na estrutura "Dictionary_NomeIdiomaEmIngles", por exemplo, "Dictionary_Portuguese". O módulo busca, em tempo de execução, pelo dicionário do idioma do modelo de processo e carrega as traduções das chaves definidas para futura utilização durante o processo de geração de texto em linguagem natural a partir do modelo;
3. Adição de referência ao novo projeto e/ou biblioteca responsável pela implementação das operações e extensão das classes definidas no projeto GeneralLanguageCommon (Figura 11) no método "SetCurrentLanguage" na classe LanguageConfig do projeto principal. Uma melhoria futura é a leitura das configurações e referências através de um arquivo externo em .xml ou em .txt, através de um mecanismo de

injeção de dependências. Desta maneira, não será necessário o conhecimento desta classe do projeto principal, responsável pela execução do processo de geração de texto em linguagem natural.

4. Considerando o pipeline GLN (Figura 9), a única etapa que precisa ser elaborada para um novo idioma é a etapa de realização de mensagens, cuja explicação técnica encontra-se na seção 4.2.3. As demais etapas e componentes não precisam ser modificados. Todos os algoritmos necessários para a correta execução das macro-etapas de "Planejamento do Texto" (4.2.1) e "Planejamento de Sentenças" (4.2.2) são utilizados através das interfaces definidas no pacote LabelAnalysis do projeto genérico GeneralLanguageCommon (Figura 11);

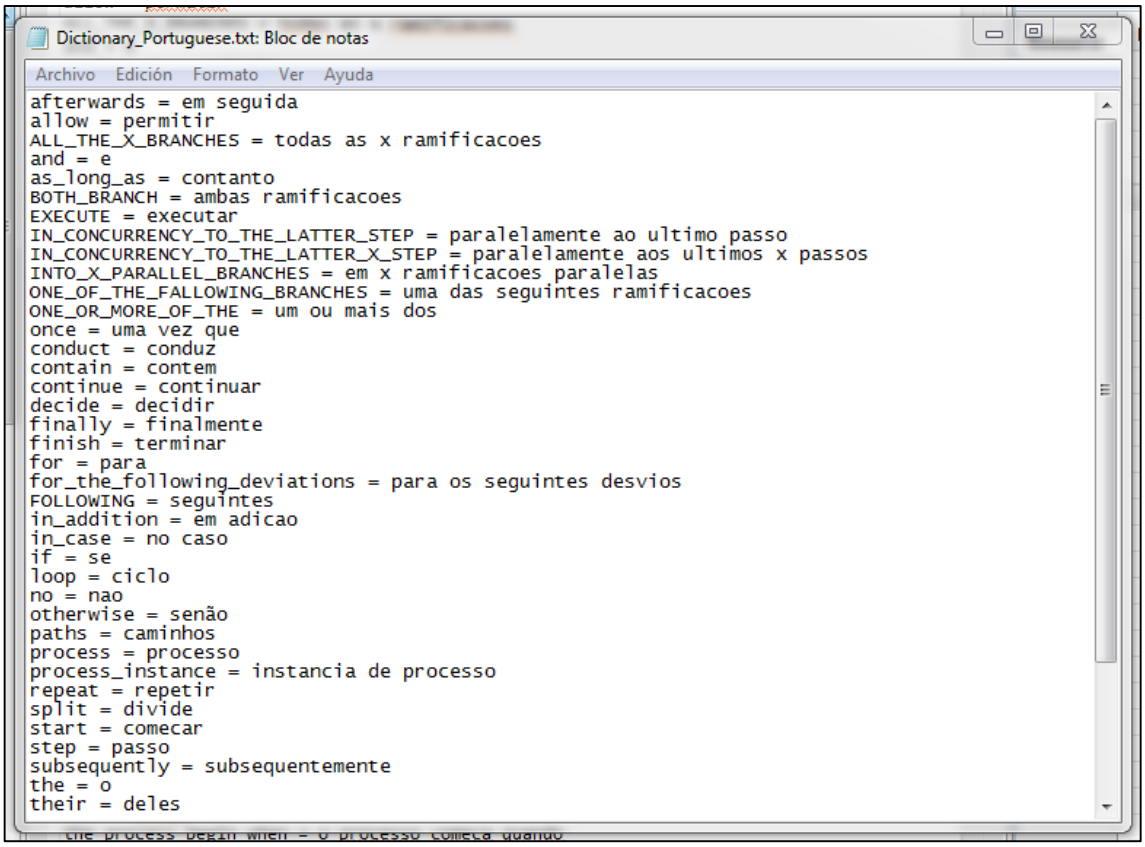


Figura 19 – Estrutura do arquivo que representa o dicionário de um determinado idioma. Neste caso, o português.

Capítulo 6: CONCLUSÃO

Modelos de processos são utilizados frequentemente em diversas organizações para compreensão, documentação e visualização das tarefas realizadas pela organização. Através da abordagem de geração de texto em linguagem natural, é possível que usuários não técnicos sejam capazes de conhecer e entender os processos organizacionais sem a necessidade da compreensão de notação utilizada para elaboração do modelo que os representam. Dessa forma, o foco está na informação a ser transmitidas pelos modelos de processos, a qual também pode ser lida textualmente.

Este trabalho apresentou a generalização do framework elaborado por Leopold *et al.* [2012] para ser capaz de tratar novos idiomas derivados do latim para geração de texto em linguagem natural a partir de modelos de processos em BPMN. Dessa forma, um conjunto de interfaces foram implementadas, cujas operações foram apresentadas e documentadas. Além disso, a fim de avaliar a generalização desenvolvida, foram implementadas as extensões necessárias para tratar o idioma português.

O *framework* obteve resultados positivos quanto aos aspectos de generalização dos componentes para aceitar a adição de um novo idioma como o de geração dos textos em linguagem natural. Porém, alguns problemas ainda devem ser corrigidos e um maior esforço para melhorar a cobertura do idioma português deve ser dedicado. O *framework* gerou textos com boa qualidade de leitura, elaborados corretamente segundo as regras gramaticais do idioma do modelo de processo para ambas as línguas, inglês e português. O processo de adição de suporte a um novo idioma é bem definido e, a princípio, não requer alteração nos principais componentes da aplicação.

Como trabalho futuro, sugerimos o uso da ferramenta em um cenário real de empresa que possua modelos de processos desenhados e que tenha usuários técnicos e não técnicos lendo estes modelos. Dessa forma, questionários poderiam ser aplicados a estes usuários a fim de avaliar se o texto gerado a partir do modelo foi suficiente para o entendimento do processo.

Também sugerimos incluir novos idiomas na ferramenta. Para isto, é necessária a implementação das interfaces específicas para o tratamento de um novo idioma, definidas no pacote "GeneralLanguageCommon" (Figura 11). Um idioma indicado para este teste seria o alemão ou espanhol.

Outra melhoria a ser elaborada é a ampliação da cobertura de elementos da notação BPMN, que atualmente consiste apenas no conjunto de elementos apresentados na Tabela 1.

Atualmente, o framework é baseado em uma interface de console simples, que não permite uma interação mais dinâmica e uma experiência visual avançada para os usuários. Sugere-se a adição de uma interface gráfica, que permita a definição do modelo de processo a ser processado, bem como o arquivo correspondente ao dicionário do idioma considerado e o arquivo de configuração dos idiomas.

Além disso, como trabalho futuro, sugerimos a expansão do vocabulário português suportado pelo módulo responsável em gerar textos no idioma. Para isso, é necessária a ampliação do banco de palavras através do cadastramento de novo corpus da língua portuguesa, bem como a leitura e armazenamento de informações linguísticas contidas em modelos de processos. Desta maneira, será possível aumentar a cobertura de modelos que a ferramenta é capaz de tratar com sucesso. Com a cobertura atual, algumas palavras utilizadas no modelo de processo, não são reconhecidas. Neste caso, alguns problemas como a adição de artigos errados (artigos femininos a substantivos masculinos, por exemplo) e a ausência da conjugação adequada do verbo, podem ocorrer. Aspectos como a desambiguação de palavras também não estão sendo tratados.

Capítulo 7: REFERÊNCIAS BIBLIOGRÁFICAS

Advancing Open Standard for the Information Society (2007). Disponível em <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Acessado em 06 de Janeiro de 2013.

BUSEMANN S.: **Best-first surface realization. Interface** (1996), pp. 10

CACTUS2000: Disponível em <http://www.cactus2000.de/uk/index.php>. Acessado em 05 de Dezembro de 2012.

DALIANIS, H.: **Aggregation in natural language generation**. Computational Intelligence 15(4) (1999) pp. 384-414

DAVIES, I.; GREEN, N.; ROSEMANN, M.; INDULSKA, M.; GALLO, S. **How do practitioners use conceptual modeling in practice** (2004).

DIXON, R.: **Deriving Verbs in English**, Language Sciences (2008).

EPC: Disponível em <http://www.ariscommunity.com/event-driven-process-chain>. Acessado em 06 de Janeiro de 2013.

FLORESTA: **Floresta corpus (2009)**. Disponível em <http://www.linguateca.pt/floresta/corpus.html>. Acessado em 02 de Janeiro de 2013.

FREDERIKS, P., WEIDE, T.: **Information modeling: The process and the required competencies of its participants** (2004).

GALLEY, M., FOSLER-LUSSIER, E., Potamianos, A.: **Hybrid natural language generation for spoken dialogue systems** (2001)

JSON (2013), **Java Script Object Notation.** Disponível em < http://www.json.org/>. Acessado em 14 de maio de 2013.

JURAFSKY, D., MARTIN J.H.: **Speech and Language Processing** (1999).

KAUTZ, K.; HANSEN, B.; JACOBSEN, D. **The utilization of information systems development methodologies in practice, Journal of Information Technology, Cases and Applications** (2004).

LARMAN, C. **Utilizando UML e padrões – Uma introdução a análise e ao projeto orientado.** 3 ed. Bookman, 2007.

LAVOIE, B.; RAMBOW, O. **A fast and portable realizer for text generation systems**. Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLC '97). Stroudsburg, PA, pp. 265–268, 1997

LEOPOLD, H., MENDLING, J., POLYVYANYY, A.: **Generating Natural Language Texts from Business Process Models** (2012). In: CAiSE 2012.

LEOPOLD, H., SMIRNOV, S., MENDLING, J.: **Recognising Activity Labeling Styles in Business Process Models**. EMISA 6(1) (2011) pp. 16-29

LEOPOLD, H., SMIRNOV, S., MENDLING, J.: **Refactoring of Process Model Activity Labels**. In: NLDB 2010. Volume 6177 of LNCS., Springer (2010) pp. 268 - 276

KONCHADY M.: **Text Mining Application Programming.** Cengage Learning, 1ª ed. (2006)**.**

MEL'CUK, I., POLGUÈRE, A.: **A formal lexicon in the meaning-text theory (or how to do lexica with words)**. Computational Linguistics (1987)

METEER, M.W.: **Expressibility and the Problem of Efficient Text Planning**. St.

MILLER, G. A. **WordNet: a Lexical Database for English** (1995).

OVCHINNKOVA, E.: **Integration of World Knowledge for Natural Language Understanding**, Atlantis Press. (2012).

OMG, **Business Process Management Initiative** (BPMI). Object Management Group, (2004). Disponível em <http://www.omg.org/>. Acessado em 06 de Janeiro de 2013.

OMG: **Business process model and notation (BPMN) version 2.0 (2011)**. Disponível em <http://www.omg.org/spec/BPMN/2.0/>. Acessado em 05 de Março de 2013.

POLYVYANYY, A., VANHATALO, J., VÖLZER, H.: **Simplified computation and generalization of the refined process structure tree**. In: Web Services and Formal Methods. Volume 6551 of LNCS. Springer (2011) pp. 25-41

PORTET F., REITER E, GATT A., HUNTER J., SRIPADA S., FREER Y., SYKES C.: **Automatic Generation of Textual Summaries from Neonatal Intensive Care Data.** Artificial Intelligence 173 (7–8): 789–816. doi:10.1016/j.artint. (2008).

RADEMAKER A., PAIVA V.: **Revisiting a Brazilian WordNet.** In: Proceedings of Global Wordnet Conference (2012).

REITER, E.: **NlG vs. templates**. In: In Proceedings of the Fifth European Workshop on Natural Language Generation. (1995) pp. 95-106

REITER, E., DALE, R. **Building Natural Language Generation Systems.** 1 ed. Cambridge University Press, 2000.

REITER, E., DALE, R.: **Building applied natural language generation systems**. Natural Language English 3 (March 1997) pp. 57-87

REITER, E., MELLISH, C.: **Optimizing the costs and benefits of natural language generation**. In: IJCAI. (1993) pp. 1164-1171

REITER, E., MELLISH, C., LEVINE, J., BRIDGE, S.: **Automatic generation of on-line documentation in the idas project**. In: ANLP. (1992) pp. 64-71

STEDE, M.: **Lexicalization in natural language generation: A survey**. Artificial

Intelligence Review 8 (1994) pp. 309-336

WordNet (2013): **WordNet A Lexical database for English.** Disponível em < http://wordnet.princeton.edu/>. Acessado em 14 de maio de 2013.

WordNet.PT (2013): WordNet, Rede Léxico-Conceptual do Português. Disponível em < http://www.clul.ul.pt/clg/wordnetpt/index.html>. Acessado em 14 de maior de 2013.

WS-BPEL, **Web Services Business Process Execution Language Version 2.0**.

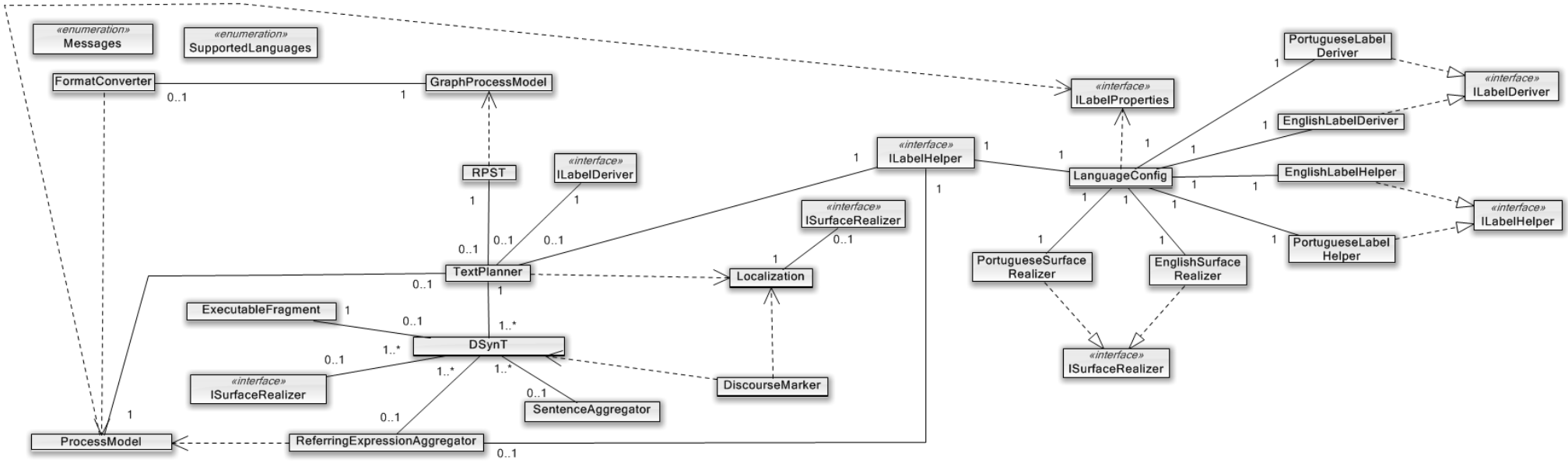
UML (2012): Disponível em <http://www.uml-diagrams.org/activity-diagrams.html>. Acessado em 14 de Janeiro de 2013.

VANHATALO, J., VÖLZER, H., KOEHLER, J.: **The refined process structure tree**. Data & Knowledge Engineering 68(9) (2009) pp. 793-818

VAN DEEMTER, K., KRAHMER, E., THEUNE, M.: **Real versus Template-Based Natural Language Generation: A False Opposition?** Computer Linguistics (2005) pp. 15 -24

YAML (2007): “**Yet Another Wrkflow Language (YAML)**”. Disponível em <http://www.yawl-system.com/>. Acessado em 14 de Janeiro de 2013.

Anexo I – Diagrama de classes simplificado



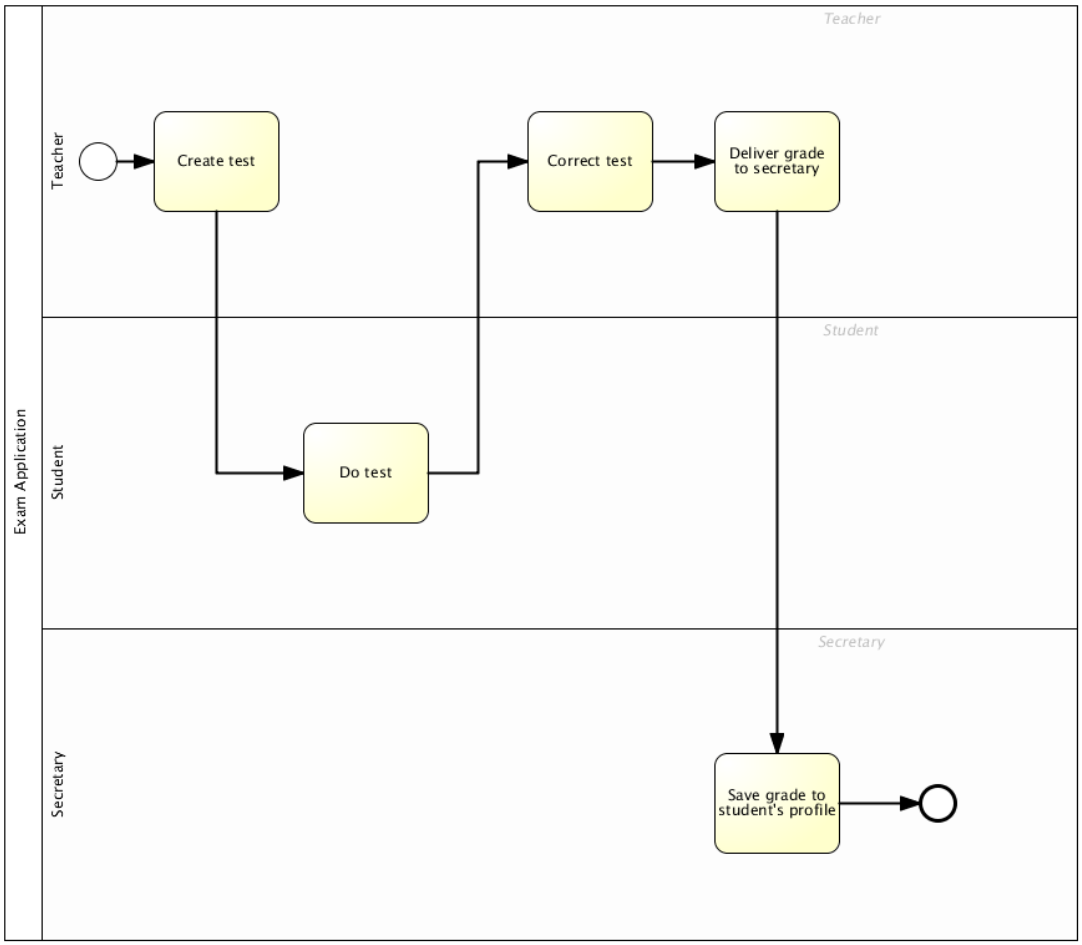
Anexo II – Classe auxiliar para armazenamento de algumas propriedades linguísticas do português.

```
1:   static class LanguageData
2:   masculineSuffixes = {"o","e","al","il","im"};
3:   feminineSuffixes = {"a","ão","er","iz"};
4:   verbalPrefixes = {"re","des","i"};
5:   articlePrepositions  = {"após","ate","com","conforme",
     "contra","desde", "durante","entre","mediante","para",
     "perante","por","sob","sobre","sem","via"};
6:   defArticles = {"o", "os","a","as"};
7:   indefArticles = {"um","uns","uma","umas"};
```

Anexo III – Derivação de Ação e objeto de negócio – Operação deriveActionAndBusinessObject da classe PortugueseLabelDeriver

```
1:  deriveActionAndBusinessObject(Label label, LabelProperties
    prop)
2:  if !prop.hasConjunctions then
3:    size=label.words.size;
4:    if prop.style==NOUN then
5:      if prop.hasPrepositions then
6:        label=label.words[1]+···+label.words[prop.pIndex−1];
7:      if size==1 then
8:        action=label.words[1];
9:      else
10:       if label.words[size−1]+label.words[size] is a phrasal verb
          then
11:         prop.action=label.words[size−1] + label.words[size];
12:         if size>2 then
13:           prop.bObject=label.words[1]+···+label.words[size−2];
14:       else
15:         prop.action=label.words[size];
16:         prop.bObject=label.words[1]+···+label.words[size−1];
17:   else if prop.style==PREPOSITION_OF then
18:     prop.action=label.words[1]+···+label.words[pIndex−1];
19:     pPhrase=label.words[pIndex+1]+···+label.words[size];
20:     if pPhrase contains prepositions then
21:       nIndex=index of the next preposition after pIndex;
22:       prop.bObject=label.words[pIndex+1]
          +···+label.words[nIndex−1];
23:     else
24:       prop.bObject=label.words[pIndex+1]
          +···+label.words[size];
25:   else if prop.style==GERUND then
26:     if prop.hasPrepositions then
27:       label=label.words[1]+···+label.words[prop.pIndex−1];
28:     if size==1 then
29:       action=label.words[1];
30:     else
31:       if label.words[1]+label.words[2] is a phrasal verb then
32:         prop.action=label.words[1]+label.words[2];
33:         if size>2 then
34:           prop.bObject=label.words[3]+···+label.words[size];
35:       else
36:         prop.action=label.words[1];
37:         prop.bObject=label.words[2]+···+label.words[size];
38:   else if prop.style==DESCRIPTIVE then
39:     for i=1→size do
40:       if label.words[i].getTag()==VERB then
41:         prop.action=label.words[i];
42:         break;
43:     prop.bObject=label.words[i+1]+···+label.words[size];
44: return prop;
```

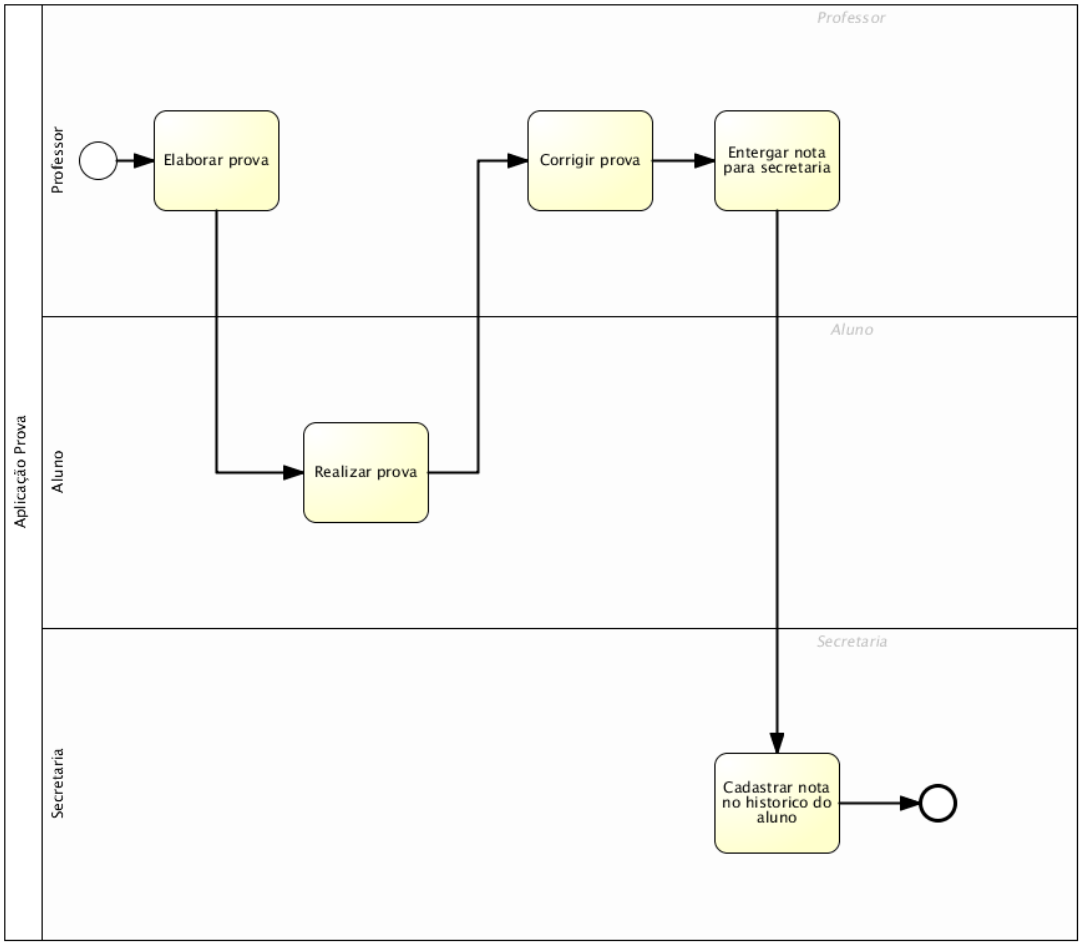
Anexo IV – Modelos utilizados para fins de avaliação do framework e os respectivos resultados obtidos como saída.



Modelo de Processo 1 – Modelo de processo trivial em inglês.

The process begins when the Teacher creates test. Then, the Student does test. Afterwards, the Teacher corrects test. Subsequently, the Teacher delivers grade to the secretary. Then, the Secretary saves grade to the student's profile. Finally, the process is finished.

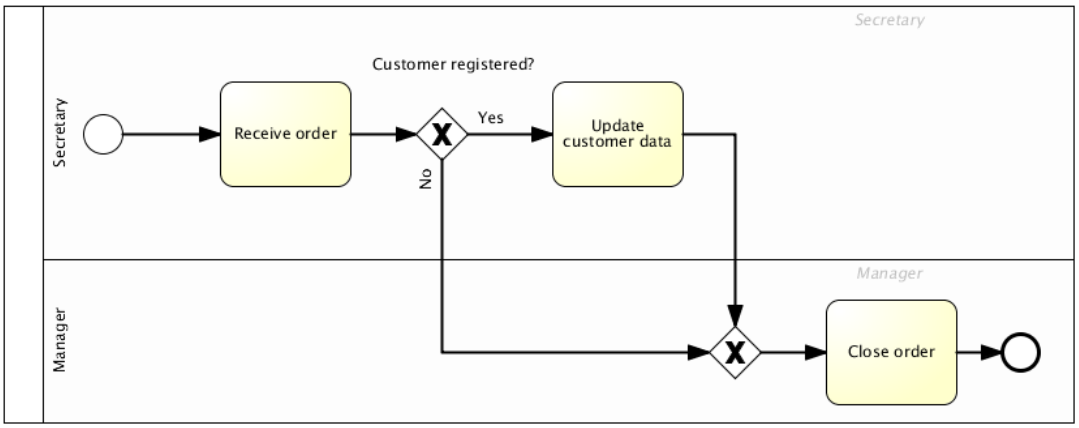
Resultado Execução 2 – Texto em linguagem natural gerado para o Modelo de Processo 1.



Modelo de Processo 2 – Modelo de processo trivial, em português.

O processo começa quando o Professor elabora a prova. Então, o aluno realiza a prova. Em seguida, o professor corrige a prova. Subsequentemente, o professor conduz a atividade de entregar nota para secretaria. Então, a secretaria cadastra a nota no histórico do aluno. Finalmente, o processo é terminado.

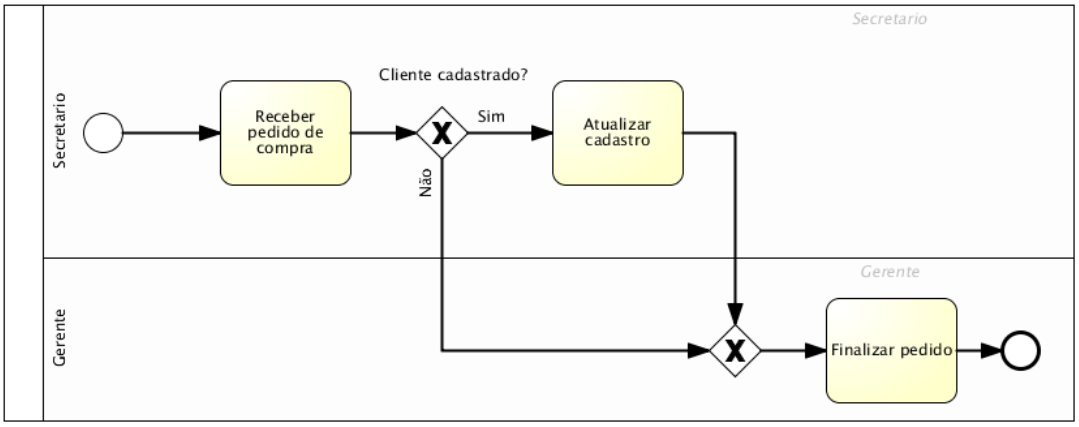
Resultado Execução 3 - Texto em linguagem natural gerado para o Modelo de Processo 2.



Modelo de Processo 3 – Modelo de processo intermediário em inglês.

The process begins when the Secretary receives order. In case the customer is registered, the Secretary updates the customer data. Then, the Manager closes order. Finally, the process is finished.

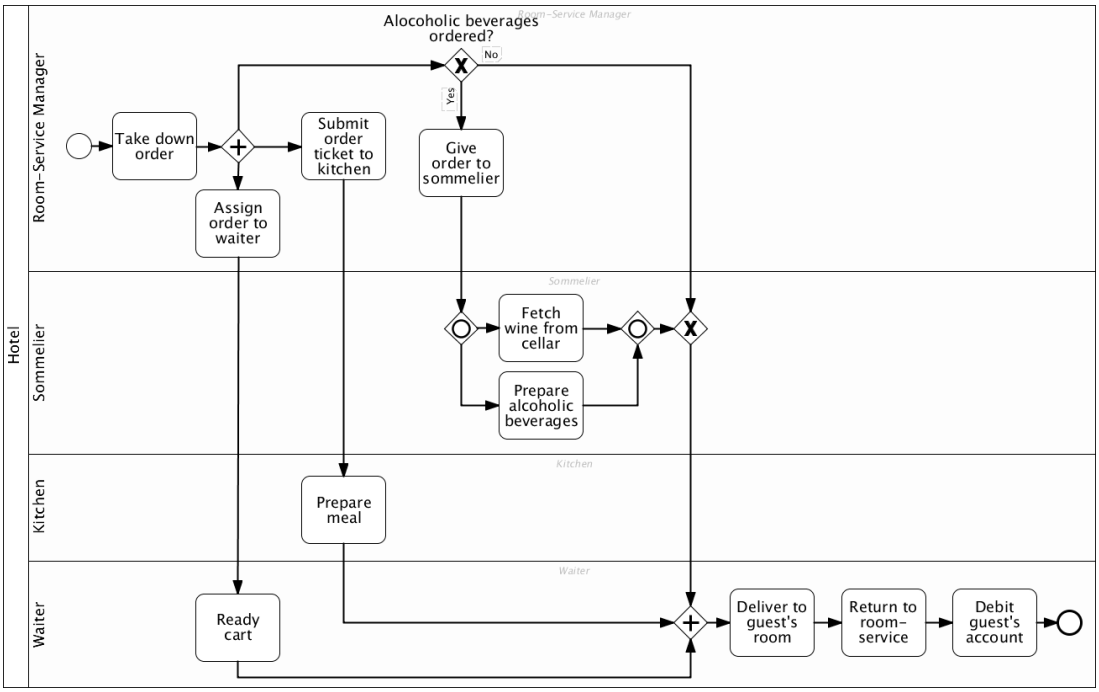
Resultado Execução 4 - Texto em linguagem natural gerado para o Modelo de Processo 3.



Modelo de Processo 4 – Modelo de processo intermediário, em português.

O processo começa quando o Secretário recebe o pedido de compra. Caso cliente é cadastrado, o Secretário atualiza o cadastro. Então, o gerente finaliza o pedido. Finalmente, o processo termina.

Resultado Execução 5 - Texto em linguagem natural gerado para o Modelo de Processo 4.



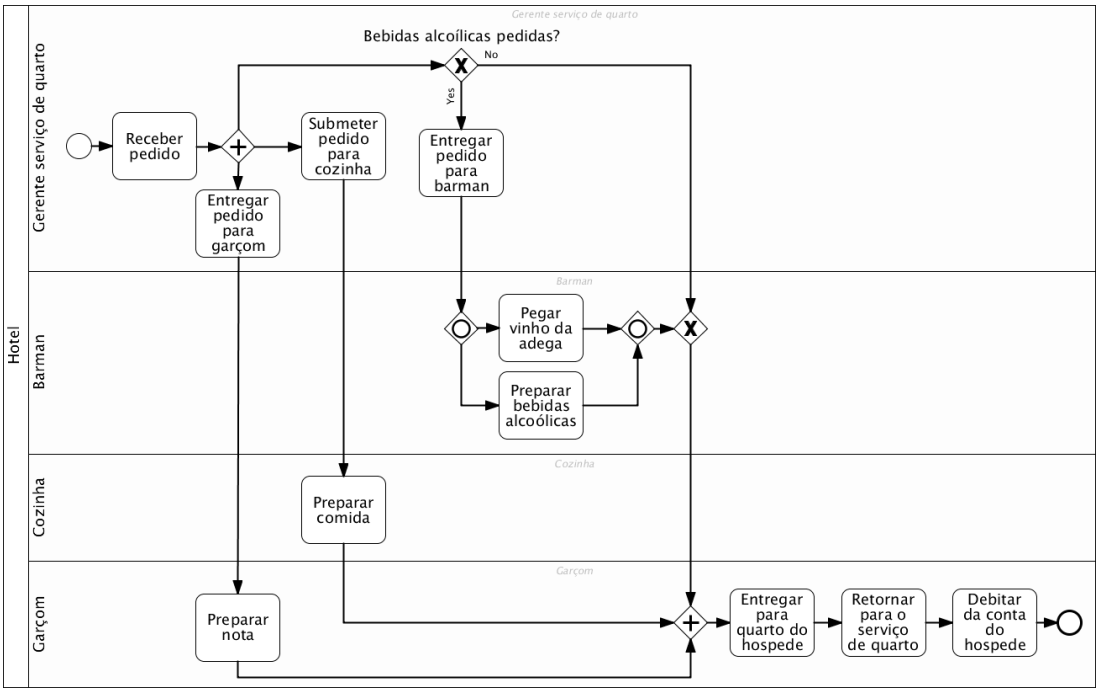
Modelo de Processo 5 – Modelo de processo avançado em inglês.

The process begins when the Room-Service Manager takes down order. Then, the process is split into the 3 parallel branches:

- In case alcoholic beverages are ordered, the Room-Service Manager gives order to the sommelier. Afterwards, the one or more of the following paths is executed:
  - The Sommelier fetches wine from the cellar.
  - The Sommelier prepares the alcoholic beverages.
- The Room-Service Manager submits the order ticket to the kitchen. Subsequently, the Kitchen prepares meal.
- The Room-Service Manager assigns order to the waiter. Then, the Waiter readies cart.

As long as all the 3 branches were executed, the Waiter delivers to the guest's room. Afterwards, the Waiter returns to room-service. Subsequently, the Waiter debits the guest's account. Finally, the process is finished.

Resultado Execução 6 - Texto em linguagem natural gerado para o Modelo de Processo 5.



Modelo de Processo 6 – Modelo de processo avançado, em português.

O processo começa quando o Gerente serviço de quarto recebe o pedido. Então, o processo é dividido em 3 ramificações paralelas.

- Caso bebidas alcoólicas são pedidas, o Gerente serviço de quarto entrega o pedido para o barman. Em seguida, os seguintes caminhos são executados:
  - O Barman pega o vinho da adega.
  - O Barman prepara as bebidas alcoólicas.
- O Gerente serviço de quarto entrega o pedido para o garçom. Subsequentemente, o garçom prepara a nota.
- O gerente serviço de quarto submete o pedido para cozinha. Então, a cozinha prepara a comida.

O Garçom entrega para o quarto do hospede. Em seguida, o garçom retorna para serviço de quarto. Subsequentemente, o garçom debita da conta do hospede. Finalmente, o processo é terminado.

Resultado Execução 7 - Texto em linguagem natural gerado para o Modelo de Processo 6.