Geração Automática de Especificação BPMN em C++ a partir de arquivo modelo oficial

Gustavo Góes de Menezes
Análise e Desenvolvimento de Sistemas
Instituto Federal da Bahia
Salvador, Brasil
Email: gustavomenezes@ifba.edu.br

Renato Novais
Instituto Federal da Bahia
Salvador, Brasil
Email: renatonovais@ifba.edu.br

***Resumo*—Quando o objetivo é sistematizar e facilitar processos organizacionais individuais complexos dentro e fora de empresas, a abordagem Business Process Management (BPM) é uma opção elegante, pois tem como intuito trazer a tona informações pertinentes de como os processos são executados para que melhorias possam ser realizadas e para que os processos possam ser gerenciados possibilitando uma melhor tomada de decisões e visão do negócio como um todo. Para podermos descrever processos nessa abordagem, existe a notação que descreve de forma gráfica esse gerenciamento, a notação Business Process Management and Notation (BPMN). Trata-se de um padrão que ilustra o processo de gerenciamente de processos de uma maneira simples e clara, pensada tanto para gerentes quanto para profissionais técnicos, basicamente representando de forma eficiente um sistema de processos com símbolos gráficos de fácil entendimento humano. Dentro deste contexto, os benefícios que desse tipo de abordagem no mundo do desenvolvimento de softwares é evidente. Cada vez mais temos o desenvolvimento de sistemas cada vez mais críticos e complexos, ambiente ideal para a implementação de Business Process Management. Porém, como se trata de uma notação padronizada, criar a ponte entre um modelo gráico de negócios para a implementação de código fonte é algo que requer tempo e análise. E como sabemos, tempo é um recurso limitado e valioso. A proposta deste documento é apresentar uma ferramente cujo objetivo é criar esta ponte. Oferecendo assim com o C++ moderno, uma representação via software do padrão Business Process Management and Notation.**

***Keywords*—C++, BPM, BPMN, Processos, Implementação, Biblioteca.**

I. Introdução

Gestão de Processos de Negócios (BPM) é a disciplina que combina o conhecimento da tecnologia da informação e o conhecimento das ciências de gestão e aplica isso aos processos de negócios operacionais [1][2].

Ela recebeu atenção considerável nos últimos anos devido ao seu potencial para aumentar significativamente produtividade e economia de custos. Além disso, hoje existe uma abundância de sistemas BPM [3].

Segundo John Jeston, BPM pode ser útil até mesmo em processos de negócios que não fazem usem de tecnologia por si só. As ferramentas de modelagem de processos são úteis para alcançar melhorias de processo em circunstâncias não tecnológicas, caso estas ferramentas sejam usadas da forma correta. Na verdade, é difícil concluir projetos complexos de melhoria de processos de maneira eficaz em termos de tempo sem o uso dessas ferramentas [4].

No contexto de desenvolvimento de software, existem muitas propostas para a integração de BPM, e é aí que nasce o BPMN (Business Process Model and Notation). BPMN define um Diagrama de Processo de Negócios (BPD), que é baseado em uma técnica de fluxograma adaptados para criar modelos gráficos de operações de processos de negócios (BPM). Um modelo de processo de negócios, então, é uma rede de objetos gráficos, que são atividades (ou seja, trabalho) e os controles de fluxo que define sua ordem de atuação. O principal objetivo do BPMN é fornecer uma notação que seja facilmente compreensível por todos usuários de negócios, desde os analistas de negócios que criam os rascunhos iniciais dos processos, até os desenvolvedores técnicos responsáveis pela implementação da tecnologia que irá realizar esses processos e, finalmente, aos empresários que irão gerenciar e monitorar esses processos [5].

A *Business Process Management Initiative* (BPMI) desenvolveu um padrão conhecido como *Business Process Modeling Notation* (BPMN). O padrão e sua especificação BPMN 1.0 foram lançados ao público em maio de 2004 [6]. O objetivo principal do esforço em produzir o BPMN era fornecer uma notação que fosse prontamente compreensível por todos usuários de negócios, desde os analistas de negócios que criam os rascunhos iniciais dos processos, até os desenvolvedores técnicos responsáveis pela implementação da tecnologia que realizará esses processos e, finalmente, aos empresários que irão gerenciar e monitorar os mesmos. O BPMN define um *Business Process Diagram* (BPD), que é baseado em uma técnica de fluxograma adaptado para a criação de modelos gráficos de operações de processos de negócios. Um Modelo de Processo de Negócios é então, uma rede de objetos gráficos, que são atividades (ou seja, trabalho) e os controles de fluxo que definir sua ordem de atuação [6].

Hoje é possível desenhar um diagrama de processos e salvá-los utilizando essa notação. Geralmente os modelos são salvos no formato XML para posterior leitura e modificação. Todavia, no caso de surgir uma necessidade de se desenvolver um software ou sistema inteiro utilizando-se como base o modelo que foi desenhado e posteriormente salvo, os desenvolvedores precisam dedicar um tempo considerável criando inúmeras estruturas e conectores a nível de software que irá, muito provavelmente, ter uma alta taxa de acoplamento à especialização do software. Além disso, nada garante afinidade com os padrões BPMN. Uma possível solução para tal problema

seria uma biblioteca que forneça todos os recursos e abstrações para que seja possível "reproduzir" o fluxograma a nível de código. Dessa forma essa biblioteca poderá ser reutilizada, pois estará livre de qualquer contextualização que não seja a descrição do modelo BPMN, livrando o código-fonte de qualquer nível de acoplamento. O resultado pode se tornar simples e eficaz.

Uma biblioteca orientada a objetos que siga fielmente todas as especificações do padrão BPMN abriria uma janela de possibilidades para os desenvolvedores de softwares que lidam com fluxo de negócios, desde os mais simples aos projetos mais complexos. Além da vantagem da reutilização da própria biblioteca, os softwares que a utilizem poderão também ser "desenhados" graficamente por editores BPMN, pois esse tipo de solução abre a possibilidade de criação de geradores gráficos de código executável. Ou seja, softwares que geram outros softwares, bastando apenas desenhar todo o esquema BPMN. No final, ao invés de gerar um XML que descreve a notação visual do BPMN, teríamos um código funcional em alguma linguagem e que descreve perfeitamente, em nível de tempo de execução, a notação BPMN desenhada.

Os editores visuais BPMN geram um XML que contem todas as informações correspondente ao que o usuário desenhou. Desta forma, existe a possibilidade desse XML ser lido para posterior "materialização" do software em código executável.

Porém para isso tudo ser possível, é necessário que exista uma biblioteca orientada a objetos que descreva o que é a notação, toda a descrição dos CMOF (Complete Meta Object Facility), arquivos de meta-objeto do modelo BPMN, da especificação é baseada em orientação a objetos e enumerações. Infelizmente, pelo menos dentro do conceito C++, não existe ainda uma solução que envolva o desenvolvimento de uma biblioteca BPMN.

Neste artigo, é apresentada uma solução C++ que, baseada no arquivo CMOF da especificação da notação, gera uma biblioteca C++ com todas as classes e enumerações que representam os componentes visuais BPMN. O uso do projeto é simples, sendo necessário apenas alimentar o mesmo com o arquivo CMOF para que o mesmo "traduza" suas especificações para código C++. Dessa forma o *output* é uma série de arquivos .cpp e .h correspondentes à implementações de suas classes e enums para posterior utilização por outros softwares.

O BPMN fornece às organizações a capacidade de entender seus procedimentos internos de negócios em uma notação gráfica, bem como a capacidade de comunicar esses procedimentos de maneira padrão. As primitivas conceituais básicas em modelos BPMN são eventos, tarefas e fluxos. Atualmente, com o BPMN é necessário para documentar os processos de negócios com o objetivo de reduzir tempo e custos através da melhoria dos processos. Os modeladores BPMN existentes permitem projetar diagramas facilmente, oferecendo opções para melhorar os processos de negócios. Entre outros modeladores, podemos destacar: Bizagi, Adonis NP e Auraportal, porém existem muitas outras que são igualmente boas e oferecem o serviço online.

Em todos os modeladores existentes, o design de interfaces e classes é colocado em segundo plano, deixando essa tarefa para o projetista, que deve se esforçar para implementar interfaces de acordo com o modelo BPMN. Como as interfaces são atualmente derivadas de modelos BPMN de forma artesanal, sem nenhum tipo de procedimento, dependendo apenas da experiência do analista, significa que apesar do esforço feito na construção do BPMN, ele não é útil na hora de projetar as interfaces ao final[7]. E esse é o ponto que o projeto deste artigo tem o objetivo de ajudar, oferecendo recursos de acordo com a padronização BPMN para auxiliar o desenvolvimento de softwares C++ que contenham processos baseados na notação.

II. Fundamentação Teórica

A. *O que é o BPMN*

O BPM pode ser definido, de forma bastante objetiva, como um rol de práticas com foco total na modelagem de processos de uma empresa. Tais práticas são (e não limitam-se a): Mapeamento de processos, padronização de processos, melhoria de processos e otimização. Muitas vantagens são atribuídas ao BPM [8].

O nome Business Process Management, BPM, formado com base na língua inglesa, significa "gestão de processos de negócio" [8]. Seu maior objetivo é ampliar a competitividade do negócio, otimizando ao máximo a produtividade dos colaboradores. Esse modelo de práticas faz uso extensivo do fator tecnológico, tornando assim a tecnologia como algo indispensável a fim de reduzir ao máximo o risco humano, aumentar a velocidade de execução, além de fornecer um panorama rico em dados e diversos pontos de vista para a direção da organização. Isso gera uma maior lucidez para tomadas de decisão rápidas e assertivas.

O BPM pode ser visto como uma evolução do conceito de WFM (WorkFlow Management). O WFM se concentra principalmente na automação dos processos de negócios, enquanto o BPM tem um escopo mais amplo: da automação e análise de processos ao gerenciamento de operações e organização do trabalho. Por outro lado, o BPM visa melhorar os processos de negócio, possivelmente sem o uso de novas tecnologias. Por exemplo, ao modelar um processo de negócios e analisá-lo usando simulação, a administração pode ter ideias sobre como reduzir custos e, ao mesmo tempo, melhorar os níveis de serviço. Por outro lado, BPM é frequentemente associado a software para gerenciar, controlar e apoiar processos operacionais. Isso deu origem a um novo tipo de tecnologia, chamados de sistemas BPM, que podem se conectar com uma variedade de sistemas (legados), bem como tecnologias emergentes (por exemplo, redes em nuvem, dispositivos móveis), e substituíram efetivamente seus predecessores, os sistemas WFM [8]. Apesar disso, o nicho ocupado pelo BPM é relativamente novo e atualmente há pouca literatura a respeito.

O fator tecnológico é extremamente importe para as metodologias BPM, porém os Recursos Humanos e os analistas de negócio também têm uma participação igualmente importante, e por isso, a legibilidade se torna necessária para a modelação do fluxo de processos operacionais. Tendo isso em mente a notação BPMN surgiu.

O Business Process Modeling Notation (BPMN) é o novo padrão para modelar fluxos de processos de negócios e serviços da web [9]. Criado pelo Processo de Negócios Management

Initiative (BPMI), o objetivo primário do BPMN é fornecer uma notação que seja facilmente compreensível por todos os usuários de negócios [9]. Isso inclui os analistas de negócios que criam os rascunhos iniciais dos processos para os desenvolvedores técnicos responsáveis por implementação da tecnologia que executará esses processos [9].

Um segundo objetivo igualmente importante é garantir que as linguagens XML projetadas para a execução de processos de negócios, como BPEL4WS (Business Process Execution Language for Web Services) e BPML (Business Process Modeling Language), podem ser expresso visualmente com uma notação comum. BPMN especifica um único diagrama de processo de negócios, chamado de Processo de Negócios Diagrama (BPD). Este diagrama foi projetado para fazer bem duas coisas. A primeira delas, existe uma facilidade em usar e compreender. Você pode usá-lo para modelar processos de negócios de forma rápida e fácil. Além disso, é muito compreensível por usuários não técnicos (geralmente de gerenciamento). Em segundo lugar, oferece a expressividade para modelar processos de negócios muito complexos e pode ser mapeado naturalmente para linguagens de execução de negócios.

Para modelar um fluxo de processo de negócios, você modela os eventos que ocorrem para iniciar um processo, os processos que são executados e os resultados finais do fluxo do processo. As decisões de negócios e a ramificação de fluxos são modeladas usando portais. Um portal é semelhante a um símbolo de decisão em um fluxograma. Além disso, um processo no fluxo pode conter subprocessos, que podem ser mostrado graficamente por outro Diagrama de Processo de Negócios conectado por meio de um hiperlink para um símbolo de processo. Se um processo não é decomposto por subprocessos, ele é considerado uma tarefa - o processo de nível mais baixo. Uma marca '+' no símbolo do processo denota que o processo é decomposto; se não tiver uma marca '+', é uma tarefa [9].

Os diagramas BMPM têm quatro categorias de elementos gráficos: objetos de fluxo, objetos de conexão, raias e artefatos [10]. Os nomes para esses objetos às vezes são semelhantes às designações usadas em um sentido orientado a objeto, mais precisamente a notação UML. Embora não sejam idênticos, há uma sobreposição significativa nos termos. Quando essas colisões de termos ocorrem, eles são anotados em um texto a parte.

A primeira colisão ocorre na categoria de objetos BPMN chamados objetos de fluxo. Esta categoria inclui atividades (uma unidade atômica de ação em BMPN) e o gateway. O gateway tem muito em comum com uma ramificação em diagramas de atividades, na medida em que denota um ponto onde uma decisão é tomada. Os elementos dentro do fluxo da categoria de objeto são os mais facilmente reconhecíveis, pois são muito semelhantes em aparência e uso ao tradicional elementos do fluxograma. Essa semelhança é intencional e forma o BPMN Nível 1, ou o que é referido como BPMN descritivo [11]. Neste nível básico, os objetos de fluxo fundamentais são usados para ajudar mostrar como os envolvidos no processo colaboram. O Nível 2 é denominado BPMN Analítico e tem como objetivo fornecer uma descrição mais precisa de como o processo funciona. Este nível mais alto de expressividade requer mais símbolos para suportar e, portanto, é menos amigável para o novato [11]. O Nível 3 BPMN também é chamado de Executable BPMN e seu foco está em desenvolvedores [11]. Isto é aqui que o BPMN faz um movimento na representação de elementos programáticos como objetos dentro do modelo. A intenção do Nível 3 é integrar os elementos do Nível 2 com a Extensible Markup Language (XML) para produzir um processo que pode ser executado por um interpretador ou compilador adequado.

Esta é a vanguarda do padrão BPMN. O objetivo dos criadores de BPMN era preencher a lacuna entre linguagens de modelagem e linguagens de execução [10].

O próximo grupo de elementos são chamados de objetos de conexão e consistem nas linhas que conectam os elementos do diagrama. Objetos de conexão consistem em fluxos de sequência (conecta elementos dentro da mesma raia), fluxos de mensagens (conecta elementos de diferentes conjuntos) e associações (ajuda a esclarecer entradas e saídas)[10]. O terceiro grupo de elementos é chamado de raias, e esses componentes são usados para segmentar o diagrama em áreas que contêm uma sequência de atividades relacionadas conectadas pelos fluxos de sequência [10]. As raias são um símbolo de comunicação poderoso porque compartimentam o diagrama em tais maneiras que se torna aparente qual entidade está tomando a ação (atividades e portais que compõem as operações e decisões) e como essas ações se relacionam com as de outros elementos. O quarto e último grupo de objetos em um diagrama BPMN é chamado de artefatos. Artefatos são elementos que ajudam agrupar e anotar o modelo, para que o leitor possa entender melhor o contexto das atividades e ações que estão representados nele. Há ocasiões em que outros elementos importantes de um moderno o fluxo do processo toma a dianteira em um modelo de processo.

Bancos de dados e Big Data estão se tornando rapidamente as ferramentas analíticas usadas por organizações orientadas a dados. Para ser abrangente em sua aplicabilidade ao moderno ambiente de negócios, o BPMN precisava fornecer um símbolo que pudesse ser usado para modelar essas fontes de dados. O objeto de dados dentro da classificação do objeto de artefato é o elemento perfeito para descrever o fato de que os dados podem ser um parte crítica do processo de negócios. Na prática, os objetos de dados são incluídos no fluxo do processo com atividades e gateways, conectando-os a fluxos de associação. O fluxo de associação dentro do BPMN não é o mesmo que em UML. A associação em BPMN é uma linha pontilhada que é usada exclusivamente para conectar dados, texto e outros elementos da classificação de artefato com elementos da categoria de objeto de fluxo. O exemplo abaixo esclarece esse tipo de associação.

B. Ferramentas de geração de código

O tipo de ferramenta mais fácil de se encontrar e implementar dentro do escopo de BPMN são as que permitem a modelagem de modelo gráficos BPMN e que oferecem suporte a gravação e persistência dos dados. É possível modelar seu fluxograma e salvar para posterior continuação ou para importar em outros tipos de ferramenta. Isso só é possível por causa da padronização do BPMN, que dita como um projeto BPMN deve ser representado por um modelo XML. O exemplo da figura 2 se trata do processo de se passar café, modelado em uma ferramenta online de construção gráfica de processos

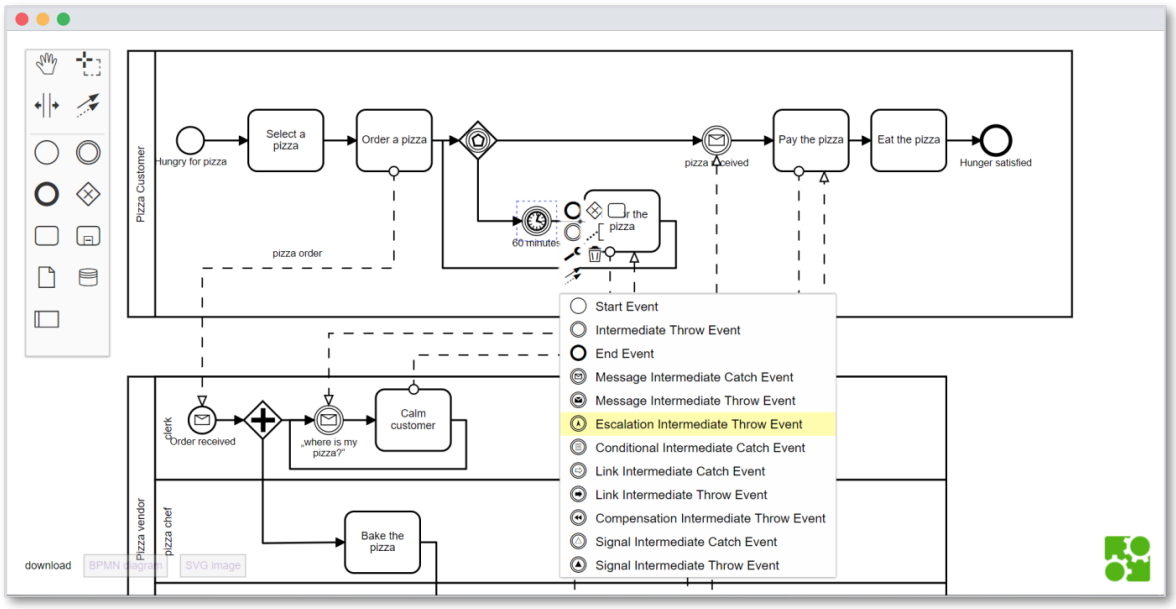

Figura 1. Exemplo típico de diagrama BPMN

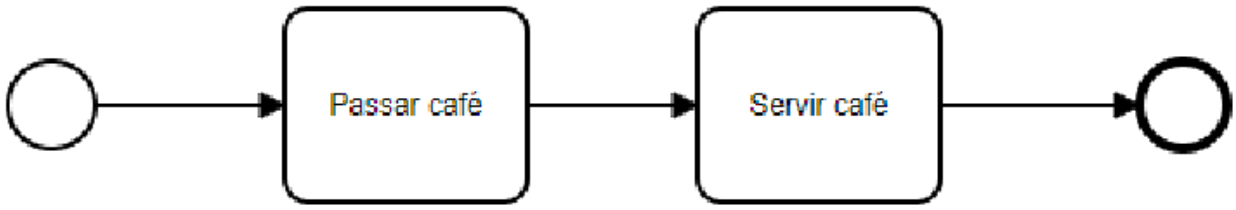

Figura 2. Exemplo simples de diagrama BPMN.

```
<bpmn:process id="Process_00w3t6d" isExecutable="false">
  <bpmn:startEvent id="StartEvent_1quctgs">
    <bpmn:outgoing>Flow_0b0vopo</bpmn:outgoing>
  </bpmn:startEvent>
  <bpmn:task id="Activity_0yynw3w" name="Passar café">
    <bpmn:incoming>Flow_0b0vopo</bpmn:incoming>
    <bpmn:outgoing>Flow_16kdtgr</bpmn:outgoing>
  </bpmn:task>
  <bpmn:sequenceFlow id="Flow_0b0vopo" sourceRef="StartEvent_1quctgs" targetRef="Activity_0yynw3w" />
  <bpmn:task id="Activity_1xcsny9" name="Servir café">
    <bpmn:incoming>Flow_16kdtgr</bpmn:incoming>
    <bpmn:outgoing>Flow_0byomdw</bpmn:outgoing>
  </bpmn:task>
  <bpmn:sequenceFlow id="Flow_16kdtgr" sourceRef="Activity_0yynw3w" targetRef="Activity_1xcsny9" />
  <bpmn:endEvent id="Event_0kus274">
    <bpmn:incoming>Flow_0byomdw</bpmn:incoming>
  </bpmn:endEvent>
  <bpmn:sequenceFlow id="Flow_0byomdw" sourceRef="Activity_1xcsny9" targetRef="Event_0kus274" />
</bpmn:process>
```

Figura 3. Exemplo da representação XML do modelo simples da figura 2.

BPMN. É um exemplo extremamente simples para o propósito de exemplificação.

Após salvar o XML que representa o desenho, e abstraindo as *tags* que descrevem as coordenadas gráficas de cada componente, teremos um XML como exemplificado na figura 3.

Como podemos ver, temos as tag que representam o estado inicial e final, tags que representam a transição (os conectores) e por fim, as tags que representam os processos em si. Devido à padronização, esse XML será portável a todas as ferramentas de importação/exportação que estejam de acordo com o BPMN e seus padrões.

O projeto apresentado nesse artigo tem uma íntima relação com os XML gerados, pois, uma vez que um desenvolvedor termine de modelar seu processo BPMN utilizando uma dessas ferramentas, será possível utilizar o XML gerador e desenvolver uma versão executável e testável desse modelo. Mais detalhes na Seção III-B.

C. Trabalhos relacionados

A filosofia de geração de códigos com um objetivo pré-estabelecido não é um conceito novo. De fato, alguns projetos famosos fazem uso desse tipo de solução [12][13]. Podemos citar, como exemplos famoso, o Django e o Rails, que geram uma estrutura de código baseada na configuração de entrada do gerador, geralmente por linha de comando ou arquivos de configuração [12][13].

Dentro do escopo de BPMN, podemos citar alguns projetos que utilizaram de forma criativa esse conceito.

1) Uma ferramenta de código aberto para transformação de BPMN para serviços Web (B2W): O planejamento de recursos empresariais (Enterprise Resource Planning ou ERP) é um sistema de gerenciamento de processos de negócios no qual aplicativos integrados são usados para gerenciar processos de negócios em um ambiente de dados compartilhados [14]. Os sistemas ERP geralmente lidam com os dois tipos de processos de negócios, ou seja, troca e conversão. No processo de troca, o recurso econômico, como o produto, é trocado por outro recurso econômico, como o processo de vendas. Em um processo de conversão, uma empresa consome recursos para produzir novos recursos, como o processo de distribuição. Geralmente, a comunicação entre os aplicativos ERP é baseada nos processos de conversão e troca e é realizada por meio de serviços Web. Nesse contexto, a implementação de serviços Web em sistemas ERP é uma tarefa complexa [14].

Para gerenciar isso, o modelo e notação de processos de negócios (BPMN) são frequentemente utilizados para simplificar o desenvolvimento de aplicativos ERP. No entanto, abordagens BPMN de última geração geralmente lidam com a modelagem de processos de troca sem considerar o processo de conversão. Além disso, a solução de transformação de modelos para gerar automaticamente serviços Web a partir dos modelos BPMN são difíceis de encontrar na literatura [14].

Portanto, a proposta desse trabalho correlato é a criação de um novo *framework*, o ERP, que suporta a modelagem de trocas e processos de conversão por meio de BPMN.[14] Particularmente, uma abordagem de modelagem é introduzida para representar os processos de ERP através de conceitos BPMN. Posteriormente, as regras são desenvolvidas para converter modelos BPMN de origem em modelos de linguagem de modelagem (SoaML) de arquitetura orientada a serviços de destino. Finalmente, as regras de transformação são desenvolvidas para gerar serviços Web Java executáveis totalmente funcionais a partir de modelos SoaML. Como parte da pesquisa, uma ferramenta completa de BPMN para transformação de serviços da Web (B2W) de código aberto é desenvolvida para gerar automaticamente os serviços da Web a partir dos modelos BPMN de alto nível. A estrutura proposta é validada por meio de estudos de caso múltiplos. Os resultados experimentais comprovam que o *framework* proposto gera serviços Web com precisão a partir dos modelos BPMN, o que eventualmente auxilia no desenvolvimento dos sistemas ERP com simplicidade [14].

2) Método para gerar protótipos de GUI a partir de BPMN: O Business Process Model and Notation (BPMN) fornece às organizações um padrão que facilita a compactação adicional do processo de negócios. O BPMN foca nos processos funcionais, deixando de lado o desenvolvimento de interfaces. Assim, o design da interface geralmente depende da experiência subjetiva do analista. Esse projeto tem como objetivo propor um novo método para gerar interfaces de usuário a partir de modelos BPMN e Diagramas de Classes [7]. O método proposto baseia-se na identificação de diferentes regras e faz uso de estereótipos para estender a notação BPMN. As regras foram extraídas de sete projetos existentes no repositório Bizagi (uma empresa privada). Especificamente, a proposta baseia-se na extração de regras para geração de interfaces de usuário com base em três padrões amplamente utilizados: padrão de sequência, padrão de decisão exclusivo e padrão de sincronização.

Como resultado da proposta, onze novos estereótipos foram adicionados à notação BPMN. Esses estereótipos permitem gerar interfaces baseadas em modelos de processos de negócios. Para uma melhor compreensão, os estereótipos propostos foram aplicados a um exemplo ilustrativo. Os resultados mostram que este trabalho é um "passo a frente" para a geração automática de código a partir de modelos [7]

D. Gerando código BPEL executável a partir de modelos BPMN

Abordagens de desenvolvimento de software orientado por modelo (MDSD) enfatizam o uso de modelos em todas as fases de desenvolvimento do sistema [15].

Este projeto apresenta duas transformações de modelo desenvolvidas em Visual Modeling e Sistema de Transformação (VMTS) que facilitam a transformação entre duas linguagens para modelagem de processos de negócios: BPMN e BPEL. O artigo do projeto também discute de forma abrangente como as transformações de modelo observaas alcançam a avaliação relacionada ao estudo de caso. Sendo os critérios utilizados: completude, exatidão, legibilidade e reversibilidade [16].

III. PROJETO DA ARQUITETURA

O fluxo de execução do projeto é exemplificado no seguinte fluxograma da Figura 6. Os passos são explicados a seguir.

1) Inicialmente, o usuário, via linha de comando, passa o parâmetro necessário para a geração de código baseado no BPMN, no caso o arquivo CMOF.

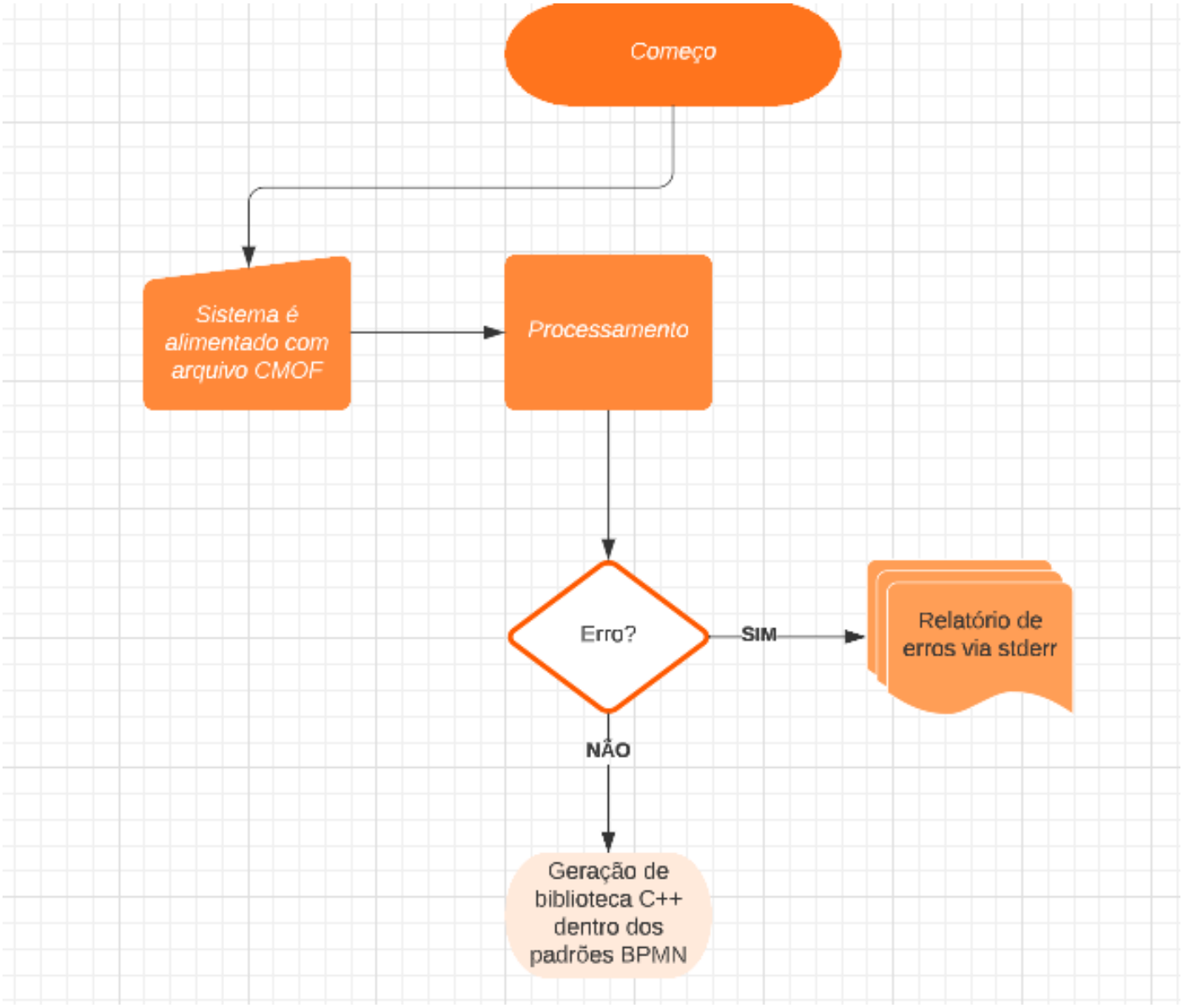

Figura 4. Visão geral do sistema

2) O arquivo CMOF é um XML que contém descrições formais de todas as relações e classes do padrão BPMN. Esse arquivo é, portanto, lido e subsequentemente transcrito para o formato JSON.
3) Nessa etapa, o gerador assume o trabalho de ler o JSON e com a biblioteca Inja, gerar código C++, essa gera código se baseando em arquivos de metacódigo C++ que servem como modelos, ou seja, exemplificam o que é uma classe, um método, etc.
4) Nesta etapa, todas as classes BPMN foram completamente relacionadas construídas em C++ e já estão prontas para uso, o conjunto delas forma a biblioteca BPMN C++.
5) O usuário final, o desenvolvedor, utiliza essa biblioteca para usufruir das vantagens do modelo BPMN em tempo de execução em seu projeto.

A solução proposta foi escrita utilizando novas características do C++ moderno, incluindo os novos smart pointers std::shared_ptr e std::unique_ptr, portados da biblioteca boost para a biblioteca padrão do C++ STL (Standard Template Library), funções lambda e representação de dados Json-like nativamente. Vale também salientar que este projeto faz uso extensivo de uma template engine chamada Inja, uma engine inspirada no Jinga para Python, e também faz uso da biblioteca de processamento de XML pugxml, para leitura e processamento do arquivo cmof. Podemos dividir a arquitetura em duas etapas:

1. A etapa de geração, responsável por gerar o código C++ da biblioteca BPMN baseando-se em arquivos modelo. Essa etapa tem a tarefa de ler o XML que contém os padrões e regras BPMN e montar internamente o bulk de dados necessários para alimentar o gerador de código da biblioteca C++ para a sintetização da mesma;

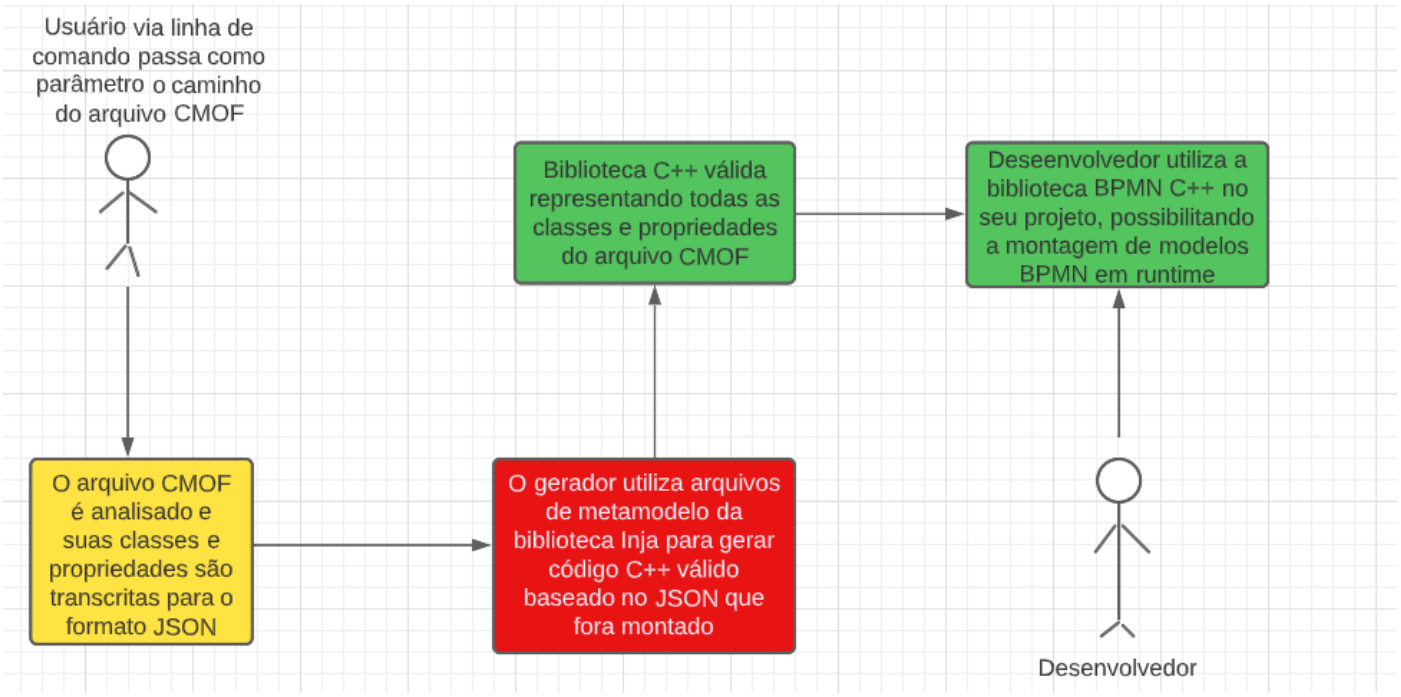

Figura 5. Fluxograma da execução do gerador. As caixas em amarelo representam a parte de análise dos metadados. As caixas em vermelho representam a etapa de geração de código. As caixas em verde representam a parte do runtime, ou seja, a biblioteca gerada que será usada pelo usuário final.

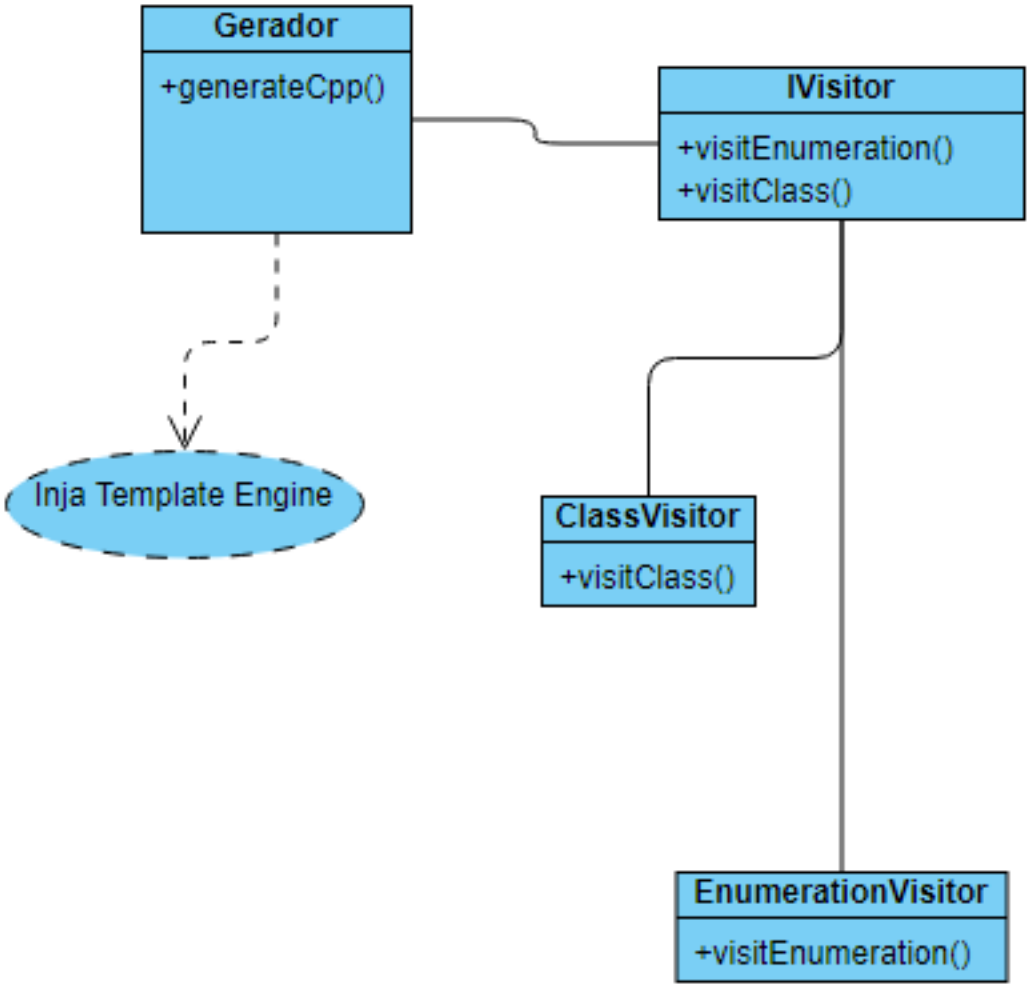

Figura 6. Estrutura arquitetural básica do gerador, evidenciando a dependência do gerador em relação à engine Inja

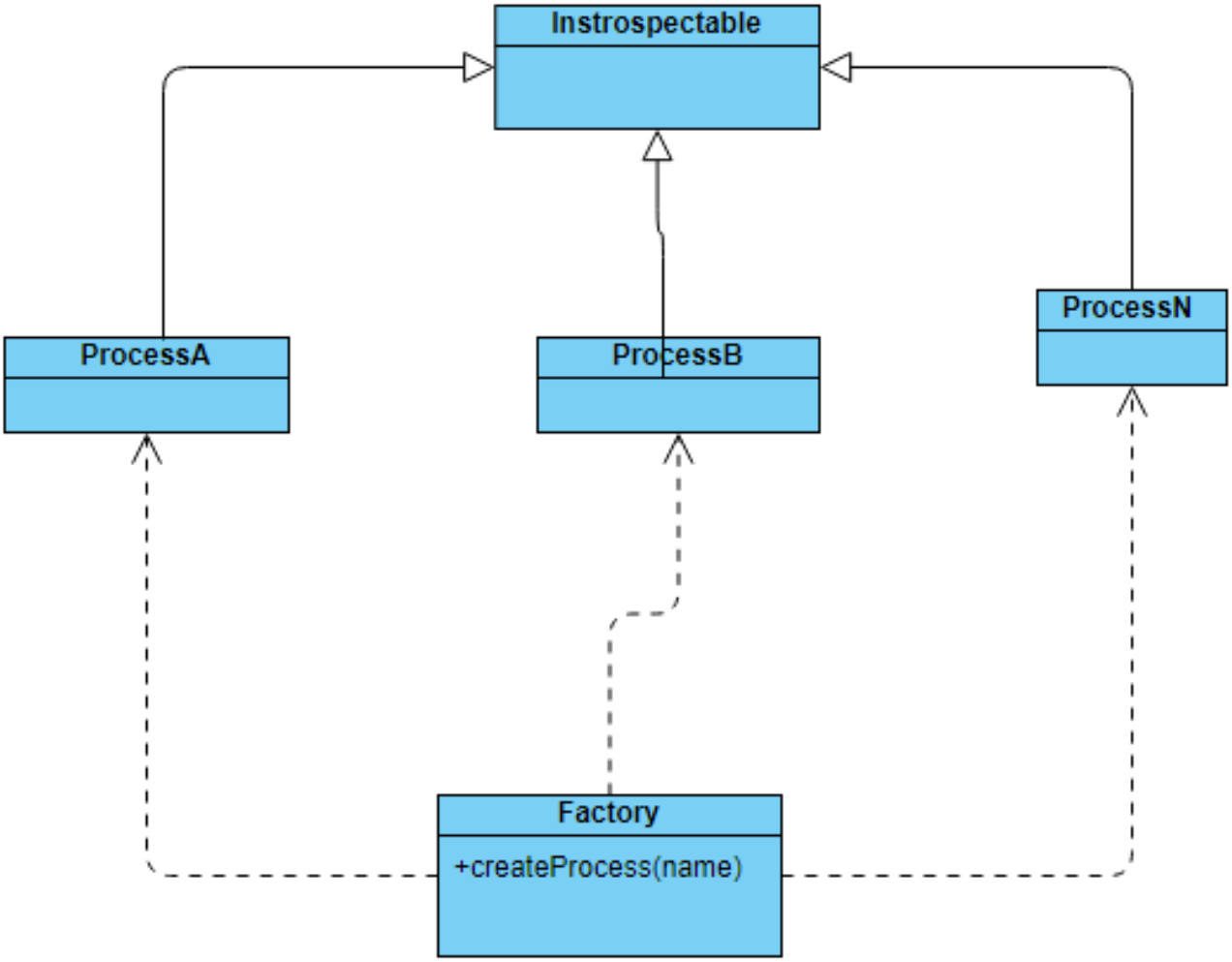

Figura 7. Estrutura arquitetural do mecanismo de introspecção com sua classe Factory.

2. A etapa de runtime é a etapa de execução da biblioteca em si. Após a geração do código-fonte C++, o mesmo pode e deve ser testado para a verificação da sua execução e funcionalidade (runtime). Essa etapa é fundamental para a leitura de arquivos .bpmp, onde os elementos serão dinamicamente instanciados.

A. *O gerador*

O arquivo CMOF descreve muitas entidades, dentre elas, as mais comuns são as enumerações, classes e suas propriedades (atributos de classe). No gerador, o padrão de projeto visitor foi utilizado para o tratamento modular de cada entidade.

Dessa forma, temos um visitor especializado sendo alimentado por dados do XML e cada visitor é capaz de montar um json que descreve cada classe e enumeração proposta pelo CMOF XML.

No final todos os jsons são fundidos em um só json global para que seja "visto"pelo ambiente de modelos da biblioteca Inja. Em paralelo a isso, arquivos de modelo .tmpl que já estão disponibilizados e fazem parte do projeto, descrevem como cada dado do json montado pelo gerador deve ser formatado para a sintetização de um código C++ válido.

A figura 5 ilustra a dependência total que o gerador tem pela biblioteca Inja, pois a mesma é utilizada tanto para iterar os objetos Json quanto para a montagem propriamente dita de código C++.

Para que toda a geração de código seja possível, a Inja nos fornece uma linguagem de programação embutida com instruções básicas tais como laços e desvios condicionais. Dessa forma, com ambos o json gerado e os arquivos de template (.tmpl) em mãos, códigos C++ válidos (.h e .cpp) correspondentes à biblioteca BPMN são gerados.

B. *O runtime*

O projeto em si oferece um recurso de runtime na biblioteca gerada.

Para oferecer a possibilidade de leitura de arquivos .bpmn e a construção dinâmica do modelo para algumas aplicações, um desafio surge. C++ não é uma linguagem naturalmente dinâmica, portanto realizar a leitura de modelos BPMN e instanciar

cada classe de processos dinamicamente não é algo fácil. Uma das solução para simular a introspecção foi justamente criar 2 arquivos de modelo .tmpl com código estático. Eles fazem uso extensivo de recursos da STL (Standard Template Library), tais como functors e std::any.

O objetivo aqui é ter uma classe base para todas as classes da biblioteca BPMN. Essa classe chama-se Introspectable, a qual fornece funções para registro de cada propriedade das classes. Dessa forma, sempre que uma classe é instanciada, ela registra suas próprias propriedades, para que essas possam ser chamadas dinamicamente, resolvendo assim o problema da introspecção. A figura 6 ilustra como todas as classes C++ realizam uma herança com a classe Introspectable, dessa maneira o registro de seus métodos e atributos pode ser realizado em temo de execução na geração de código.

Vale lembrar que a instanciação das classes também só é possível graças a uma classe Factory. Infelizmente é uma classe hard-coded, contendo todos os nomes das classes que serão instanciadas. O hard-coded aqui foi um preço ínfimo a ser pago pelo suporte de introspecção utilizando C++ puro.

IV. As funcionalidades da solução

O software que compõe esse solução foi projetado para ser de uso rápido e simples, mas também para ser leve. Portanto ele funciona via linha de comando e tem poucas opções de inicialização. Basicamente ele apenas solicita por uma entrada (o arquivo de extensão .cmof) e uma saída (diretório onde serão despachados os arquivos C++ correspondentes à biblioteca BPMN). Abaixo veremos algumas funcionalidades do projeto em si e do uso da biblioteca gerada.

A. As opções de inicialização

- Opção -h ou –help

  Essa opção exibe informações sobre o software e sobre o seu uso

- Opção -i ou –input

  Essa opção exige como argumento o endereço completo do diretório que contém o arquivo BPMP .cmof

- Opção -o ou –output

  Essa opção exige como argumento o endereço onde a biblioteca C++ será gerada, o valor padrão será sempre o diretório atual da execução do software

- Opção -v ou –verbose

  Essa opção habilitada o modo "verborrágico"de execução, ou seja, exibirá após a execução o estado completo de cada artefato C++ gerado bem como o status final da geração, podendo conter desde sucesso a warning ou falha grave. Pode ser útil para rastrear possíveis erros de parsing dos arquivos de modelo. Uma boa recomendação seria desviar a saída verbose através de um »"para um arquivo local para posterior análise, como mostrado neste exemplo: ./bpmp -i BPMN.cmof -o ./mycpp/ –verbose > logs.txt

B. Exemplo de código de uso da API gerada

Modelo BPMN são, geralmente, descritos usando XML em arquivos .bpmn. Em um leitor gráfico desses modelos, é possível ver com precisão todo o diagrama desenhado. Porém, ler esse mesmo arquivo para que todos os tipos e objetos sejam instanciados requer instanciação dinâmica. Associando então valores e propriedades a seus respectivos setters e getters.

Como exemplo de uso, vamos utilizar o simples diagrama BPMN da figura 8.

Este diagrama é representado por um arquivo .bpmn cujo código está exposto no apêndice A, código-fonte 1.

Aqui vamos salientar a importância da introspecção computacional que, diferentemente da reflexão computacional que permite a um programa modificar a estrutura e o comportamento de um objeto em tempo de execução, a introspecção computacional provê uma forma de examinar os tipos e as propriedades do objetos. Tal técnica, ao contrário da reflexão, é possível utilizando C++ puro. Apesar do C++ atualmente não ser uma linguagem dinâmica, é possível emular a reflexão através de alguns toolkits como o Qt.

Voltando ao assunto, o modelo bpmn acima pode ser instanciado estaticamente de forma simples, utilizando a biblioteca C++ gerada. Um código exemplo está exposto no apêndice A, código-fonte 2

Porém, como a leitura do arquivo modelo é dinâmica, a introspecção se torna uma necessidade, e por isso, a classe base de toda a hierarquia da API gerada deriva da classe Introspectable. Tal classe fornece a todas as classes a capacidade de registrarem o nome dos seus respectivos métodos e propriedades, permitindo assim, o fenômeno da introspecção. No apêndice A, código-fonte 3, temos um exemplo de como instanciar o modelo BPMN dinamicamente, valendo-se da capacidade de introspecção das classes.

Nesse exemplo, podemos observar também a presença de um Factory Method que faz a instanciação de facto das classes. O mesmo espera o nome da classe a ser instanciada e por isso, o software que utilizar essa API C++ deve fornecê-la após a leitura e análise de modelo .bpmn. Cada classe herda da classe Instrospectable, setters e adders e essas igualmente esperam receber dinamicamente o nome de seus respectivos métodos.

C. Diagrama de sequência

Na figura 9, podemos ver uma ilustração do fluxo esperado básico do uso dessa ferramenta geradora.

O software é invocado via linha de comando, o mesmo gera uma API e retorna para o usuário seu local. O usuário desenvolve uma software standalone com capacidade de ler modelos BPMN e utiliza a API C++ gerada para a instanciação dos objetos.

V. Detalhes de implementação da solução

A. Visão Geral

O projeto proposto envolve uma ferramenta escrita em C++ para a geração uma interface BPMN para que outros softwares baseados em BPMN possam usufruir. A interface gerada nada mais é do que uma biblioteca C++ contendo uma

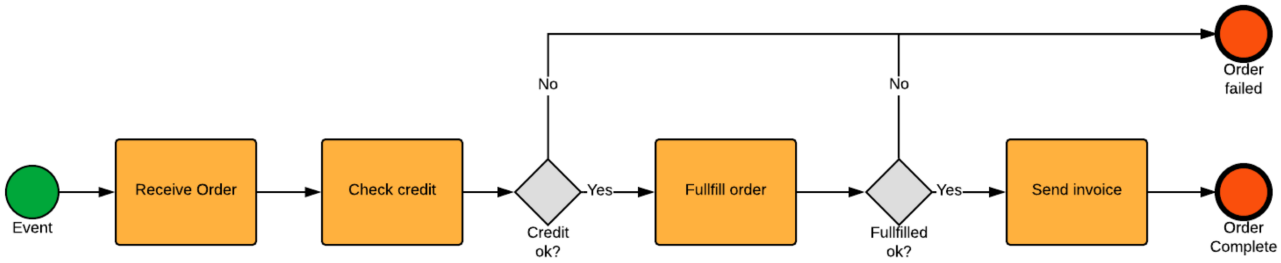

Figura 8. Exemplo de diagrama BPMN

sólida implementação semântica do padrão BPMN, padrão esse baseado na leitura do XML oficial da OMG, um arquivo chamado BPMN20.cmof. Uma vez gerada, essa biblioteca pode ser facilmente integrada a qualquer solução C++ que deseja usufruir do diagrama de modelos de processos BPM via software. A principal vantagem aqui é, digamos que os padrões BPMN sejam atualizados, o único trabalho que o desenvolver necessitará aqui será apenas gerar uma nova biblioteca e fazer as adaptações necessárias em sua aplicação.

B. Uso

Para que esse gerador funcione, primeiro deve-se realizar o download do XML da especificação BPMN, o BPMN20.cmof, esse XML contém informações sobre classes, métodos, processos, associações, operações, entre outros tipos de entidades intrinsecas ao padrão BPMN. Após o download o gerador apenas deve ser alimentado por esse cmof via linha de comando utilizando a seguinte linha de comando: "./gerador -i BPMN20.cmof -o ./"O argumento "./"após o -o refere-se ao destino dos arquivos que serão gerados, esses arquivos são código fonte C++ e todos são parte da biblioteca BPMN gerada.

C. A biblioteca gerada

Após o uso do gerador, ele lê todas as informações e instruções necessárias do XML para a geração de uma interface C++ pronta para uso para aplicações BPMN, dessa forma poupa-se um considerável tempo e trabalho do desenvolvedor de ter que ler especificações para produzir um trabalho compatível com os padrões e convenções do OMG. O objetivo aqui é realmente que essa biblioteca seja usada sem receio de fuga do design BPMN, seguindo os padrões idiomáticos do C++ com Programação Orientada a Objetos.

VI. Conclusão

Geração de códigos-fonte funcionais a partir de modelos é um grande facilitador na área de desenvolvimento. Mas como podemos ver, quando o objetivo é sistematizar processos e atividades de um sistema, o esforço para transcrever todos os relacionamentos para código não é algo trivial e achar soluções elegantes se torna um verdadeiro desafio. BPMN não tem como objetivo final oferecer uma interface sólida para o desenvolvedor final, oferecendo apenas um XML descritivo, mas pouco "*human-friendly*". A "ponte"entre a representação gráfica dos processos e a sua efetiva implementação é o objetivo da ferramenta divulgada neste artigo e, como foi apresentado na seção de funcionalidades, as figuras 5 e 9, a execução e fluxo do projeto tem como principal característica a simplicidade de uso. Sem muitos parâmetros desnecessários, necessitando apenas do arquivo da própria especificação bpmn (CMOF). O uso da biblioteca C++ gerada também tem o objetivo de simplicidade e coesão, podendo ser feitos paralelos entre o código XML que representa a modelagem das atividades com sua contra-parte em C++.

Mas, assim como todas as soluções de software, existem algumas limitações que podem ser exploradas para futuras melhorias.

A. Limitações deste Trabalho

1) No estado atual do projeto, é necessária a análise manual do XML que representa a modelagem BPMN do sistema. Ou seja, o desenvolvedor terá que manualmente utilizar a biblioteca gerada pelo projeto e instanciar/relacionar todas as atividades e processos do sistema. É inegável que o projeto poupa o desenvolvedor de ter que se preocupar em criar uma biblioteca própria para criar as relações BPMN, mas inevitavelmente o esforço para a instanciação terá que ser feito.
2) O projeto é em C++ moderno, portanto, um conhecimento básico de C++ será necessário, bem como conhecimento das classes e estruturas BPMN para a correta instanciação e relacionamento.
3) Caso a especificação BPMN sofra alguma atualização, o desenvolvedor será forçado a baixar o arquivo CMOF do endereço [**?**] para uma nova geração de biblioteca C++ atualizada. Mudanças muito radicais no CMOF poderá requerer uma atualização completa do projeto para atender as especificações afim de poder continuar com a geração da biblioteca C++ corretamente.

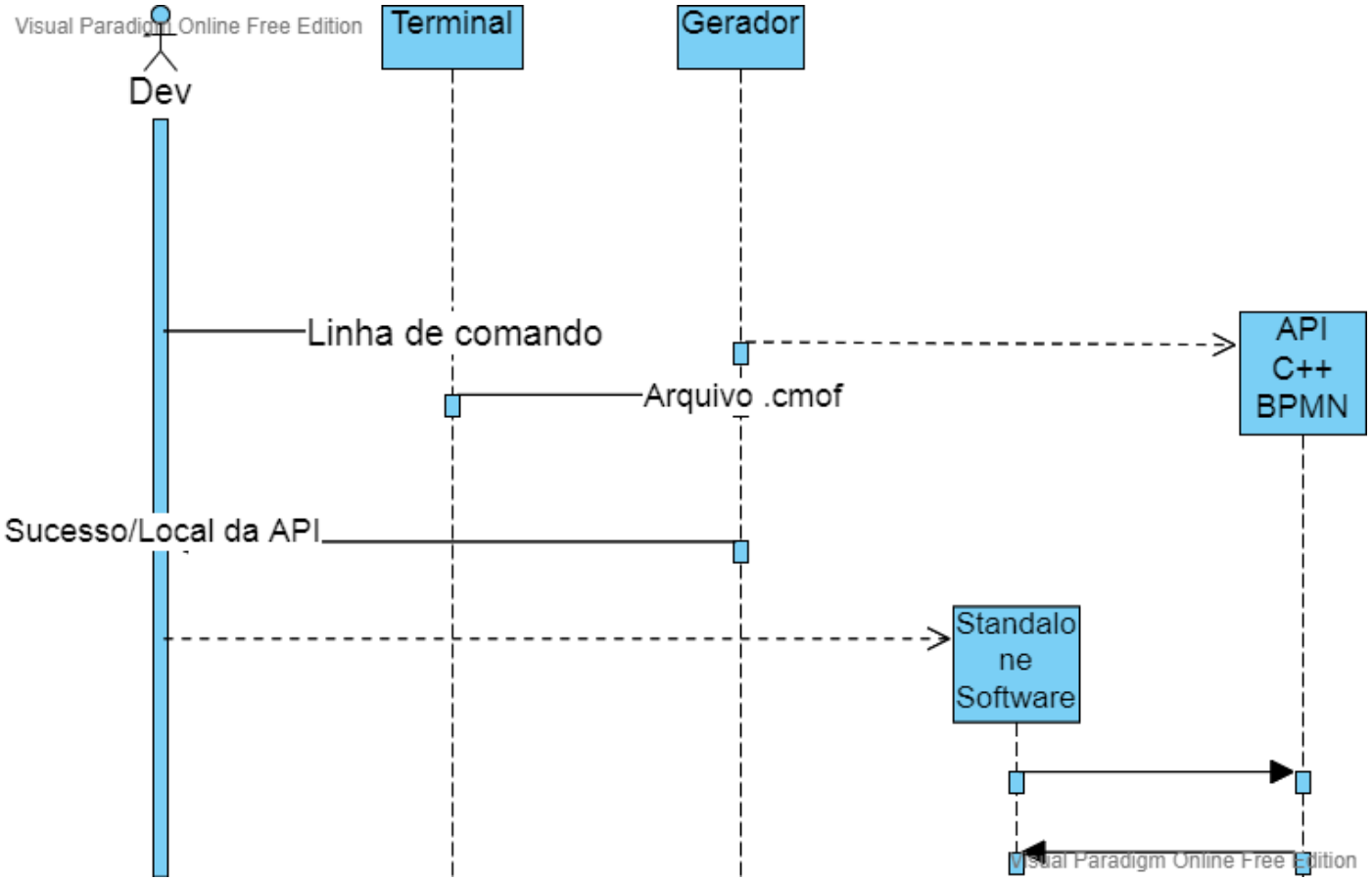

Figura 9. Diagrama de sequência do gerador

B. *Trabalhos Futuros*

Um trabalho que faria um bom par com a solução apresentada neste documento seria um solução C++ para a leitura, interpretação e análise do XML que representa o modelo BPMN de atividades e processos de um sistema. Após essa análise, gerar um código C++ válido que utiliza a biblioteca gerada. Um complementaria o outro e pouparia assim o desenvolvedor final de ter que fazer a árdua análise do XML mencionada na seção anterior. Obviamente e provavelmente o código gerado necessitaria de ajustes pessoais para adequação ao sistema alvo, mas já sério de grande proveito e pouparia provavelmente um tempo precioso.

APÊNDICE A
CÓDIGOS-FONTE

1)

```
<bpmn:process id="Process_154zshl"
isExecutable="false">
    <bpmn:startEvent
    id="StartEvent_02fqmjk"
    name="Initial State">
      <bpmn:outgoing>Flow_1q59u7i
      </bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:task id="Activity_1pqr5i0"
    name="Receive order">
      <bpmn:incoming>Flow_1q59u7i
      </bpmn:incoming>
      <bpmn:outgoing>Flow_04g6ahk
      </bpmn:outgoing>
    </bpmn:task>
    <bpmn:sequenceFlow
    id="Flow_1q59u7i"
    sourceRef="StartEvent_02fqmjk"
    targetRef="Activity_1pqr5i0"/>
    <bpmn:task
    id="Activity_1i1n9p1"
    name="Check credit">
      <bpmn:incoming>Flow_04g6ahk
      </bpmn:incoming>
      <bpmn:outgoing>Flow_0p94vgm
      </bpmn:outgoing>
    </bpmn:task>
    <bpmn:exclusiveGateway
    id="Gateway_0wv9qhz"
    name="Credit ok?">
      <bpmn:incoming>Flow_0p94vgm
      </bpmn:incoming>
      <bpmn:outgoing>Flow_0ucjs10
      </bpmn:outgoing>
      <bpmn:outgoing>Flow_1crm1f5
      </bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:task
    id="Activity_18e7xme"
    name="Fullfill order">
      <bpmn:incoming>Flow_0ucjs10
      </bpmn:incoming>
```

```
      <bpmn:outgoing>Flow_0tjjjt4
      </bpmn:outgoing>
    </bpmn:task>
    <bpmn:exclusiveGateway
    id="Gateway_1imf3p3"
    name="Fullfilled ok?">
      <bpmn:incoming>Flow_0tjjjt4
      </bpmn:incoming>
      <bpmn:outgoing>Flow_0s0bokk
      </bpmn:outgoing>
      <bpmn:outgoing>Flow_0gzhp71
      </bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:task id="Activity_1dvya6k"
    name="Send invoice">
      <bpmn:incoming>Flow_0s0bokk
      </bpmn:incoming>
      <bpmn:outgoing>Flow_0jn14rw
      </bpmn:outgoing>
    </bpmn:task>
    <bpmn:endEvent id="Event_02forxp"
    name="Order complete">
      <bpmn:incoming>Flow_0jn14rw
      </bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:endEvent id="Event_0v28q64"
    name="Order failed">
      <bpmn:incoming>Flow_1crm1f5
      </bpmn:incoming>
      <bpmn:incoming>Flow_0gzhp71
      </bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_04g6ahk
        "
    sourceRef="Activity_1pqr5i0"
    targetRef="Activity_1i1n9p1"/>
    <bpmn:sequenceFlow id="Flow_0p94vgm
        "
    sourceRef="Activity_1i1n9p1"
    targetRef="Gateway_0wv9qhz" />
    <bpmn:sequenceFlow id="Flow_0ucjs10
        "
    name="Yes" sourceRef="
        Gateway_0wv9qhz"
    targetRef="Activity_18e7xme"/>
    <bpmn:sequenceFlow id="Flow_0tjjjt4
        "
    sourceRef="Activity_18e7xme"
    targetRef="Gateway_1imf3p3" />
    <bpmn:sequenceFlow id="Flow_0s0bokk
        "
    name="Yes" sourceRef="
        Gateway_1imf3p3"
    targetRef="Activity_1dvya6k" />
    <bpmn:sequenceFlow id="Flow_0jn14rw
        "
    sourceRef="Activity_1dvya6k"
    targetRef="Event_02forxp" />
    <bpmn:sequenceFlow id="Flow_1crm1f5
        "
    name="No" sourceRef="
        Gateway_0wv9qhz"
    targetRef="Event_0v28q64" />
    <bpmn:sequenceFlow id="Flow_0gzhp71
        "
    name="No" sourceRef="
        Gateway_1imf3p3"
    targetRef="Event_0v28q64" />
  </bpmn:process>
```

.

2)

```
// Instantiate objects
auto *Process_154zshl = new Process;
auto *StartEvent_02fqmjk = new
    StartEvent;
auto *Activity_1pqr5i0 = new Task;
auto *Activity_1i1n9p1 = new Task;
auto *Activity_18e7xme = new Task;
auto *Activity_1dvya6k = new Task;
auto *Gateway_0wv9qhz = new
    ExclusiveGateway;
auto *Gateway_1imf3p3 = new
    ExclusiveGateway;
auto *Event_02forxp = new EndEvent;
auto *Event_0v28q64 = new EndEvent;
auto *Flow_1q59u7i = new SequenceFlow;
auto *Flow_0p94vgm = new SequenceFlow;
auto *Flow_0tjjjt4 = new SequenceFlow;
auto *Flow_0s0bokk = new SequenceFlow;
auto *Flow_0jn14rw = new SequenceFlow;
auto *Flow_1crm1f5 = new SequenceFlow;
auto *Flow_0gzhp71 = new SequenceFlow;

// Adjust their properties
Process_154zshl->setIsExecutable(false)
    ;
StartEvent_02fqmjk->setName("Initial
    State");
StartEvent_02fqmjk->addOutgoing(
    Flow_1q59u7i);

Activity_1pqr5i0->setName("Receive
    Order");
Activity_1pqr5i0->addIncoming(
    Flow_1q59u7i);
Activity_1pqr5i0->addOutgoing(
    Flow_04g6ahk);
Activity_1i1n9p1->setName("Check Credit
    ");
Activity_1i1n9p1->addIncoming(
    Flow_04g6ahk);
Activity_1i1n9p1->addOutgoing(
    Flow_0p94vgm);
Activity_18e7xme->setName("Fullfill
    Order");
Activity_18e7xme->addIncoming(
    Flow_0ucjs10);
Activity_18e7xme->addOutgoing(
    Flow_0tjjjt4);
Activity_1dvya6k->setName("Send invoice
    ");
Activity_1dvya6k->addIncoming(
    Flow_0s0bokk);
Activity_1dvya6k->addOutgoing(
    Flow_0jn14rw);

Gateway_0wv9qhz->setName("Credit Ok?");
Gateway_0wv9qhz->addIncoming("
    Flow_0p94vgm");
Gateway_0wv9qhz->addOutgoing("
    Flow_0ucjs10");
Gateway_0wv9qhz->addOutgoing("
    Flow_1crm1f5");
Gateway_1imf3p3->setName("Credit Ok?");
Gateway_1imf3p3->addIncoming("
    Flow_0tjjjt4");
```

```
Gateway_1imf3p3->addOutgoing("Flow_0s0bokk");
Gateway_1imf3p3->addOutgoing("Flow_0gzhp71");

Event_1bnog1v->setName("Order failed");
Event_1bnog1v->addIncoming(Flow_1crm1f5);
Event_1bnog1v->addIncoming(Flow_0gzhp71);
Event_1bnog1v->setName("Order completed");
Event_1bnog1v->addIncoming(Flow_0jn14rw);

Flow_04g6ahk->setSourceRef(Activity_1pqr5i0);
Flow_04g6ahk->setTargetRef(Activity_16br0gr);
Flow_0p94vgm->setSourceRef(Activity_1i1n9p1);
Flow_0p94vgm->setTargetRef(Gateway_0wv9qhz);
Flow_0ucjs10->setSourceRef(Gateway_0wv9qhz);
Flow_0ucjs10->setTargetRef(Activity_18e7xme);
Flow_0tjjjt4->setSourceRef(Activity_18e7xme);
Flow_0tjjjt4->setTargetRef(Gateway_1imf3p3;
Flow_0s0bokk->setSourceRef(Gateway_1imf3p3);
Flow_0s0bokk->setTargetRef(Activity_1dvya6k);
Flow_0jn14rw->setSourceRef(Activity_1dvya6k);
Flow_0jn14rw->setTargetRef(Event_02forxp);
Flow_1crm1f5->setSourceRef(Gateway_0wv9qhz);
Flow_1crm1f5->setTargetRef(Event_0v28q64);
Flow_0gzhp71->setSourceRef(Gateway_1imf3p3);
Flow_0gzhp71->setTargetRef(Event_0v28q64);
```

.

3)

```
// Instantiate objects
std::map<string, Introspectable *> objects;
objects["Process_154zshl"] = Factory::create("Process");
objects["StartEvent_02fqmjk"] = Factory::create("StartEvent");
objects["Activity_1pqr5i0"] = Factory::create("Task");
objects["Activity_1i1n9p1"] = Factory::create("Task");
objects["Activity_1i1n9p1"] = Factory::create("Task");
objects["Activity_1i1n9p1"] = Factory::create("Task");
objects["Gateway_0wv9qhz"] = Factory::create("ExclusiveGateway");
objects["Gateway_1imf3p3"] = Factory::create("ExclusiveGateway");
objects["Event_02forxp"] = Factory::create("EndEvent");
objects["Event_0v28q64"] = Factory::create("EndEvent");
objects["Flow_1q59u7i"] = Factory::create("SequenceFlow");
objects["Flow_0p94vgm"] = Factory::create("SequenceFlow");
objects["Flow_0tjjjt4"] = Factory::create("SequenceFlow");
objects["Flow_0s0bokk"] = Factory::create("SequenceFlow");
objects["Flow_0jn14rw"] = Factory::create("SequenceFlow");
objects["Flow_1crm1f5"] = Factory::create("SequenceFlow");
objects["Flow_0gzhp71"] = Factory::create("SequenceFlow");

// Adjust their properties
objects["Process_154zshl"]->set("isExecutable", false);
objects["StartEvent_0fkrhzc"]->set("name", "Initial State");
objects["StartEvent_0fkrhzc"]->add("outgoing", "Flow_1q59u7i");

objects["Activity_1pqr5i0"]->set("name", "Receive Order");
objects["Activity_1pqr5i0"]->add("incoming", "Flow_1q59u7i");
objects["Activity_1pqr5i0"]->add("outgoing", "Flow_04g6ahk");

objects["Activity_1i1n9p1"]->set("name", "Check Credit");
objects["Activity_1i1n9p1"]->add("incoming", "Flow_04g6ahk");
objects["Activity_1i1n9p1"]->add("outgoing", "Flow_0p94vgm");

objects["Activity_18e7xme"]->set("name", "Fullfill Order");
objects["Activity_18e7xme"]->add("incoming", "Flow_0ucjs10");
objects["Activity_18e7xme"]->add("outgoing", "Flow_0tjjjt4");

objects["Activity_1dvya6k"]->set("name", "Send invoice");
objects["Activity_1dvya6k"]->add("incoming", "Flow_0s0bokk");
objects["Activity_1dvya6k"]->add("outgoing", "Flow_0jn14rw");


objects["Gateway_0wv9qhz"]->set("name", "Check ok?");
objects["Gateway_0wv9qhz"]->add("incoming", "Flow_0p94vgm");
objects["Gateway_0wv9qhz"]->add("outgoing", "Flow_0ucjs10");
objects["Gateway_0wv9qhz"]->add("outgoing", "Flow_1crm1f5");


objects["Gateway_1imf3p3"]->set("name", "Credit ok?");
objects["Gateway_1imf3p3"]->
```

```
add("incoming", "Flow_0tjjjt4");
objects["Gateway_1imf3p3"]->
add("outgoing", "Flow_0s0bokk");
add("outgoing", "Flow_0gzhp71");

objects["Event_1bnog1v"]->
set("name", "Order failed");
objects["Event_1bnog1v"]->
add("incoming", "Flow_1crm1f5");
add("incoming", "Flow_0gzhp71");

objects["Event_1bnog1v"]->
set("name", "Order completed");
objects["Event_1bnog1v"]->
add("incoming", "Flow_0jn14rw");

objects["Flow_04g6ahk"]->
set("sourceRef", "Activity_1pqr5i0");
objects["Flow_04g6ahk"]->
set("targetRef", "Activity_16br0gr");

objects["Flow_0p94vgm"]->
set("sourceRef", "Activity_1i1n9p1");
objects["Flow_0p94vgm"]->
set("targetRef", "Gateway_0wv9qhz");

objects["Flow_0ucjs10"]->
set("sourceRef", "Gateway_0wv9qhz");
objects["Flow_0ucjs10"]->
set("targetRef", "Activity_18e7xme");

objects["Flow_0tjjjt4"]->
set("sourceRef", "Activity_18e7xme");
objects["Flow_0tjjjt4"]->
set("targetRef", "Gateway_1imf3p3");

objects["Flow_0s0bokk"]->
set("sourceRef", "Gateway_1imf3p3");
objects["Flow_0s0bokk"]->
set("targetRef", "Activity_1dvya6k");

objects["Flow_0jn14rw"]->
set("sourceRef", "Activity_1dvya6k");
objects["Flow_0jn14rw"]->
set("targetRef", "Event_02forxp");

objects["Flow_1crm1f5"]->
set("sourceRef", "Gateway_0wv9qhz");
objects["Flow_1crm1f5"]->
set("targetRef", "Event_0v28q64");

objects["Flow_0gzhp71"]->
set("sourceRef", "Gateway_1imf3p3");
objects["Flow_0gzhp71"]->
set("targetRef", "Event_0v28q64");
```

.

REFERÊNCIAS

[1] G. R. J.Desel, W. Reisig, "Business process management demystified: a tutorial on models, systems and standards for workflow management,," *Lectures on Concurrency and Petri Nets*, pp. 1–65, 2004.

[2] M. Weske, *Business Process Management: Concepts, Languages,Architectures*, 2007.

[3] W. M. Van der Aalst, "Business process management: a comprehensive survey." *International Scholarly Research Notices 2013*, 2013.

[4] J. Jeston, "What is business process management?" in *Business process management: practical guidelines to successful implementations*. Routledge, 2014.

[5] S. A. WHITE, "Introduction to bpmn," p. 0, 2004.

[6] ——, "Introduction to bpmn. ibm cooperation," 2004.

[7] E. DIAZ, "Towards a method to generate gui prototypes from bpmn," pp. 1–12, 2018.

[8] W. M. Van der Aalst, ""advances in business process management."data knowledge engineering," pp. 1–8, 2004.

[9] J. Owen, M; Raj, "Bpmn and business process management. introduction to the new business process modeling standard," 2003.

[10] white, "n.d." pp. 2–6).

[11] C. Systems, "Centrum systems, n.d."

[12] B. P. FORCIER, Jeff, "Python web development with django," 2008.

[13] P. BÄCHLE, Michael; KIRCHBERG, "Ruby on rails. ieee software," 2007.

[14] I. e. a. ZAFAR, "A novel framework to automatically generate executable web services from bpmn models," pp. 93 653–93 677, 2019.

[15] M. Dumas, "Case study: Bpmn to bpel model transformation."

[16] M. T. . L. L. Asztalos, M., "Generating executable bpel code from bpmn models," 2009.