FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Implementace BPMN modeláře na platformě OpenPonk
Student:	Bc. David Primus
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Práce navazuje na bakalářskou práci diplomanta "Podpora standardu BPMN na platformě OpenPonk". Cílem tohoto pokračování je jednak dokončení podpory pro všechny klíčové typy elementů a dále automatizované kontroly kvality modelu.
1. Seznamte se se standardem pro modelování procesních modelů BPMN. Nastudujte existující pravidla a doporučení pro modelování v BPMN notaci. Proveďte analýzu podpory kontrol BPMN modelů v nejpoužívanějších modelovacích nástrojích.
2. Seznamte se s aktuálním stavem platformy OpenPonk a zejména s frameworkem pro verifikace modelů.
3. Upravte/rozšiřte architekturu modulu pro BPMN modelování dle potřeb dalších typů elementů a implementaci verifikací. Doimplementujte další typy elementů BPMN do modeláře, navrhněte a implementujte sadu pravidel pro kontrolu kvality modelů a zobrazení výsledků uživateli.
4. Otestujte a zdokumentujte výsledek.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 5. října 2022 v Praze.

Diplomová práce

IMPLEMENTACE BPMN MODELÁŘE NA PLATFORMĚ OPENPONK

Bc. David Primus

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
2. května 2023

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

1 Úvod 1

2 Cíle a metodika 3

3 Teoretická část 5
3.1 Základy modelování procesů . . . 5
3.1.1 Workflow . . . 5
3.1.2 Flowchart . . . 6
3.1.3 Business Process . . . 6
3.2 Business Process Management . . . 6
3.3 Jazyk BPMN 2.0 . . . 7
3.3.1 Object Management Group . . . 7
3.3.2 ISO/IEC 19510:2013 . . . 8
3.3.3 Prvky jazyka BPMN 2.0 . . . 8
3.4 Validace BPMN . . . 16
3.4.1 Způsoby validace BPMN . . . 17
3.4.2 Pravidla BPMN podle specifikace BPMN 2.0.2 . . . 17
3.4.3 Pravidla BPMN podle Bruce Silvera . . . 19
3.4.4 Validace v nástroji SAP Signavio . . . 22
3.4.5 Přehled *Signavio Best Practice* pravidel . . . 23
3.4.6 Validace v nástroji bpmnlint . . . 25
3.4.7 Ostatní nástroje a doplňky pro validaci BPMN diagramů . . . 27
3.5 OpenPonk . . . 28
3.5.1 Pharo . . . 29
3.5.2 Spec 2 . . . 29
3.5.3 Roassal 3 . . . 30
3.6 Analýza současné implementace . . . 30
3.6.1 Analýza současné implementace BPMN modeláře . . . 31
3.6.2 Analýza implementace verifikace OntoUML . . . 34

4 Praktická část 37
4.1 Návrh validačních pravidel pro BPMN 2.0 . . . 37
4.1.1 Určení závažnosti validačního pravidla . . . 37
4.1.2 Provedení validace . . . 38
4.1.3 Validační pravidla . . . 38
4.1.4 Porovnání validačních pravidel s BPMN Method and Style . . . 42

4.2 Návrh a implementace 42
4.2.1 Návrh a implementace modelovacího nástroje pro BPMN 43
4.2.2 Návrh a implementace validace BPMN 48
4.3 Dokumentace a testování 52
4.3.1 Jednotkové testování 52
4.3.2 Testování validačních pravidel 54
4.4 Shrnutí výsledku implementace 55

5 Závěr **57**

A Dokumentace sady validačních pravidel k vzniklému nástroji **59**

Obsah přiloženého média **85**

Seznam obrázků

3.1 Všechny události jazyka BPMN 2.0. [15] 10
3.2 Zjednodušený přehled aktivit v BPMN 2.0. [15] 12
3.3 Ukázka validace v nástroji SAP Signavio. [24] 23
3.4 Ukázka validace v nástroji Camunda Modeler. [27] 26
3.5 Syntaxe jazyka Pharo zobrazená na pohlednici. [36] 30
3.6 Vzhled BPMN modeláře z bakalářské práce. [1] 31
3.7 Diagram základních tříd BPMN modeláře z bakalářské práce. [1] 33
3.8 Okno *Inspector* s výsledkem verifikace z balíčku OntoUML. [40] (Obrázek autora) 34
3.9 Zjednodušená architektura tříd OntoUML verifikačního balíčku. [40] (Obrázek autora) 35

4.1 Ukázka spolupráce konkrétních tříd včetně základních metod pro prvek *Activity* založené na architektuře MVC. (Obrázek autora) 43
4.2 Schéma tříd implementovaného balíčku pro validaci. (Obrázek autora) 50
4.3 Toolbar menu s možností spuštění validace. (Obrázek autora) 51
4.4 Ukázka tvorby validačního pravidla AN04 a provedení validace s chybami k odstranění. (Obrázek autora) 55
4.5 Test validačního pravidla AN04. (Obrázek autora) 55
4.6 Ukázka některých možných prvků k vytvoření v OpenPonk BPMN 2.0 rozšíření. (Obrázek autora) 56

Seznam tabulek

4.1 Porovnání s pravidly BPMN Method and Style 41

Seznam výpisů kódu

4.1 Ukázka implementace metody renderShapeDetails ve třídě OPBPMNNodeShape 46
4.2 Ukázka validace pravidla SE03. (*Start Event* typu *Message Receive* musí mít příchozí *Message Flow*.) 49

Chtěl bych poděkovat především svému vedoucímu bakalářské práce, doc. Ing. Robertu Perglovi, Ph. D., za cenné rady a podněty během tvorby práce. Také bych chtěl poděkovat celému týmu stojícímu za vývojem platformy OpenPonk, že mi umožnili se na nástroji podílet a byli ochotni sdílet své rady a zkušenosti.
Závěrem bych chtěl poděkovat své rodině, přítelkyni a přátelům za podporu nejen při tvorbě této práce, ale i během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 2. května 2023 .....

Abstrakt

Platforma OpenPonk je nástroj pro konceptuální modelování vyvíjený v systému Pharo. Cílem této práce je rozšíření platformy o BPMN 2.0 modelář, který zahrnuje také možnost validace BPMN 2.0 diagramů. V práci je navržena vlastní sada pravidel pro validaci založená na kombinaci pravidel z populárních nástrojů. Přínosem této práce je integrace BPMN 2.0 modeláře do sady podporovaných notací platformy OpenPonk a možnost použití nejen ve výuce.

Klíčová slova OpenPonk, Pharo, BPMN, BPMN 2.0, validace BPMN, pravidla BPMN, modelovací plugin

Abstract

The OpenPonk platform is a conceptual modeling tool developed in the Pharo system. The goal of this work is to expand the platform with a BPMN 2.0 modeler, which also includes the possibility of validating BPMN 2.0 diagrams. The thesis proposes a custom set of rules for validation based on a combination of rules from popular tools. The benefit of this work is the integration of the BPMN 2.0 modeler into the set of supported notations of the OpenPonk platform and the possibility of use not only in teaching.

Keywords OpenPonk, Pharo, BPMN, BPMN 2.0, BPMN validation, BPMN rules, modeling plugin

Seznam zkratek

BPMN	Business Process Model and Notation
BPM	Business Process Management
OMG	Object Management Group
OP	OpenPonk
XML	Extensible Markup Language
ISO	International Organisation for Standardization
SF	Sequence Flow
MF	Message Flow
AN	All Nodes
SE	Start Event
EE	End Event
BE	Boundary Event
IE	Intermedaite Event
GW	Gateway
AC	Activity
PR	Process

Kapitola 1
Úvod

V bakalářské práci „Využití platformy OpenPonk pro orchestraci webových služeb pomocí BPMN“ [1] byl na platformě OpenPonk [2] vytvořen základní modelář pro modelování *Business Process Model and Notation* (BPMN) diagramů. Základní sada prvků notace, tzv. paleta prvků BPMN Level 1 [3], je dostačující pro potřeby orchestrace webových služeb, není však dostačující pro plnohodnotné modelování procesních diagramů.

BPMN 2.0.2 je nejnovější verzí standardu jazyka BPMN [4]. Tento standard, společně s ISO certifikací [5] potvrzenou v roce 2022, upevňuje pozici jazyka BPMN jako nejrozšířenějšího standardu disciplíny *Business Process Management* (Procesní řízení). Procesní řízení je soubor činností, které slouží k plánování, sledování a zvyšování výkonnosti v realizaci podnikových procesů. Jednou ze základních činností správného zachycení průběhu procesu pro jeho následnou optimalizaci, je jeho namodelování pomocí jazyka BPMN.

Jazyk BPMN je společností *Object Management Group*, která standard vytváří, prezentován jako jednoduše pochopitelný jazyk pro širokou veřejnost. Model vytvořený v jazyce BPMN, který popisuje fungování podnikového procesu by měl pochopit dělník, který je přímým účastníkem vykonávání procesu, manažer, který proces řídí, ale také IT specialista, který dostane za úkol proces zefektivnit. Také vytváření samotného modelu by mělo být jednoduché a intuitivní tak, aby i přímý účastník procesu mohl zakreslit práci, kterou vykonává.

Z těchto důvodů je nutné nejen laikům v oblasti procesního modelování poradit, jak vytvořit validní BPMN diagram. Standard jazyka BPMN neposkytuje uživatelům přehledně zapsanou sadu pravidel BPMN modelování, existuje ale mnoho publikací a nástrojů, které základní validaci BPMN diagramů poskytují. Implementaci BPMN 2.0 rozšíření platformy OpenPonk je proto vhodné doplnit o možnost validace BPMN diagramů.

Platforma OpenPonk, která vznikla jako studentský týmový projekt na Katedře sofwarového inženýrství FIT ČVUT, nyní vyvíjená Centrem pro konceptuální modelování a implementaci [6], se zaměřuje na podporu notací z oblasti konceptuálního modelování. Do této oblasti se řadí již podporované notace UML, OntoUML, BORM, Petriho sítě, či konečné automaty. Ale také jazyk BPMN, který plnou podporu na platformě zatím nenabízí.

Plnohodnotná podpora standardu BPMN 2.0 na platformě OpenPonk by umožnila nástroj využít nejen při výuce jazyka BPMN na Katedře sofwarového inženýrství FIT ČVUT, ale také jej zapojit do dalších projektů, které jsou v rámci Centra pro konceptuální modelování a implementaci realizovány.

Kapitola 2
Cíle a metodika

Cílem této práce je vytvořit BPMN modelář pro platformu OpenPonk, který bude poskytovat sadu elementů procesního modelování na úrovni jazyka BPMN 2.0. Modelář bude doplněn o možnost kontroly kvality modelu, která bude probíhat pomocí navržených pravidel. Automatickou kontrolu modelu bude možné spustit stisknutím tlačítka a výsledek kontroly bude zobrazen uživateli v přehledné formě.

Teoretická část této práce se bude zabývat nejen jazykem BPMN 2.0, jeho vznikem a vlastnostmi jednotlivých prvků. Hlavním bodem této části práce je rešerše existujících pravidel jazyka BPMN, specifikace jazyka vydávané společností *Object Management Group*, ale také ostatních publikací a komerčních nástrojů, které tato pravidla aplikují. Dílčím cílem je zjištění způsobů, zda a jak jsou tato pravidla kontrolována a zobrazena uživateli v nejpoužívaněj- ších nástrojích.

Ještě před samotnou implementací je třeba analyzovat současný stav platformy OpenPonk, změny ve vývoji při přechodu na novou verzi prostředí Pharo 11. V mezidobí došlo také ke změně zobrazovacích knihoven, včetně změn architektury celé platformy. Analýzu musí doplnit rozbor současné implementace BPMN modeláře a implementace verifikačního frameworku pro OntoUML. Verifikační framework je již úspěšně používán pro kontrolu diagramu a anti-patternů v jazyce OntoUML. Analýza tohoto rozšíření platformy OpenPonk by mohla poskytnout základní návrh pro implementaci validace BPMN diagramů.

Cílem praktické části práce je navrhnout a poskytnout sadu validačních pravidel vycházející z několika zdrojů prezentovaných v rešerši. Je nutné návrh pravidel vytvořit tak, aby byl v souladu se specifikací jazyka BPMN. Tento návrh bude později zapracován do implementace jako doplněk pro validaci BPMN 2.0 diagramů. Modelář pro tvorbu BPMN diagramů musí být předělán tak, aby odpovídal architektuře celé platformy. Zároveň bude rozšířen tak, aby pokryl co největší množství konstruktů jazyka BPMN 2.0.

Finálním cílem celé práce je dosažené výsledky otestovat a prezentovat výsledek. K sadě validačních pravidel by měla být vytvořena online dokumentace v anglickém jazyce, která umožní lepší vysvětlení jednotlivých pravidel a nastíní, jak je možné provádět v nástroji uživatelské změny.

Kapitola 3
Teoretická část

Teoretická část této práce se zabývá zejména rešerší konceptu jazyka BPMN 2.0, konkrétně jeho prvky, pravidly pro modelování a validací BPMN diagramů. Nastíní funkci jazyka BPMN a jeho použití v různých oblastech. Závěrem bude představena platforma OpenPonk, základní verze BPMN modeláře, která je již v platformě dostupná, a framework pro verifikaci OntoUML diagramů, který by mohl poskytnout návrh pro implementaci validace BPMN diagramů.

3.1 Základy modelování procesů

Modelování procesů je často spojováno s počítači a automatizací. První publikace, která se zabývá touto problematikou je ale již z roku 1921. Jedná se o příručku s názvem *Process Charts*. Vzestup průmyslu po první světové válce vyžadoval vyšší a vyšší efektivnost, zabývaly se jí členové Americké asociace inženýrů (American Society of Mechanical Engineers).

„Procesní diagram je nástroj pro vizualizaci procesu a složí jako prostředek k jeho zlepšení.“ [7] Autoři dbali na to, že je třeba na papíře zachytit každý drobný detail. Zároveň musel být diagram jednoduchý na pochopení, kompaktní a rychle vytvořitelný. Výstupem publikace je dokonce jakýsi modelovací standard, který určuje jak má jaký prvek vypadat a jakou má funkci. Kupodivu se některé prvky od současného BPMN mnoho neliší.

Autoři nad celým konceptem kreslení diagramů přemýšleli stejně, jako je tomu v moderním *Business Process Management*. Jako první se má nakreslit diagram současného stavu, jak proces funguje. Poté je proces možné postupně na papíře vylepšovat, zlepšit vzájemnou kolaboraci a podchytit případné problémy. Finálním výstupem celého zdokumentování procesu pochopitelně není automatizace, jak ji známe dnes, ale zefektivnění a standardizace rutinní práce.

Celý diagram doplňuje formulář, který určuje odpovědi na základní otázky, které nejsou z obrázku čitelné. Otázky jsou položeny jasně a jednoduše: Kdo?, Kde?, Kdy?, Co?, Proč?, Komu? Oproti tomu současné BPMN má elementy, které nám na tyto otázky graficky odpovídají. [7]

Process Management je důležitou součástí podnikání již od dob prvních průmyslových revolucí. Technologie se zlepšují, ale princip kreslení procesních diagramů zůstává: „*Udělat první krok, k nalezení nejlepšího způsobu, jak vykonávat naši činnost.*“ [7]

3.1.1 Workflow

„Jedná se o organizovaný a opakovatelný vzor obchodních činností, který umožňuje systematická organizace zdrojů do procesů, které transformují materiály, poskytují služby nebo zpracovávají informace.“ [8] (Přeloženo autorem)

Workflow by se česky dal definovat jako daný, opakovatelný pracovní postup. Může být

nakreslen jako diagram zobrazující sled operací, práce osoby nebo skupiny. Zobrazuje jak jednoduché, tak složité mechanismy. Jedná se o tok aktivit nebo dokumentů ve firemním procesu. Může stanovovat jaký uživatel provádí jakou akci, k tomu je třeba mít stanovené kompetence.

Pojmy *workflow* a proces jsou často zaměňovány. *Workflow* se skládá z opakovatelných aktivit nutných k dokončení úkolu. Naproti tomu proces se týká všech prvků nezbytných k dosažení většího organizačního cíle. *Workflow* často zohledňuje podrobné detaily, ale proces se zaměřuje na komplexní výsledky. [9]

Používají se podobné termíny jako v procesním řízení. *Workflow* definuje akce, úkoly a procedury. Obsahuje řízení podmínek, toků a také jejich synchronizaci. Často je také používán termín proces pro běžící instanci *workflow*. Oba diagramy jsou na první pohled velmi podobné a pro laika můžou být matoucí.

Workflow se často používá ke stanovení podnikových standardů a aplikaci směrnic. Jedná se o jednoduchý způsob, jak dosáhnout přesného souladu s předpisy. Začaly se více používat s rozvojem průmyslu a stanovování podrobných předpisů v druhé polovině 20. století. Dnes existuje mnoho standardizovaných implementací. Některé s přímým napojením na business aplikace. Tomuto propojení se říká *Workflow Management Systems*. Umožňuje snazší monitorování a definování sekvencí procesů a úloh. Zároveň podporuje snižování nákladů a správně definovanou výměnu informací v podniku. [10] [11]

3.1.2 Flowchart

Flowchart, česky vývojový diagram, je druh diagramu, který reprezentuje *workflow* nebo proces. Je obecným označením všech diagramů, které rozkreslují řešení nějakého problému krok po kroku a rozdělují jej na jednotlivé úkoly. Může být definován také jako grafická reprezentace algoritmu.

Obsahuje obrazce různého tvaru jako jsou obdélníky, kosočtverce, kruhy, apod. Tyto tvary jsou vzájemně propojené šipkami. Obrazce definují jednotlivé kroky, šipky pak tok řízení. Šipky neoznačují tok dat, ale pořadí provádění operací. Vždy jsou směrové, aby bylo jasné pořadí provádění. Vývojové diagramy se nejčastěji čtou shora dolů, existují i varianty, které se kreslí zleva doprava. Nejčastějším symbolem pro jednotku práce je obdélník. Jedná se o dílčí krok zpracování diagramu, který je třeba vykonat atomicky buď celý, nebo vůbec. [7] [11]

3.1.3 Business Process

Podnikový proces, neboli *Business Process*, byl v bakalářské práci definovaný takto: *„Podnikový, či obchodní proces (Business Process) je sadou činností, které jsou prováděny v koordinovaném prostředí. Tyto aktivity společně realizují obchodní cíl. Každý obchodní proces je vykonáván jednou organizací, každá organizace je v podstatě organizovaná soustava procesů a činností, které na sebe vzájemně navazují, vzájemně interagují probíhají napříč organizačními jednotkami a reagují na různé podněty z vnitřního i vnějšího prostředí.“* [1]

3.2 Business Process Management

Česky procesní řízení nebo také řízení podnikových procesů, zkratkou BPM. Jedná se o manažerskou disciplínu, která současně využívá technologie procesního řízení. Procesní řízení se zaměřuje na řízení celého životního cyklu podnikání, což zahrnuje zvládnutí mnoha změn. Vyjádření firemní kultury, vysvětlení myšlenek BPM od posledního dělníka až po nejvyššího manažera. Zároveň se BPM musí postarat o zvládnutí změn v čase. [12]

Vizuální vyjádření podnikového procesu proto musí být jednoduché a snadno srozumitelné. Tento diagram na pozadí obsahuje složité koncepty, metodologie a administraci. Definování podnikového procesu vede subjekty k jejich analýze a zefektivnění.

Podnikové procesy jsou základním nástrojem procesního řízení. Díky tomu se spojuje několik různých odvětví a také dvě odlišné komunity zabývající se IT prostředím. Specialisty zkoumající formální metodologií procesů, kteří objevují strukturální vlastnosti procesů. Jejich abstrakce se snaží zachytit fragment reálného světa, což umožňuje zachytit důležité vlastnosti a struktury reálných podnikových procesů do abstraktního vyjádření. Druhou IT komunitou jsou software vývojáři, kteří vytváří komplexní a robustní systémy, které jsou často odtrženy od reality. Spojením procesních diagramů s vývojem rozděluje software na malé části, které jsou snadno udržovatelné a pochopitelné. [13]

BPM jako manažerská disciplína multidisciplinární obor zaměřený na proaktivní řízení podnikání. Jedná se o obor na pomezí mezi *Computer Science* a podnikáním. Obsahuje mnoho konceptů, jak pomocí manažerských metodik využívat již zmapovaný podnikový proces a snažit se jej více zdokonalit. Důležitými disciplínami jsou koučink, projektové řízení a řízení změn. Mezi nejčastější praktické příklady patří využití sbírání metrik za běhu procesu pro jejich následnou analýzu. Další z metodik je reinženýring procesu, který pomocí radikálních změn ve struktuře organizace slouží k odchycení slabých míst procesu.

Procesní řízení využívá mnoho modelovacích nástrojů. Model je upřednostňován před pouhým textovým zápisem. Používají se různé metodiky a notace, podle toho jakou část procesního řízení je potřeba zachytit. Mezi obecné modelovací nástroje pro zachycení modelování systémů se používá UML, DFD nebo Petriho sítě. K modelování procesů se nejčastěji používá Business Process Model and Notation, neboli BPMN. [12]

3.3 Jazyk BPMN 2.0

Nejnovější verzí jazyka BPMN je verze 2.0.2, která byla vydána v roce 2014. Tato verze standardu upevnila pozici BPMN jako nejrozšířenějšího standardu pro modelování podnikových procesů. Mezi základní vlastnost jazyka patří důraz na jednotnou grafickou vizualizaci a srozumitelnou čitelnosti diagramů také pro uživatele bez předchozí znalosti BPMN. Mezi výhody patří standardizovaný metamodel BPDM (Business Process Definition Metamodel), který díky definicím převodu do XML umožňuje snadnou přenositelnost diagramu a strojovou čitelnost.

Jak bylo řečeno v bakalářské práci [1], první verze standardu vznikla v roce 2004. Za touto verzí stála nezisková organizace BPMI (Business Process Management Initiative), která vznikla ze společné iniciativy 16 společností. Cílem této organizace bylo vytvořit nový, jednotný standard procesního modelování. Standard měl umožnit optimalizaci podnikových procesů a jejich částečnou automatizaci. Postupně se organizace rozrostla na 80 podniků a standard BPMN se stával nejpoužívanějším pro modelování podnikových procesů.

V roce 2006 došlo ke sloučení standardu s rozpracovanou verzí podobného jazyka, kterou vyvíjela společnost OMG (Object Management Group), tato společnost spravuje jazyk BPMN dodnes. Standard BPMN byl také ratifikován jako norma ISO/IEC 19510:2013 (viz kapitola 3.3.2).

3.3.1 Object Management Group

Object Management Group je konsorcium pro standardy počítačového průmyslu. Bylo založeno v roce 1989 skupinou technologických společností, mezi které patří například HP, IBM, či Apple. Původním záměrem bylo tvořit společné přenosné objektově založené standardy.

Významným počinem této společnosti bylo převzetí vývoje jazyka UML (Unified Modeling Language) a jeho následné vydání v roce 1997 jako standardu. Později byl také schválen jako ISO standard, tím se stal nejrozšířenějším jazykem konceptuálního modelování. V současné době se jedná o nedílnou součást návrhu a vývoje software.

OMG poskytuje pouze specifikace standardů, nikoliv jejich implementace. Proto byl vymyšlen postup pro schvalování standardů, aby byla zaručena jejich použitelnost v praxi. Členové skupiny

zadavatelů standardu musí zaručit, že do jednoho roku od vydání uvedou na trh produkt, který bude nový standard plně podporovat. Takto jsou nové standardy diskutovány a schvalovány čtyřikrát ročně na zasedáních konsorcia. Na druhou stranu se jedná o velice složitý proces, který může i jen opravení několika chyb z předchozí verze protáhnout na několik let. [14]

V roce 2021 společnost OMG oznámila vznik nové dceřinné organizace IIC (Industry IoT Consortium). Tato organizace se zaměřením na Internet of Things, by měla vytvářet standardy související s průmyslovým využitím internetu. I v této oblasti by měl najít využití standard BPMN, který dokáže kontrolovat složité business procesy a spojovat světy lidí, automatizace a internetu. [14]

3.3.2 ISO/IEC 19510:2013

International Organization for Standardization je světová federace národních normalizačních organizací. Jedná se o společnost založenou již v roce 1947 se sídlem v Ženevě. ISO/IEC je součást této federace, která se zabývá elektrotechnickými standardy.

Norma pro jazyk BPMN byla naposledy potvrzena v roce 2022, jedná se proto o stále platnou certifikaci. Norma je identická s OMG BPMN 2.0.1 standardem. V praxi se jedná o potvrzení BPMN jako nejpoužívanějšího standardu pro procesního modelování a snahu o to, aby se stal jediným používaným jazykem pro modelování podnikových procesů.

„ISO/IEC 19510:2013 představuje spojení osvědčených postupů v rámci komunity business modelování za účelem definování zápisu a sémantiky diagramů spolupráce, procesních diagramů a diagramů choreografie. Záměrem normy ISO/IEC 19510:2013 je standardizovat model obchodních procesů a notaci tváří v tvář mnoha různým modelovacím notacím a pohledům.“ [5] (Přeloženo autorem)

Podle normy jazyk BPMN poskytuje zápis, který je srozumitelný pro všechny uživatele, od obchodních analytiků, přes vedení společností, technické vývojáře až po lidi z byznysu, kteří budou takto zmapované prosesy řídit a monitorovat. BPMN tak vytváří most mezi světem návrhu obchodních procesů a jejich implementací. [5]

3.3.3 Prvky jazyka BPMN 2.0

V mé bakalářské práci [1] jsem se zabýval pouze deskriptivními prvky jazyka BPMN, neboli paletou prvků Level 1 podle Bruce Silvera [3]. Prvky tak byly rozděleny pouze do čtyř kategorií: tokové objekty, spojovací objekty, plavecké dráhy a artefakty. Tyto kategorie jsou zachovány, ale jsou společně s analytickou sadou, která představuje rozšíření o další typy prvků, pojmenovány jako procesní prvky.

První, tokové objekty definují chování procesů, jedná se o událost (*Event*), aktivitu (*Activity*) a bránu (*Gateway*).

Druhou, a to nově přidanou kategorií jsou data, která představují fyzické uložení dat nebo dokumentu. Rozdělují se na: datové objekty (*Data Objects*), datové vstupy (*Data Inputs*), datové výstupy (*Data Outputs*) a datové sklady (*Data Store*).

Třetí, spojovací objekty se rozrostly na čtyři typy hran: sekvenční toky (*Sequence Flow*), toky zpráv (*Message Flow*), asociace (*Association*) a datové asociace (*Data Association*).

Čtvrtá kategorie, plavecké dráhy, je beze změny a obsahuje pouze plaveckou dráhu (*Lane*) a bazén (*Pool*). Slouží k seskupení prvků do logických skupin. Pool může být někdy označován jako participant (účastník procesu, nejčastěji organizace) nebo jako proces samotný, jelikož proces z definice vykonává pouze jeden participant.

Poslední, artefakty, poskytují dodatečné informace o procesu. Jedná se o skupinu (*Group*) a anotaci (*Text Annotation*).

BPMN 2.0 nově odlišuje 4 typy diagramů, každý z nich vyžaduje jiný přístup k modelování a odlišnou paletu prvků:

Podnikové procesy Představují vnitřní podnikové procesy nebo veřejné procesy reprezentující interakci mezi více účastníky. Tyto procesy mohou být modelovány na různých úrovních abstrakce pomocí deskriptivních a analytických procesních prvků jazyka BPMN.

Choreografie Choreografie je definice očekávaného chování mezi dvěma účastníky. Využívají také procesní prvky, ale aktivity, které mají speciální označení, jsou chápány jako interakce reprezentující sadu výměn zpráv. Není zde žádná centrální entita.

Kolaborace Kolaborace je vzájemná interakce mezi dvěma nebo více entitami, které jsou reprezentovány pomocí bazénů. Výměna zpráv je zobrazena pomocí *Message Flow*. Kolaborace využívá procesní prvky jazyka, nemá své speciální grafické označení.

Konverzace Konverzační diagram je poslední přidanou součástí BPMN. Konverzace zobrazuje logickou výměnu zpráv, které jsou na sobě vzájemně závislé. Diagram konverzací obsahuje vlastní notaci a nemusí představovat proces, stojí proto stranou oproti ostatním typům. Nejsou zařazeny ani mezi základní kategorie prvků. [4]

3.3.3.1 Event

Dle Bruce Silvera [3] je důležité chápat význam událostí v BPMN 2.0 správně. Nejedná se pouze o definici, že se něco stane, ale o to, jak proces reaguje na signál, že se něco stalo. V tomto případě se jedná o tzv. *Catching Event*. Nebo v opačném případě, jak proces generuje signál, že se něco stalo. Poté se jedná o *Throwing Event*.

Catching Event má definovaný spouštěč, neboli *trigger*. Jakmile je spouštěč aktivován vnějším signálem, je spuštěn i tok procesu. Na podobném principu funguje i Throwing Event. Opět má definovaný spouštěč, který je aktivován přijetím toku procesu. Jeho výstupem je vygenerovaný definovaný signál, který je možné odchytit vně procesu.

BPMN 2.0 obsahuje přes 60 typů událostí. Bude představen koncept, jak se od sebe jednotlivé typy událostí liší. Základní tvar události je kruh, který může být vykreslen tenkou, silnou, dvojitou, či přerušovanou čarou, tím jsou určeny základní vlastnosti události. Uvnitř kruh může obsahovat nějakou ikonu, která určuje typ události.

Vykreslení kruhu tenkou čárou představuje *Start Event*, dvojitou čárou *Intermediate Event* a silnou tlustou čárou *End Event*.

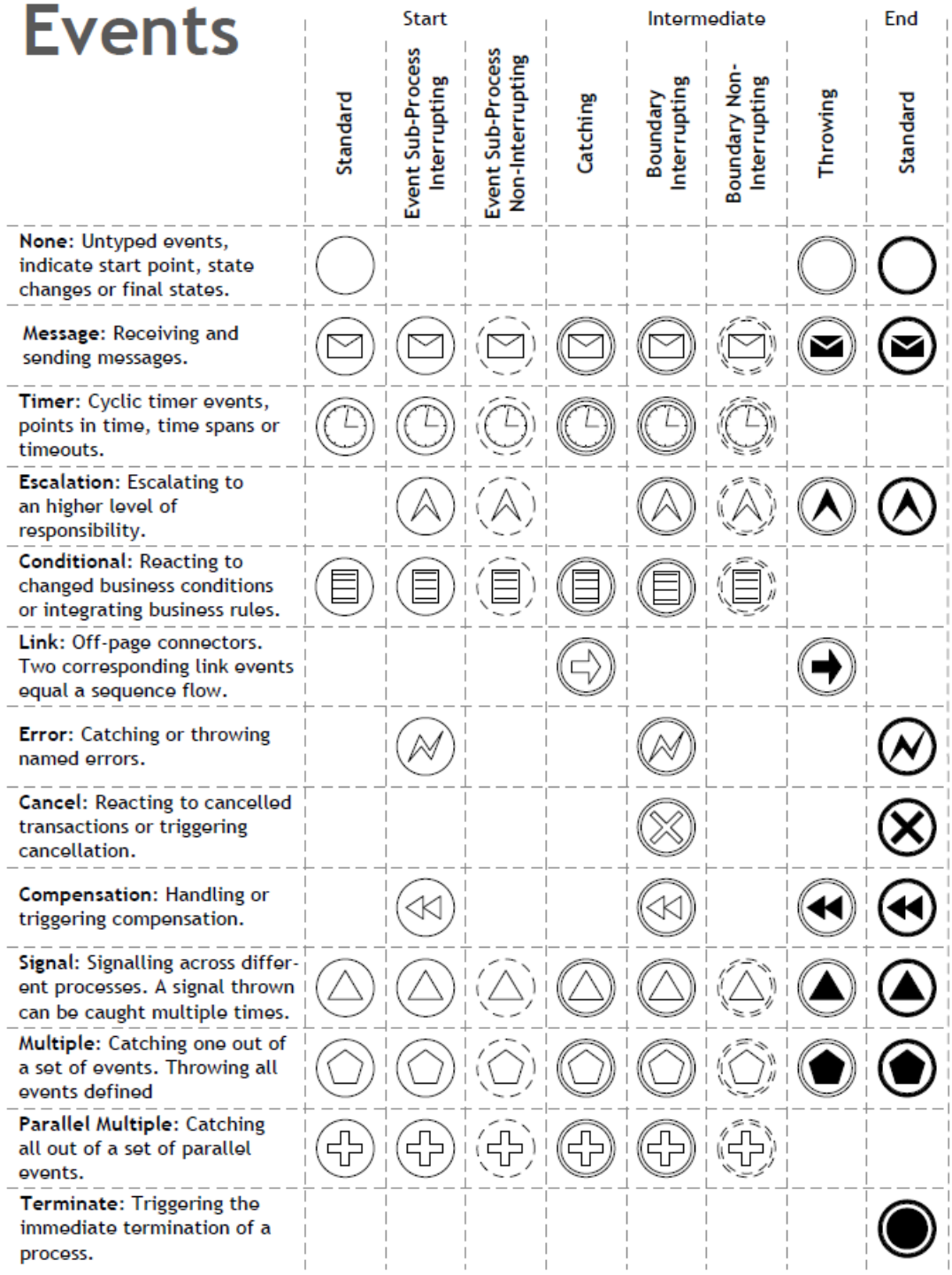
Styl čáry, plná vs. čárkovaná, odlišuje dopad události na celý běh procesu. Plná čára označuje přerušující (*Interrupting*) událost. Jedná se o události, které jsou spuštěny v okamžiku, kdy událost nastane, bez ohledu na to, zda byly okolní aktivity dokončeny. Čárkovaná čára představuje naopak nepřerušující (*Non-Interrupting*) událost. Jedná se o události, které budou čekat na dokončení aktivity předtím, než bude tok procesu pokračovat.

Catching a *Throwing* události jsou odlišené stylem ikony. Ikona může být bez výplně v případě *Catching Event* nebo může mít tmavou výplň v případě *Throwing*.

Start Event, jak název napovídá, indikuje jak a kde proces začíná. *Start Event* spouští tok procesu, nesmí proto mít příchozí *Sequence Flow*. V dokumentaci je tok jedné instance procesu označen jako virtuální *Token*, který je ve *Start Event* vygenerován a musí úspěšně projít celým procesem až do *End Event*. V případě, že neobsahuje žádný typ, nepřijímá žádný venkovní signál, a je proto spuštěna implicitně jedna instance procesu. Start Event se používá také jako počáteční prvek podprocesu, v tomto případě přijímá implicitní signál z rodičovského procesu a nesmí obsahovat ani příchozí *Message Flow*.

Intermediate Event indikuje událost někdy v průběhu procesu. Ovlivňuje běžící instanci procesu, ale nezačíná ji, ani nekončí. *Intermediate Event* může být použitý pro přijetí, či odeslání zpráv a signálů, indikaci průběžného stav procesu okolnímu světu, ovlivnění časové souslednosti nebo přerušení běžícího normálního toku procesu prostřednictvím zpracování výjimek.

Speciálním typem události je *Boundary Intermediate Event*. Jedná se o událost, která je zobrazena na hranici *Activity*, nejčastěji podprocesu. V případě spuštění přerušuje běžné chování



Obrázek 3.1 Všechny události jazyka BPMN 2.0. [15]

daného prvku, vytváří nový signál okolním prvkům procesu a přenáší tok instance do sebe. Jedním z použití je vytvoření výjimky. Proces vždy po projití *Boundary Event* nepokračuje běžnou cestou, často se jedná „Unhappy path“ vedoucí k výsledku běhu instance s nějakou

chybou,

End Event indikuje jak a kde proces končí. Neobsahuje proto žádnou odchozí *Sequence Flow*. Hlavní úlohou této události je korektní ukončení procesu. Všechny toky procesu musí být zpracovány před ukončením celého procesu, přesněji všechny vytvořené virtuální tokeny musí být zpracovány pomocí *End Event*. Proces může obsahovat více *End Event*, musí být proto kontrolováno, že instance procesu skončí až po zpracování posledního tokenu. Typ *End Event* určuje výsledek procesu, který předává dál mimo proces pomocí signálu. *End event* bez typu označuje nedefinovaný výsledek procesu, také není generován žádný signál, který by konec procesu indikoval. [4] [16] [17]

Nyní budou popsány některé nejdůležitější typy událostí. Ne všechny kombinace jsou v BPMN možné, seznam všech možných typů událostí lze vidět na obrázku 3.1.

- **Message** – Přijímání (*Catch*) nebo odeslání (*Throw*) zpráv. Většina procesů vyžaduje komunikaci s externími entitami, která je v BPMN reprezentována pomocí zpráv. Není určen typ zprávy, o který se jedná. *BPMN Message* reprezentuje každou informaci předávanou mezi odesílatelem a příjemcem. Dobrým zvykem je používat pro odesílání zpráv v průběhu procesu aktivitu *Send Task*, nikoliv *Throwing Intermediate Event*.

- **Timer** – Určuje čas spuštění, či dobu zpoždění. Je možné je modelovat jako odpočet nebo určit přesné datum a čas spuštění. Důležité je vždy řádně popsat, jaká úloha se má vykonat.

- **Escalation** – Podobné jako *Error* s tím rozdílem, že není přerušen normální tok procesu a běží paralelně s touto výjimkou. Používá se zejména v podprocesech.

- **Conditional** – Představuje jednoduchou jednostrannou podmínku, instance procesu pokračuje pouze pokud je splněna. Příkladem může být podmínka ”Trouba je rozehřáta na 180 C”.

- **Link** – Jedná se o speciální konstrukt jazyka, který lze využít místo sekvenčního toku. Ten je možné přerušit pomocí *Link Event* a druhým párovým *Link Event* se stejným popiskem jej zase navázat. Používá se v případech, kdy by *Sequence Flow* zhoršila čitelnost diagramu.

- **Error** – Pokud se v procesu může vyskytnout nějaká chyba, je možné ji identifikovat a zpracovat, či eliminovat. Chybu je možné odchytit pomocí *Boundary Event*, pokud není možné chybu zpracovat, může celá instance procesu končit chybou.

- **Cancel** – Jediné správné použití *Cancel Event* je v transakčním podprocesu. Takto je možné transakci přerušit.

- **Compensation** – V některých případech je potřeba kompenzovat chybný výsledek aktivity jinou akcí, která vrátí vše zpět do původního stavu, pokud je to možné. *Cancel Task* v tomto případě stojí samostatně mimo standardní běh procesu a je spuštěna pouze pokud je vyvolána příslušná *Throw Compensation* akce. Použití je doporučeno pouze ve složitých obchodních procesech, protože tento konstrukt není snadno pochopitelný na první pohled.

- **Signal** – Signály jsou podobné jako zprávy, lze je modelovat stejně. Jediným rozdílem je, že signál nemá adresáta, jedná se o broadcast. Signál proto může odchytit kdokoliv, kdo proces pozoruje.

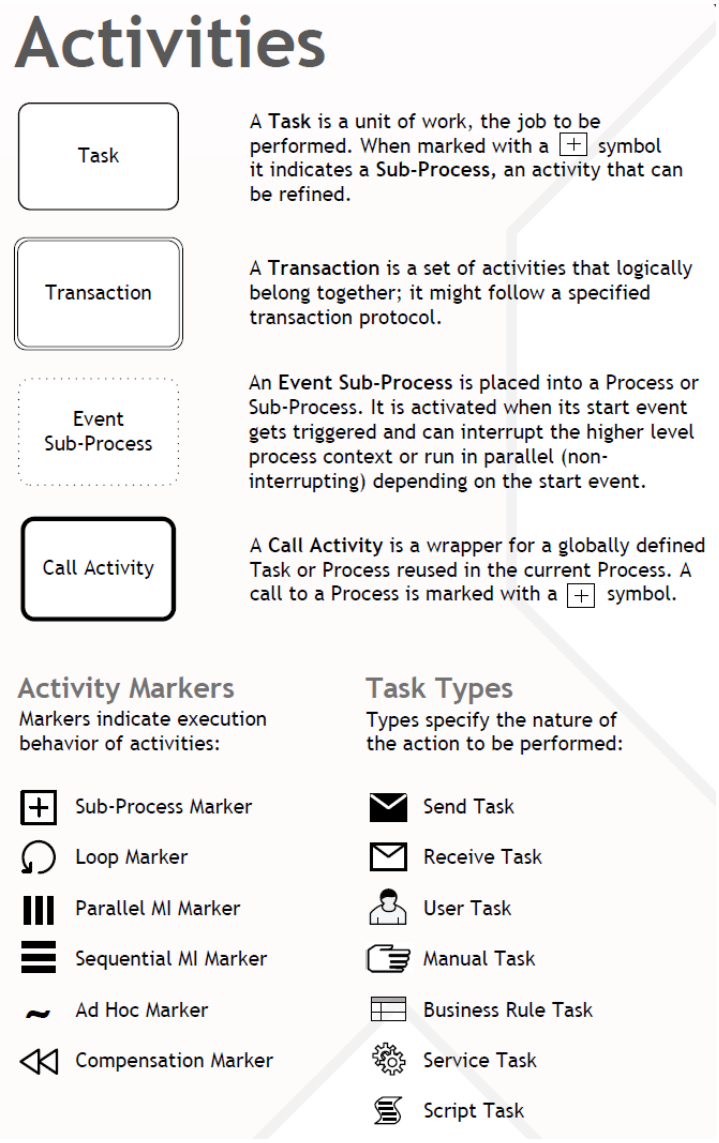
- **Multiple** – *Multiple Event* slouží ke spojení více různých událostí do jednoho symbolu. Z matematického hlediska představuje *Multiple Event* logický XOR.

- **Parallel Multiple** – Představuje stejně jako předchozí typ možnost pro provedení více událostí. V tomto případě je mezi všemi událostmi logický AND, což znamená, že musí být vykonány všechny události najednou, než může tok procesu pokračovat dál.

- **Terminate** – Tento typ události označuje, že všechny aktivity procesu musí okamžitě skončit. To zahrnuje všechny instance procesu. Proces je ukončen bez kompenzací i výjimek. [4] [16] [18]

3.3.3.2 Activity

BPMN 2.0 označuje slovem *Activity* abstraktní nadtřídu všech typů aktivit, tedy činností, které musí být vykonány. Jde o obecný pojem pro práci, která je vykonávána v rámci procesu. Jedná se o jediné prvky, které posouvají a mění děj procesu. Všechny jsou označeny obdélníkem se zaoblenými rohy. Jsou rozlišovány čtyři základní typy aktivit: *Task*, *Transaction* (transakce), *Event Sub-Process* a *Call Activity*. Zjednodušený přehled aktivit je zobrazen na obrázku 3.2.



Obrázek 3.2 Zjednodušený přehled aktivit v BPMN 2.0. [15]

Aktivity mohou obsahovat tzv. *Marker*, který indikuje chování aktivity při spuštění. Toto označení se nachází dole uprostřed a je možné i některé markery kombinovat spolu.

- **Sub-Process Marker** – Označuje složený (*Collapsed*) podproces. Tato aktivita by se měla jmenovat stejně jako podproces, který označuje. Některé nástroje umožňují přepínat mezi složeným a rozloženým podprocesem. Rozložený (*Expanded*) podproces neobsahuje žádné speciální označení.
- **Loop Marker** – Aktivita se opakuje, dokud není splněna přidružená podmínka.
- **Parallel Marker** – Označuje, že aktivita vytváří několik paralelně běžících instancí.
- **Sequencial Marker** – Označuje, že aktivita vytváří několik sekvenčně za sebou běžících instancí.
- **Ad-Hoc Marker** – Používá se u podprocesů, označuje, že jsou vnitřní aktivity prováděné v náhodném pořadí. Tyto aktivity nejsou spojeny hranami.
- **Compensation Marker** – *Compensation Activity* je volána z *Compensation Event*, jedná se o explicitní vyjádření, že aktivita musí stát sama o sobě bez připojení hran. [4] [15] [16]

Task může být buď jednoduchý úkol, který je atomickou částí a nelze dále dělit, nebo se pod Task řadí podproces (Sub-Process). Často je podproces řazen do vlastní kategorie. Podproces jako takový může existovat sám o sobě, jedná se vnořený proces, který má *Start* a *End Event* a další prvky z běžného procesu. Na hranici podprocesu mohou stát tzv. *Boundary Events*, které slouží pro předání výjimek, či dalších informací zpět do nadřazeného procesu. Podproces může být zakreslen jako *Collapsed* nebo *Expanded*, obě varianty mají syntakticky stejný význam.

Atomické *Task* představují nejjemnější úroveň detailu procesu. Vždy je provedena jedním vykonavatelem (uživatel nebo stroj) a je provedena buď celá nebo vůbec, případně končí celá s výjimkou. Neměl by být používán základní typ bez jakékoliv ikony, vždy by měl být použit buď marker, typ nebo kombinace obou.

Service Task je zpracovávána webovou službou, či jinou automatizovanou aplikací. Vždy je vykonána bez jakéhokoliv zásahu člověka.

Send Task slouží k odeslání zprávy jinému procesu. Na rozdíl od *Send Event* je možné v této aktivitě vytvořit obsah zprávy. Aktivita je ukončena po odeslání zprávy.

Receive Task čeká na zprávu od externího procesu. Jakmile je zpráva přijata, je aktivita ukončena.

User Task reprezentuje lidskou interakci. Je nutný zásah člověka s použitím softwarové aplikaci, aby byla aktivita ukončena. Patří sem například i pouhé potvrzení jinak automatizované akce.

Manual Task je vykonána bez použití jakýchkoliv technologií. Příkladem může být zapojení kabelu nebo ruční podepsání smlouvy.

Business Rule Task představuje mechanismus, který poskytuje vstup dat do procesního engine a získává výstup v podobě zpracovaných dat. Může se jednat například o výsledek analýzy podnikových dat.

Script Task je zpracována procesním engine. Aktivita definuje parametry pro skript, který je následně interpretován a spuštěn. Aktivita je ukončena, když skript doběhne. Výstupem je výsledek běhu skriptu.

Transaction je speciální typ podprocesu. Transakce je logická jednotka, která musí být vykonána najednou a nelze ji přerušit jiným signálem. Transakční podproces nesmí být znovu použit jiným procesem nebo modelem, proto se ve většině případů modeluje jako expandovaná. Transakce může mít jeden z následujících tří výsledků: 1. Transakce je úspěšně ukončena pomocí *End Event*. 2. Transakce končí s chybou, je použit *Cancel Event*. V tomto případě je vyvolána kompenzace a je spuštěna odpovídající *Compensation Activity* pro vrácení transakce do původního stavu. 3. Transakce je ukončena s kritickou chybou, pomocí *Error Event*. Jedná se o nepodchycenou chybu, v tomto případě nedojde k vyvolání kompenzace. Tuto výjimku je možné odchytit, ale není definované další obecné chování instance procesu.

Event Sub-Process je opět speciálním typem podprocesu, který je spuštěn nějakou specifickou událostí. Event Sub-Process proto nemá žádnou příchozí ani odchozí hranu a kreslí se vedle procesu. Jeho chování je podobné *Boundary Event* s rozdílem, že se vztahuje na celý proces, u kterého je nakreslen. Může být přerušující (*Interrupting*) nebo nepřerušující (Non-Interrupting) podle typu jeho *Start Event*. Nepřerušující lze spustit vícekrát. Přerušující lze pustit pouze jednou, ten pokud je spuštěn, ukončuje všechny aktivní instance procesu.

Call Activity umožňuje buď volat jiný proces jako součást původního procesu nebo použít jednoduchou *Task* globálně pro více procesů stejně. Jedná se o podobný princip jako u klasického *Collapsed Sub-Process*, ale tento proces/*Task* může být volaný z více různých procesů. Jedná se o samostatně uložený globálně (v rámci celé organizace) použitelný BPMN model. Notace podprocesu a *Call Activity* je oddělena z důvodu, že může být potenciálně změněno chování externího procesu bez vědomí uživatele původního procesu. Pokud se jedná o globálně využitelnou *Task*, je možné použít klasické typy jako *User*, *Service*, apod. [4] [16] [17]

3.3.3.3 Gateway

Gateway, česky brána, se používá k řízení toho, jak se sekvenční toky vzájemně ovlivňují. Pojem brána v češtině vyjadřuje místo, kde je třeba kontrolovat a regulovat průchod, tak je tomu i v BPMN. Brána povoluje, či zakazuje průchod dál procesem, nebo slučuje a rozděluje jednotlivé toky.

Na rozdíl od *Activity* a *Task*, brány nemají žádný způsob, jak ovlivnit chodu procesu, nejsou schopny jej změnit. Slouží pouze k usměrnění vstupu na výstup. Fakta musí být známy již na vstupu brány. Jedna brána musí mít buď více vstupů nebo více výstupů. Pokud má brána více výstupů, pak rozděluje tok podle daného typu a podmínky buď jedním směrem, nebo i více najednou. Pokud má brána více vstupů, slučuje toky, opět záleží na typu brány. Někdy pouští dál hned první přijatý token, jindy čeká na tokeny ze všech příchozích toků. Obecně platí, že není doporučeno používat jednu bránu, jak na spojování, tak na rozdělování toků. Také není doporučeno používat na spojení toků jiný typ brány, než na jejich rozdělení. [4]

BPMN nevyžaduje použití brány pro rozdělení toku. Je možné vytvořit aktivitu s více odchozími sekvenčními hranami, takto se ale proces dělí na paralelní toky. Následné případné spojení paralelních toků opět bez brány, již korektní není, protože aktivitu spouští první přijatý token. Možné je tok rozdělit bez použití brány a poté toky spojit pomocí *Parallel Gateway*, Druhou variantou je použít *Exclusive Gateway* pro rozdělení toků a poté je sloučit bez brány. Ani jedna z možností není snadno čitelná, a proto není doporučeno je používat. [3]

Exclusive Gateway představuje logický XOR. Vždy je jejím výstupem jediná exkluzivní větev. Pokud je podmínka stanovena tak, že ji nemusí žádná větev splnit, je nutné zakreslit do diagramu ještě jednu větev bez podmínky, která bude volena jako výchozí při nesplnění ostatních. Klasickým příkladem použití této brány je podmínka ano/ne a rozdělení na dvě větve. Ke spojení toků rozdělených pomocí *Exclusive Gateway* se opět používá *Exclusive Gateway*, která se spouští přijetím prvního tokenu.

Parallel Gateway se používá k rozdělení toku do více souběžných. Lze ji označit logickým AND. *Parallel Gateway* neobsahuje žádnou podmínku, jejím úkolem je rozkopírovat token do všech výstupních toků. K opětovnému spojení toků je potřeba použít *Parallel Gateway* znovu. Při spojování toků čeká tato brána na dokončení všech příchozích toků, až poté je dokončena. Při chybném modelování může použitím slučující *Parallel Gateway* vzniknout *deadlock*.

Inclusive Gateway představuje logický AND. Při rozdělení je aktivována alespoň jedna větev. Všechny aktivované větve musí být před synchronizací pomocí *Inclusive Gateway* dokončeny. U tohoto typu brány se často používají výchozí větve, protože se podmínky nemusí striktně vylučovat, ani pokrývat celý pravdivostní prostor.

Event-based Gateway, někdy označovaná jako exkluzivní a vždy modelovaná s dvojitým kruhem, nesměruje tok na základě podmínek, či přijatých dat, ale na základě událostí, které se nacházejí na všech výstupních větvích bezprostředně za bránou. Je vybrán tok události, která

se odehraje nejdřív. Přikladem může být čekání na příchozí zprávu, jeden tok obsahuje událost čekající na zprávu, druhý tok obsahuje časovou událost s maximální dobou čekání. Událost, která se stane dřív vyhrává, následně je provedena její větev toku. Důležitou vlastností je, že *Event-based Gateway* má k dispozici pouze jeden token, tedy pouze první tok je proveden. Ostatní zůstávají neaktivní. Z tohoto důvodu se často pro spojení toků rozdělených *Event-based Gateway* používá *Exclusive Gateway*.

Exclusive Event-based Gateway (instantiate), s jednoduchým kruhem, je speciálním druhem předchozího typu. Každá postupně splněná událost nacházející se za touto bránou spouští novou instanci procesu. Tím je myšleno, že je při každém splnění události vygenerován nový token, který je třeba při spojení správně synchronizovat.

Parallel Event-based Gateway (instantiate) je typem paralelní brány. Je umožněno spuštění více větví zároveň pomocí událostí. K následné synchronizaci je potřeba, aby byly všechny větve s událostmi spuštěny. Synchronizaci je třeba provádět pomocí *Parallel Gateway*.

Complex Gateway umožňuje složitější rozhodování, které není možné vykonat pomocí předchozích typů bran. Ve výsledku je možné kombinovat více různých podmínek, či jiných faktorů. Výsledkem vždy musí být alespoň jedna cesta. Opět je možné, pro případ nepokrytí všech podmínek, použít výchozí větev toku. Zajímavým příkladem použití v jazyce BPMN je, že ji lze použít také pro spojování paralelních toků, a to s určením podmínky, který tok bude přijat. [3] [16] [17]

3.3.3.4 Data

Pojmem data je označována kategorie čtyř prvků jazyka BPMN. Jedná se o zobrazení práce s informacemi a daty. Speciální typ hrany *Data Association* naznačuje směr toku dat, případně z jakých míst procesu mají být data viditelná.

Data Object přestavuje instanci lokální proměnné, či sady proměnných. Je viditelný pouze od úrovně procesu, na které je definovaný, k podrobnějším zobrazením procesu.

Data Store slouží k uložení a čtení dat procesu na externím úložišti. Úložiště se nachází mimo oblast procesu, je viditelné také pro další procesy.

Data Input a ***Data Output*** jsou volitelnými konstrukty jazyka BPMN. Nemusí být implementovány. Jsou podobné *Data Object*. V okamžiku, kdy je k dispozici instance *Data Input*, je spuštěna připojená aktivita. Naopak instance *Data Output* je vytvořena ve chvíli, kdy je ukončena připojená aktivita.

3.3.3.5 Spojovací objekty

Spojovací objekty tvoří základní kostru procesu. Definují pořadí a směr provádění jednotlivých částí procesu. Spojovací objekt vždy obsahuje informace o zdroji a cíli, ke kterým je připojen. Vždy musí obsahovat zdroj i cíl, není zakázáno, aby byl zdroj stejný jako cíl, ale z hlediska BPMN pro to není žádný vhodný důvod.

Sequence Flow, neboli sekvenční tok, se používá k zobrazení pořadí provádění jednotlivých akcí. Slouží k naznačení přesunu virtuálního tokenu mezi jednotlivými tokovými objekty. Sekvenční tok může probíhat pouze v rámci jednoho participanta, tedy pouze v rámci jednoho procesu. Není vhodné jej označovat za kontrolní tok, nemá žádný způsob, jak tok procesu měnit, či kontrolovat. Kontrolovat tok může speciální typ *Conditional Sequence Flow*. Tento typ se značí na začátku tvarem diamantu. Obsahuje podmínku, který musí být splněna, aby mohl být tok vykonán. Tento typ se nesmí používat v kombinaci s bránou, protože duplikuje její funkci. Druhým typem sekvenčního toku je *Default Sequence Flow*. Tento typ se naopak používá bezprostředně za bránou. Defaultní sekvenční tok je spuštěn pouze v případě, že bráně nevyhovuje žádný jiný tok.

Message Flow slouží k zobrazení přenosu zprávy, či informací od odesílatele k adresátovi. Zprávy tak mohou být odesílány pouze mezi různými participanty – bazény. *Message Flow* může

být ještě označen symbolem zprávy, který jako nevybarvený představuje inicializační zprávu. Vybarvený symbol zprávy označuje ostatní zprávy pro přenos dat.

Association, někdy nazývaná jako *Data Association*, slouží k zobrazení přiřazení dat aktivitě nebo procesu. Může být použita také jako směrová a označovat tím směr toku dat. [4] [17]

3.3.3.6 Plavecké dráhy

Koncept plaveckých drah a bazénů se využívá jako mechanismus pro organizaci aktivit do vizuálně oddělených kategorií. Účelem je ilustrování různých funkčních schopností nebo odpovědností.

Pool, česky bazén, představuje ohraničení jednoho procesu. Proces vždy probíhá pouze v jedné organizaci. Také je někdy označován jako participant, což je v tomto smyslu vnímáno jako jedna organizace. Je doporučeno nazývat bazén názvem celého procesu. Pokud se v celém diagramu vyskytuje pouze jeden bazén, nemusí být tento bazén zakreslen.

Pool může být zobrazen jako tzv. *Black Box*, prázdný bazén s názvem účastníka procesu. Takto je v diagramu zakreslen externí účastník procesu, se kterým není možné komunikovat pomocí sekvenčního toku, ale je třeba komunikovat prostřednictvím zpráv. Je však mimo náš rozsah, jak konkrétně procesy v tomto externím účastníkovi fungují.

Lane, neboli dráha, představuje vizuální oddělení částí procesu. Jsou takto oddělovány různé osoby/služby nebo role, které danou část procesu vykonávají. Všechny aktivity, které vykonává jedna osoba jsou organizovány do dané dráhy. Dráha je proto označována jménem vykonavatele části procesu. Dráhy můžou být zakresleny vertikálně nebo horizontálně. Pokud jsou zakresleny vertikálně, je definován směr čtení procesu shora dolů. Častějším způsobem je horizontální zakreslení, které definuje směr čtení procesu zleva doprava. Některé nástroje mohou podporovat pouze jeden směr kreslení drah. [4] [17]

3.4 Validace BPMN

Aktuálně nejnovější standard jazyka BPMN 2.0.2, ani žádné předchozí verze, neposkytují sadu syntaktických, ani validačních pravidel. [4] V textu specifikace jazyka od společnosti OMG, která standard spravuje, jsou zmíněná pouze pravidla týkající se jednotlivých elementů. Není zde pohlíženo na proces jako na celek, ani nejsou popsány složitější konstrukce, či vzory, jako je tomu například u jazyka OntoUml. [19]

BPMN standard obsahuje několik ukázkových procesů a vysvětluje základní principy, jak procesy modelovat. Neobsahuje však seznam „Best Pracice“ modelování v jazyce BPMN. Některé principy a pravidla jsou popsány příliš obecně. V některých případech je umožněno více variant, a není popsán preferovaný způsob modelování.

Příkladem může být například *Pool* (bazén), který lze podle specifikace kreslit vertikálně i horizontálně. Vertikální určuje směr čtení diagramu shora dolů, horizontální pak zleva doprava. Existují implementace, které použití omezují pouze na jeden typ, například Bizagi modeler, který umožňuje pouze horizontální *Pool*. [20]

Některé implementace BPMN 2.0 standardu definují vlastní sadu pravidel pro modelování. Nejpoužívanější je sada pravidel od Bruce Silvera, který ve své knize *BPMN Method and Style* vychází z dlouholetých zkušeností, které načerpal zejména při výuce jazyka BPMN. [3]

Správnost diagramu, tedy ověření definovaných pravidel modelování,je v BPMN modelářích nejčastěji kontrolována po ručním stisknutí tlačítka ověřit, či „Verify“. Výstupem je seznam porušení pravidel rozdělený minimálně do dvou kategorií:

- Error - Chyba, kvůli které není proces validní. (např. aktivita není připojena k diagramu pomocí *Flow*)
- Warning - Varování, které znamená, že proces je validní, ale obsahuje nejasnosti. (např. chybějící popis elementu)

[18]

Tato práce se zaměřuje na validaci BPMN diagramů, jako vizuálně nakresleného schématu. Je ale možné validovat XML reprezentaci BPMN, tzv. *XSD* (XML Schema Definition). To je důležité zejména v návaznosti na spouštění diagramů a automatizaci. Nevýhodou může být to, že se v tomto případě nedbá na vzhled diagramů jako takových.

V roce 2018 existovalo 47 nástrojů podporující standard BPMN 2.0, pouze 3 z nich dokáží přímo spouštět a tím i validovat nativní BPMN formát. Tyto nástroje dokáží ověřit spustitelnost XML reprezentace BPMN diagramu pomocí jejich *Process Engine*. Mluvíme o nástrojích *Camunda BPM*, *jBPM* a *Activiti*. Bohužel ani tyto tři nástroje nejsou jednotné, ani stoprocentní v analýze nevalidních diagramů. [21]

Ve studii *BPMN 2.0: The state of support and implementation* [21] byl proveden test chybovosti těchto 3 procesních enginů. Byla testována standardní XML reprezentace diagramů v podobě XSD. Celkem bylo vytvořeno 301 chybných BPMN schémat, která pokrývala 600 různých omezení jazyka BPMN (omezení vyplývající z XSD reprezentace, nikoliv přímo z BPMN standardu). Nejlepšího výsledku dosáhl nástroj *jBPM*, který dokázal detekovat pouze 74 % chybných diagramů. Výsledek ukazuje, že ani validace formálně uchopené reprezentace XSD formátu není jednoduchým úkolem.

3.4.1 Způsoby validace BPMN

Jazyk BPMN je možné validovat několika způsoby. Nemusí být řešen pouze vzhled diagramu, ale také jeho formální správnost.

Článek *Verification of BPMN 2.0 process models: An event log-based approach* [22] přichází s jednoduchým rozdělením validace BPMN do čtyř kategorií:

- **Validace podle stylu modelování** – Použití sady sémantických pravidel pro detekci chyb během modelování diagramu.
- **Validace s použitím simulačních technik** – Technika, která používá spouštění diagramu a simuluje různé cesty průchodem procesu. Používá se pro rychlou analýzu procesů ve velkých organizacích.
- **Formální validace** – Technika, která převádí model do formálních modelů, které jsou většinou založené na Petriho sítích.
- **Validace s použitím procesu, tzv. *Process-Mining*** – Technika, která porovnává logy událostí běžících procesů. Tím dochází k odchycení chyb a jejich možnému zpracování.

Tato práce se zabývá první technikou validace: Validace podle stylu modelování. V dalších kapitolách jsou porovnána nejzásadnější pravidla pro modelování procesů v jazyce BPMN.

3.4.2 Pravidla BPMN podle specifikace BPMN 2.0.2

532-stránková formální specifikace standardu jazyka BPMN neobsahuje žádné definice, jak validovat správnost BPMN modelů. Zaměřuje se zejména na vzhled jednotlivých prvků a možnosti propojení mezi nimi. *„Implementace, která vytváří a zobrazuje BPMN diagramy, MUSÍ odpovídat specifikacím a omezením s ohledem na vazby a další schematické vztahy mezi grafickými prvky.“* [4] (Přeloženo autorem)

V originálním textu jsou pravidla pro modelování velmi často označována slovem „MUSÍ“. Pokud se jedná o volitelnou, či doporučenou funkcionalitu, je v textu explicitně označena slovem „MŮŽE“ (*may*, či *should*). Tímto způsobem lze v textu nalézt klíčové vlastnosti jazyka BPMN. Následující přehled popisuje nejdůležitější pravidla a doporučení pro implementaci a modelování výsledných BPMN procesních diagramů, která jsou ve specifikaci řečena.

3.4.2.1 Použití barev a textu

Pro každý prvek z jazyka BPMN platí, že musí mít grafickou reprezentaci, která je v rámci jazyka unikátní a odpovídá standardu.

Každý prvek může mít popisky umístěné uvnitř, pod nebo nad prvkem.

Výplně prvků můžou být průhledné nebo barevné za dodržení několika podmínek: 1. výplň označení *Throw Events* (události, jejichž výstup opouští linii procesu - např. odeslání zprávy, vyvolání výjimky) musí mít tmavou barvu. 2. Účastník *Choreography Task* (Výměna zpráv mezi více účastníky procesu), který není iniciátorem, musí být zabarven světlou, nikoliv bílou, výplní.

3.4.2.2 Proces

Proces je množina prvků spojená pomocí hran. Množinou prvků rozumíme *Activities*, *Events* a *Gateways*, hrany tvoří *Sequence Flow*. Proces je znovupoužitelný, může volat jiné procesy nebo být volán.

3.4.2.3 Hrany

Hrana vždy vede mezi právě dvěma prvky a může směřovat libovolným směrem. Doporučením standardu je kreslit hrany vzájemně v pravých úhlech.

Sequence Flow nesmí překročit hranice podprocesu (*Sub-Process*) ani bazénu (*Pool*). To znamená, že nesmí spojovat dva prvky z různých bazénů, ani podprocesu s ostatními částmi procesu. *Sequence Flow* smí vést pouze z: *Start Event, Activity, Sub-Process, Gateway* a *Intermediate Event*. Jejím cílem může být pouze: *Activity, Sub-Process, Gateway, Intermediate Event* a *End Event*. Ostatní prvky, které nejsou vyjmenované, nesmí spojovat.

Podmíněná *Sequence Flow* jdoucí z *Activity* musí na svém začátku obsahovat značku ve tvaru diamantu. Naopak podmíněná *Sequence Flow* začínající v *Gateway* tuto značku nesmí obsahovat. Pokud má být hrana jdoucí z *Gateway* hranou defaultní, tedy použitou v případě, že všechny ostatní podmínky jsou nepravda, musí být na svém začátku označena zpětným lomítkem. Nutnost popisku u podmíněné hrany není ve standardu definována.

Message Flow nesmí spojovat prvky v rámci jednoho bazénu. Může vést přes hranice bazénu nebo být připojená přímo k němu. *Message Flow* smí začínat v: *Pool, Activity, Sub-Process, Intermediate/End Send Event*. Smí končit v: *Start Receive Event, Pool, Activity, Sub-Process* a *Intermediate Receive Event*. Volitelně může být na hraně zobrazena ikona zprávy. Pokud je zobrazena, musí mít zahajovací zpráva bílou/průhlednou výplň, další zprávy pak světle barevnou.

Association (asociace) smí spojovat pouze *Activity*, *Event* nebo *Flow* s Artefaktem (datové symboly a doplňující popisy). Asociace může být směrová a tím určit směr toku dat.

Conversation Link tvoří hranu pouze mezi *Pool, Activity* nebo *Event* a *Conversation Node*.

3.4.2.4 Uzly

Pool může být prázdný a vystupovat jako tzv. „Black Box“, nebo může obsahovat proces. Bazén vystupuje jako kontejner pro ostatní elementy. Pokud diagram obsahuje jen jeden *Pool*, nemusí být jeho hranice zobrazeny. Pokud jich obsahuje více, musí být být hranice bazénu zobrazeny a žádné elementy se nesmí nacházet mimo hranice bazénů.

Conversation Node smí být spojený dvěma nebo více *Conversation Link*. Slouží jako modelování komunikace mezi dvěma a více participanty (*Pool*). Z jednoho *Pool* může vést i více *Conversation Links*, z různých uzlů (např. různých aktivit).

Activity může mít nula, či více příchozích *Sequence Flow*. Pokud nemá žádnou, musí být aktivita spuštěna ve chvíli, kdy je proces spuštěn. Pokud má více příchozích hran, je aktivita spuštěna ve chvíli přijetí prvního signálu. *Activity* může mít také více odchozích *Sequence Flow*, v tomto případě je signál po skončení aktivity odeslán do všech odchozích hran najednou. Pokud nemá žádnou odchozí hranu, je větev procesu ukončena. Pro *Message Flow* platí stejná pravidla, aktivita může mít žádnou nebo více odchozích/příchozích *Message Flow*.

Start Event nemusí být v diagramu použita, musí být použita v případě, že diagram obsahuje *End Event*. *Start Event* nesmí mít příchozí *Sequence Flow*.

End Event nemusí být v diagramu použita, musí být použita v případě, že diagram obsahuje *Start Event*. *End Event* nesmí mít odchozí *Sequence Flow*. [4]

3.4.3 Pravidla BPMN podle Bruce Silvera

Bruce Silver Associates je jednou z největších a nejslavnějších společností poskytující školení a certifikace jazyka BPMN. Její zakladatel se proslavil knihou *BPMN Method and Style*, která slouží jako základ pro poskytovaná školení. Kniha nevysvětluje pouze významy symbolů jazyka BPMN, ale poskytuje metodiku, jak tvořit konzistentní, dobře strukturované a přehledné modely procesů. [23]

Kvůli své popularitě a obsáhlé kritice standardu jazyka BPMN byl Bruce Silver pozván jako jeden z konzultantů při tvorbě specifikace BPMN 2.0. Některé podněty, vycházející z jeho knihy, byly ve standardu použity, například jeho rozdělení prvků jazyka do dvou skupin - paleta levelu 1 (deskriptivní prvky) a paleta levelu 2 (analytické prvky). Bohužel nedokázal prosadit všechny své myšlenky, proto vznikla druhá edice knihy *BPMN Method and Style*. [3]

V knize nalezneme seznam pravidel pro procesní modelování v jazyce BPMN. Tento seznam v základu vychází ze specifikace jazyka BPMN 2.0, v některých bodech ji však jistým způsobem omezuje. Jedná se o seznam, který vychází ze zkušeností Bruce Silvera pro „dobrý“ model.

Za pravdu autorovi dávají i některé modelovací nástroje, které tato pravidla převzala jako verifikační pravidla pro BPMN modely. Například nástroj *SAP Signavio* obsahuje modul pro validaci diagramů podle *BPMN Method and Style*. [24] Podobně validují modely také nástroje *Microsoft Visio* a *Trisotech*. [23]

Následující seznam je volným překladem pravidel z knihy *BPMN Method and Style*. [3]

3.4.3.1 Sequence Flow

1. *Sequence Flow* musí spojovat tzv. *flow uzel* (*Activity, Event* nebo *Gateway*) na obou koncích. Ani jeden konec nesmí být nespojený.
2. Pokud proces obsahuje *Start Event*, musí mít všechny *flow uzly* mimo *Start Event, Boundary Event* a *Catching Event* příchozí hranu, *Sequence Flow*.
3. Pokud proces obsahuje *Start Event*, musí mít všechny *flow uzly* mimo *Ent Event* a *Throwing Event* odchozí hranu, *Sequence Flow*.
4. *Sequence Flow* nesmí překročit hranice procesu (*Pool*).
5. *Sequence Flow* nesmí překročit hranice podprocesu (*Sub-Process*).
6. *Podmíněná Sequence Flow* nesmí být použita, pokud je jedinou odchozí hranou elementu.
7. *Sequence Flow* vedoucí z *Parallel/Event Gateway* nesmí být podmíněná. U ostatních *Gateway* není podmíněný atribut (diamant na začátku podmíněné hrany) zobrazovaný.
8. *Activity* nebo *Gateway* můžou mít nejvýše jednu defaultní odchozí hranu.

Z této první sady pouze pravidla 2. a 3. lehce zpřesňují specifikaci. Ostatní pravidla ji odpovídají.

3.4.3.2 Message Flow

9. *Message Flow* nesmí spojovat prvky ve stejném procesu (*Pool*).
10. Zdrojem *Message Flow* musí být buď *Message/Multiple End Event*, *Throwing Intermediate Event*, *Activity* nebo *Black-Box Pool*.
11. Cílem *Message Flow* musí být buď *Message/Multiple Start Event*, *Catching Intermediate Event*, *Boundary Event*, *Activity* nebo *Black-Box Pool*.
12. Oba konce *Message FLow* musí být validně připojeny, ani jeden konec nesmí být nepřipojen.

3.4.3.3 Start Event

13. *Start Event* nesmí mít příchozí *Sequence Flow*.
14. *Start Event* nesmí mít odchozí *Message Flow*.
15. *Start Event* s příchozí *Message Flow* musí být typu *Message* nebo *Multiple*.
16. *Start Event* nesmí být typu *Error*, s výjimkou použití v podprocesu.
17. *Start Event* v podprocesu nesmí mít typ, pokud není spuštěn na základě nějaké události tzv. *Event*.

3.4.3.4 End Event

18. *End Event* nesmí mít odchozí *Sequence Flow*.
19. *End Event* nesmí mít příchozí *Message Flow*.
20. *End Event* s odchozí *Message Flow* musí být typu *Message* nebo *Multiple*.

3.4.3.5 Boundary Event

21. *Boundary Event* musí mít právě jednu odchozí *Sequence Flow*. (S výjimkou *Compensation Event*.)
22. *Boundary Event* může být pouze typu *Message, Timer, Signal, Error, Escalation, Conditional, Multiple, Cancel, Compensation* nebo *Multiple-Parallel*.
23. *Boundary Event* nesmí mít příchozí *Sequence Flow*.
24. *Error Boundary Event* v podprocesu musí obsahovat odpovídající *Error Throw Event*.
25. *Error Boundary Event* nesmí být typu *Non-Interrupting*.
26. *Escalation Boundary Event* v podprocesu musí obsahovat odpovídající *Escalation Throw Event*.

3.4.3.6 Throwing or Catching Intermediate Event

27. *Intermediate Event* s příchozí *Message Flow* musí být typu *Catching Message* nebo *Catching Multiple*.
28. *Intermediate Event* s odchozí *Message Flow* musí být typu *Throwing Message* nebo *Throwing Multiple*.
29. *Throwing Intermediate Event* může být typu *Message, Signal, Escalation, Link, Compensation nebo Multiple*.
30. *Catching Intermediate Event* může být typu *Message, Signal, Timer, Link, Conditional* nebo *Multiple*.
31. *Throwing Link Event* nemusí mít odchozí *Sequence Flow*.
32. *Catching Link Event* nemusí mít příchozí *Sequence Flow*.

3.4.3.7 Gateway

33. *Gateway* nesmí mít příchozí *Message Flow*.
34. *Gateway* nesmí mít odchozí *Message Flow*.
35. Rozdělující *Gateway* musí mít více než jednu odchozí *Sequence Flow*.
36. *Event Gateway* může ve svých větvích obsahovat pouze *Catching Intermediate Event* nebo *Receive Task*.

3.4.3.8 Process (Pool)

37. Proces musí obsahovat alespoň jednu *Activity*.
38. Prvky nejvýše jednoho procesu mohou být obsaženy v jednom *Pool*.
39. *Pool* v sobě nesmí obsahovat jiný *Pool*. Pokud jeho podproces obsahuje *Pool*, musí tento *Pool* odkazovat na stejného participanta jako proces nadřazený.

3.4.3.9 Stylová pravidla procesního modelování pro Level 2

Předchozí pravidla vycházela ze specifikace, případně ji upřesnila. Také se jednalo o pravidla, při jejichž porušení se stává model nevalidním. Následující pravidla upravují specifikaci jazyka tak, aby byl měl model význam procesu.

Popisky prvků

40. *Activity* by měla mít popisek, ideálně ve tvaru sloveso-podstatné jméno.
41. Dvě *Activity* ze stejného procesu nesmí mít stejné jméno, s výjimkou *Call Activity*.
42. Start Event obsahující nějaký spouštěč by měl mít odpovídající popisek: *Message Start Event* popisek ve tvaru „Přijímá zprávu (Receive)...“, *Timer Start Event* by měl indikovat čas spuštění procesu, *Signal Start Event* by měl obsahovat název signálu a *Conditional Start Event* obsahovat podmínku.
43. *Boundary Event* by měl obsahovat popisek.
44. Popisek *Error Boundary Event* podprocesu musí odpovídat popisku *Error End Event* obsaženým v podprocesu.

45. Popisek *Escalation Boundary Event* podprocesu musí odpovídat popisku *Throw Escalation Event* obsaženým v podprocesu.
46. *Throwing* a *Catching Intermediate Event* by měly mít popisky.
47. Odpovídající *Link Events* by měly mít odpovídající popisky.
48. *Throwing* a *Catching Events*, které náleží stejnému signálu, by měly mít odpovídající popisky, pokud se nachází ve stejném BPMN modelu.
49. Název *End Event* by měl odpovídat názvu koncového stavu.
50. Rozdělující *XOR Gateway* musí mít nanejvýš jednu nepojmenovanou odchozí hranu.
51. Rozdělující *XOR Gateway* nebo *Inclusive Gateway* musí mít samy popisek, pokud mají nějakou nepopsanou hranu.
52. Název podprocesu, který je zobrazen ve zvláštním okně, musí odpovídat názvu zakresleného prvku v rodičovském procesu.

End Event

53. Dvě *End Events* ve stejném procesu nesmí mít stejný název. Pokud by měly znamenat stejný koncový stav, je třeba je spojit do jedné.
54. Pokud je podproces následován rozhodující ano/ne *Gateway*, měl by název alespoň jedné *End Event* odpovídat této podmínce.

Expanse podprocesu

55. V podprocesu by měl být použit právě jeden *Start Event*. V případě, že je podproces zobraením paralelního průchodu, není použit žádný *Start Event*.
56. Podproces nesmí být nakreslený prostřednictvím *Expanded Sub-Process*, pokud je reprezentovaný jako oddělený diagram.

Message Flow

57. *Message Flow* by měla být popsána názvem zprávy.
58. *Send* prvek by měl mít odchozí *Message Flow*.
59. *Receive* prvek by měl mít příchozí *Message Flow*.
60. *Message Start Event* by měl mít příchozí *Message Flow*.
61. *Catching Message Event* by měl mít příchozí *Message Flow*.
62. *Throwing Message Event* by měl mít odchozí *Message Flow*.
63. Příchozí i odchozí *Message Flow* ze zabaleného podprocesu by měla být zobrazena, jak v podprocesu, tak v rodičovském procesu. [3]

3.4.4 Validace v nástroji SAP Signavio

„SAP Signavio Process Manager nabízí intuitivní, cloudové, profesionální modelování procesů. Komplexní řešení umožňuje organizacím dokumentovat, modelovat, navrhovat a simulovat procesy. Stejně tak dokáže simulovat jejich propojení za účelem vytvoření sdíleného porozumění, a tím v měřítku dosáhnout zvýšení výkonu.“ [24]

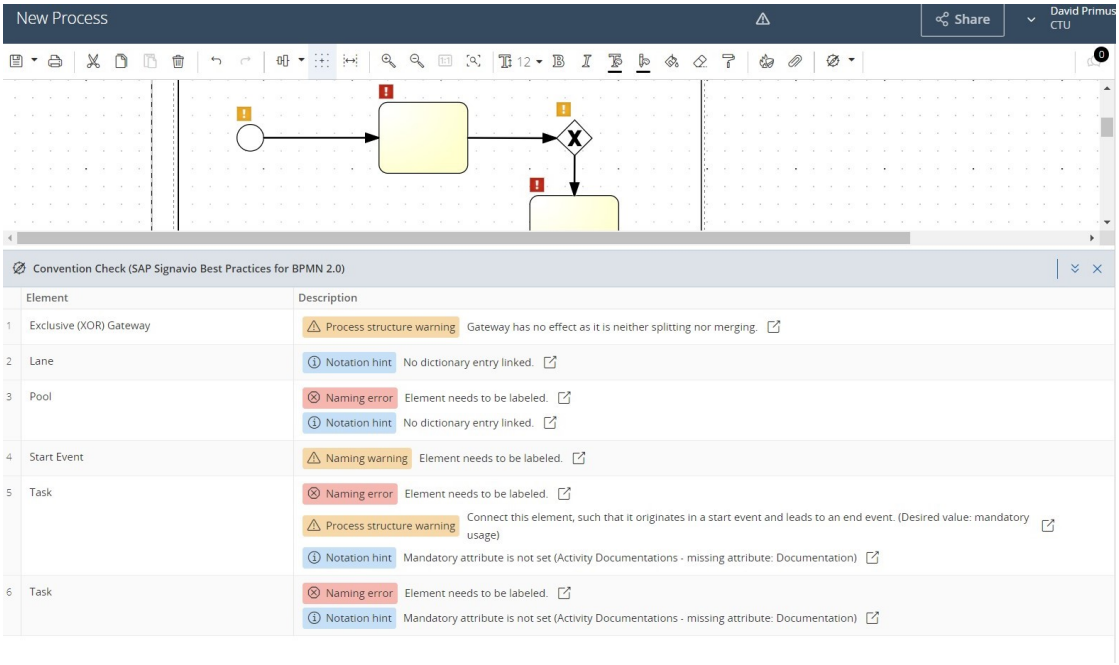
Signavio je jedním z komplexních BPMN modelovacích nástrojů, který podporuje plnou verzi BPMN 2.0, včetně spouštění procesů. Tento nástroj roce 2021 koupila společnost SAP, která

je známá především svým stejnojmenným *Enterprise Resource Planning* nástrojem. Díky spojení businessu a BPMN modeláře vzniká nástroj, který obě sféry propojuje. V současné době umí nástroj BPMN procesy tvořit, editovat, spouštět, validovat, transformovat, automatizovat a spravovat.

Pro tuto práci je zajímavá zejména část validační. SAP Signavio umožňuje procesy validovat dvěma způsoby: Podle konvence *BPMN Method and Style* nebo podle *SAP Signavio best Practices for BPMN 2.0*. Tento nástroj tedy přímo implementuje sadu pravidel *BPMN Method and Style* od Bruce Silvera představenou v předchozí kapitole.

K tomu přidává sadu vlastních pravidel, které ve svém nástroji po kontrole zobrazuje pouze jako nápovědy, či varování. Jedná se také o komplexnější pravidla, která kontrolují diagram jako celek. Zajišťují jeho dobrou čitelnost a zaměřují se také na možné průchody diagramem, tak aby jednotlivé cesty byly jednoznačné. Avšak některá uvedená pravidla jsou jedinečná pro tento komplexní nástroj a těžko použitelná v jiných případech.

Obě validace je možné kombinovat. Uživateli je v dolní části obrazovky zobrazeno okno s informacemi o porušení pravidel. Ke každému porušení je přiřazena závažnost: Error, Varování nebo Nápověda. Každý prvek, který nějaké pravidlo porušuje, je v diagramu označen čtverečkem v levém horním rohu. Při kliknutí na chybu v seznamu je prvek ještě více zvýrazněn. Porušení některého z první části pravidel *BPMN Method and Style* (do pravidla 39.) znamená chybu typu Error. Ostatní jsou typu varování, či Nápovědy.



Obrázek 3.3 Ukázka validace v nástroji SAP Signavio. [24]

3.4.5 Přehled *Signavio Best Practice* pravidel

Nápověda k nástroji obsahuje konkrétní popis ke všem pravidlům, včetně ukázek. [25]

1. **Absence deadlocks.** Zablokování (deadlock) vzniká v důsledku špatné kombinace bran. Dojde tím k blokaci dalšího procesního postupu, takto blokovaný proces není spustitelný.
2. **Absence překryvu hran.** Hrany by se neměly překrývat. Model může být nejasný, pokud by bylo toto pravidlo porušeno.

3. **Absence vícenásobných sloučení cest (multi-merges).** Vícenásobná sloučení jsou opakem deadlocku. Často se vyskytují, když se brány používají nesprávně, například při sloučení více různých cest bez použití brány. Je nutné se vícenásobnému sloučení vyhnout, protože může dojít k neočekávanému chování.

4. **Použití explicitních rozdělení toků za vykonanou aktivitou nebo *Event*.** Sekvenční toky by se měly dělit pouze pomocí bran, aby byla zajištěna jasnost a čitelnost diagramu.

5. **Použití explicitních rozdělení toků před vykonanou aktivitou.** Sekvenční toky by se měly spojovat pouze pomocí bran, aby byla zajištěna jasnost a čitelnost diagramu.

6. **Absence překryvu uzlů.** Uzly by se neměly překrývat s jinými uzly. Porušením tohoto pravidla může být model nejasný.

7. **Absence spojení a následného rozdělení v rámci jednoho uzlu.** Je třeba striktně oddělovat brány, které slouží ke slučování od bran, které slouží k rozdělení.

8. **Absence cyklů ve vztahu k podprocesům.** Vztahy k podprocesům by měly být přísně hierarchické. Pokud není toto pravidlo dodrženo, mohou se neúmyslně tvořit smyčky s neplatným/dvojitým propojením.

9. **Dodržení maximální velikosti diagramu.** Velké procesní diagramy jsou těžko pochopitelné a mají tendenci obsahovat více chyb. Maximální velikost diagramu v nástroji *SAP Signavio* odpovídá velikosti strany A3.

10. **Konzistentní využití názvů bazénů mezi podprocesem a hlavním procesem.** Při používání nadřazených procesů a podprocesů by měla existovat konzistence rolí. (Toto pravidlo odpovídá pravidlu 39. z *BPMN Method and Style*.)

11. **Konzistence názvů atributů s položkami v propojeném slovníku.** (SAP Signavio používá slovník objektů a jejich struktur. To umožňuje se lépe orientovat v dané doméně, zároveň je tak zaručena konzistence názvů napříč různými procesy. Jsou evidovány základní popisy objektů, zařazení do kategorií nebo také překlady do různých jazykových verzí diagramu. [26] Další pravidla týkající se slovníku nebudou v tomto textu zmíněna.)

12. **Hrana asociace by měla být modelována přímo bez ohybů.**

13. **Konzistentní modelování hran *Message Flow*.** *Message Flow* by měla být modelována vertikálně vzhledem k bazénu. Měla by vést co nejvíce přímo bez zbytečných ohybů.

14. **Konzistentní modelování hran *Sequence Flow*.** Sekvenční hrany je třba vést konzistentně pro snadnou čitelnost diagramu. *Signavio Editor* umožňuje ohýbání hran pouze pod úhlem 90 stupňů.

15. **Vzájemně konzistentní příchozí a odchozí *Message Flow*.** Konzistentní výměna zpráv mezi dvěma participanty.

16. **Vzájemně konzistentní příchozí a odchozí *Sequence Flow*.** Konzistentní výměna zpráv mezi dvěma uzly.

17. **Konzistentní použití *Start* a *End Event*.** Proces by měl mít alespoň jeden *Start Event* a jeden *End Event*. (Takové pravidlo nezmiňuje ani dokumentace, ani BPMN Method and Style.)

18. **Korektní použití *Boundary Event*.** Je důležité, aby byl *Boundary Event* korektně spojen s aktivitou a umístěn na jejím obvodu.

19. **Korektní použití *Conditional* a *Default Flow*.** Na jeden uzel může být maximálně jedna výstupní *Default Flow*. *Conditional Flow* musí mít odpovídající popisek, díky kterému je možné pomocí ano/ne možné určit, zda je podmínka splněna a tok se může vykonat.
20. ***Data Object, Event, Pool* a *Lane* musí být vždy pojmenované.** (Dokumentace jazyka BPMN vyžaduje pojmenování *Activity* a *Gateway*.)
21. **Definice správného směru modelování.** V *Signavio Editor* je možné určit směr čtení diagramu horizontální - zleva doprava, nebo vertikální - shora dolů.
22. **Korektní výměna zpráv mezi bazény.** Každý bazén musí být spojený s jiným pomocí alespoň jedné *Message Flow*. V diagramu nesmí být samostatný *Pool*, který není připojený, pokud není jediným *Pool* diagramu.
23. **Použití omezeného počtu zobrazených expandovaných bazenů.** Diagram by měl obsahovat v ideálním případě pouze jeden expandovaný bazén, ostatní by měly být typu *Black Box*. Jelikož každý bazén představuje vlastní proces.
24. **Správné použití *Gateway* pouze pro větvení a slučování toku.** *Gateway* musí mít více odchozích nebo příchozích hran.
25. **Proces, či podproces by měl mít pouze jeden *Start Event*.**
26. **Dodržení odpovídající vzdálenosti mezi prvky.** Mezi prvky by měla být vždy stejná a dostačující vzdálenost pro snadnou čitelnost.
27. **Unikátnost jmen uzlů diagramu.** Každý uzel by měl mít jedinečný název. [25]

3.4.6 Validace v nástroji bpmnlint

Nástroj bpmnlint je rozšíření pro *Camunda Modeler* [27] a *BPMN.io* [28], které umožňuje pokročilejší validaci BPMN diagramů.

Nástroj *Camunda Modeler* obsahuje sám o sobě základní validaci modelů v reálném čase. Zaměřuje se však na spustitelnost diagramu, nikoliv na sémantickou správnost. Jedním z příkladů může být chybová hláška: „¡Service Task¿ musí mít ¡Task definition type¿“. Musí být definovaná služba, která má být zavolána při spuštění diagramu. Naopak *Camunda Modeleru* nevadí brána, která nemá větvení, ani žádný jiný význam. Bohužel nebylo možné dohledat, co konkrétně *Camunda Modeler* kontroluje. Příkladem je obrázek 3.4. [27]

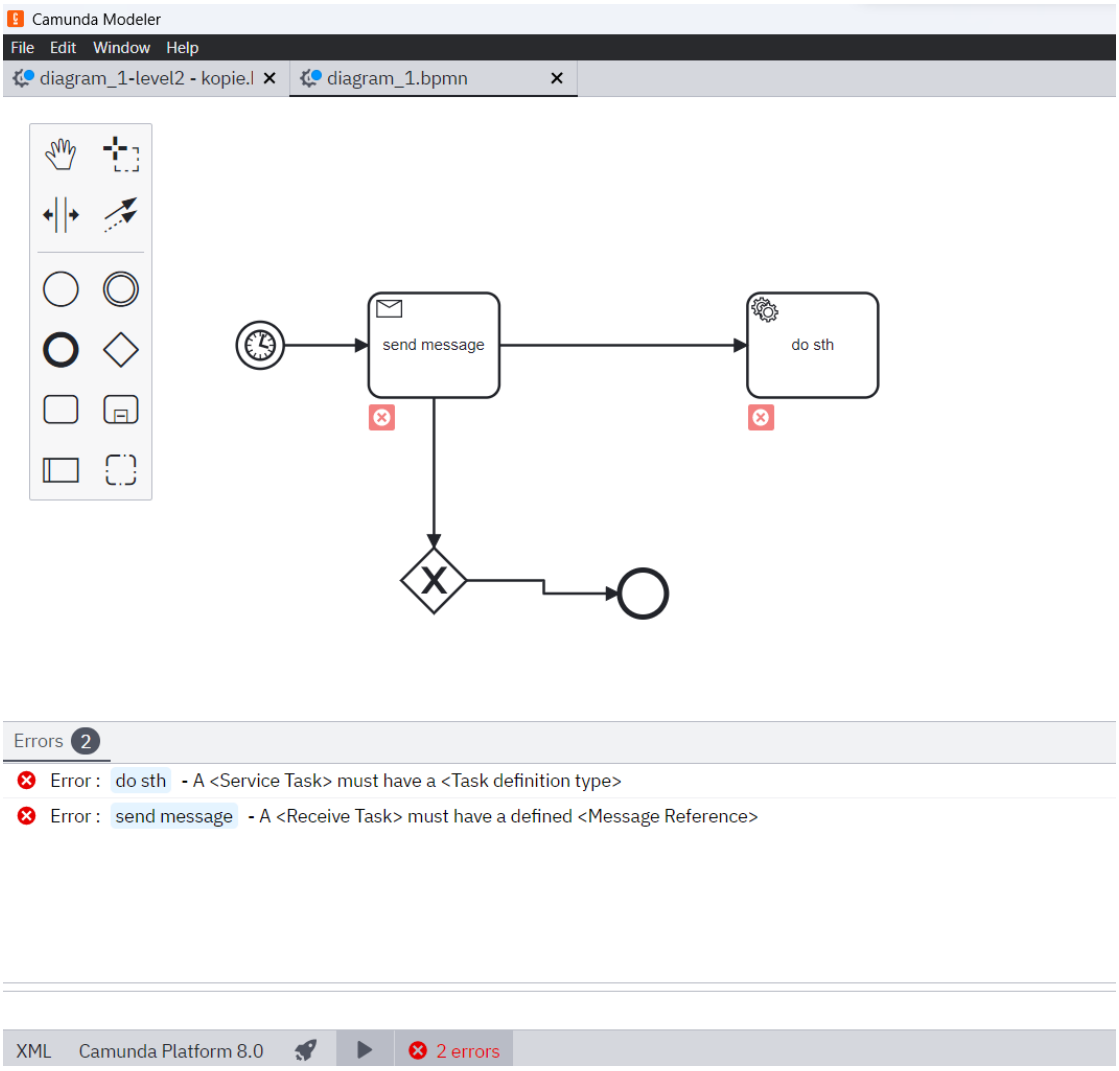
Nástroj *bpmnlint* je zveřejněn na *GitHub.com* pod *Open Source licencí MIT*, je tedy možné nahlédnout do zdrojového kódu, který je napsaný v jazyce *JavaScript*. Ze zdrojového kódu je patrné, že nástroj pracuje nad XML reprezentací jazyka BPMN.

Diagram je stromově procházen a nad každým prvkem je provedena kontrola pomocí sady pravidel. Před spuštěním validace je možné sadu pravidel upravit a některá pravidla vypnout. Každé pravidlo má své přizpůsobené chybové hlášky a úroveň závažnosti chyba (*Error*) nebo varování (*Warning*). Všechna porušení pravidel jsou uživateli v případě *Camunda Modeler* vypsaná na spodní části obrazovky do tzv. *Error Log*. Navíc prvky, které porušují nějaké pravidlo jsou v diagramu zvýrazněny červeným křížkem v případě chyby nebo žlutým otazníkem v případě varování. Také je uživateli zobrazen celkový počet porušení pravidel: chyby + varování. [29]

3.4.6.1 Seznam pravidel v knihovně bpmnlint

Nápověda k nástroji v sekci */docs/rules* obsahuje konkrétní popis ke všem pravidlům, včetně ukázek. [29]

1. ***Conditional Sequence Flow*. (Doporučené, chyba)** Podmínková hrana musí obsahovat podmínku.



Obrázek 3.4 Ukázka validace v nástroji Camunda Modeler. [27]

2. **Povinný *End Event*. (Doporučené, chyba)** Každý proces nebo podproces musí obsahovat *End Event*.

3. ***Event Sub-Process* obsahuje *Start Event* událostního typu. (Doporučené, chyba)**

4. **Chybné spojení hran. (Doporučené, varování)** Pro spojování, či větvení toků je vždy použita brána.

5. **Povinné popisky. (Doporučené, chyba)** Všechny uzly musí být popsány.

6. **Chybějící *BPMNDI* informace. (Doporučené, chyba)** Každý prvek odkazující na *BPMNDI* element by měl mít odpovídající data v XML reprezentaci. Jinak se nemusí správně vykreslit v diagramu. (Netýká se této diplomové práce.)

7. **Použití odpovídajících bran. (Doporučené, chyba)** Na spojení toků je nutné použít odpovídající *Gateway*, která byla použita na rozdělení toku.

8. **Žádné nepřipojené uzly. (Doporučené, chyba)** V diagramu neexistuje uzel, který by neměl vstupní, či výstupní hranu.

9. **Žádné duplikované hrany. (Doporučené, chyba)** V diagramu neexistují dvě hrany, které by měly stejnou funkci.

10. **Jedna brána pouze pro spojování, či rozdělování toků. (Doporučené, chyba)** Neexistuje brána, která by zároveň spojovala i rozdělovala toky hran.

11. **Žádné implicitní rozdělování toků bez použití *Gateway*. (Doporučené, chyba)**

12. **Nepoužívání *Inclusive Gateway*. (Doporučené, chyba)** *Inclusive Gateway* je špatně podporovaná v *BPMN Process Engines* včetně *Camunda BPM*. (Netýká se této diplomové práce.)

13. **Pouze jeden *Start Event* bez typu na proces, či podproces. (Doporučené, chyba)** Pokud je *Start Event* jiného typu, může jich být v diagramu více.

14. **Povinný *Start Event*. (Doporučené, chyba)** Každý proces, či podproces musí mít *Start Event*.

15. ***Start Event* bez typu v podprocesu. (Doporučené, chyba)** Podproces obsahuje pouze *Start Event* bez typu, protože je spouštěn vnější událostí.

16. ***Gateway* s pouze jednou vstupní a jednou výstupní hranou. (Doporučené, varování)** Brány, které nemají více než jednu vstupní a výstupní hranu jsou zbytečné, a proto by neměly v diagramu být. [29]

3.4.7 Ostatní nástroje a doplňky pro validaci BPMN diagramů

V předchozích kapitolách byly představeny nejrozšířenější a nejpoužívanější sady pravidel pro validaci BPMN diagramů. Existují ale také další nástroje, které validaci BPMN diagramů podporují. Pokud nástroj nebude obsahovat zajímavá, či rozšiřující pravidla, základní pravidla již nebudou vypsanána.

3.4.7.1 Activiti

Nástroj *Activiti* plně podporuje BPMN 2.0, včetně XML reprezentace, která je lehce upravena pro potřeby procesního engine. Podpora jejich vlastního procesního engine a tím i spouštění diagramů je dostupná až v nejnovější verzi Activiti 7. Nástroj se pyšní pečlivě zpracovanou dokumentací, ve které je pečlivě rozepsán každý prvek jazyka BPMN 2.0, jeho grafická notace, XML reprezentace a popis prvku včetně základních chyb.

Na pozadí tento nástroj pracuje v jazyce Java, ve kterém i validace provádí, a až poté jsou zvýrazněny v diagramu a XML. Interně jsou jednotlivé prvky diagramu uloženy jako Java objekty, to umožňuje nástroji snadnou validaci, konverzi do XML a další práci s prvky. [30]

Jsou validována základní porušení pravidel zhruba v rozsahu BPMN 2.0 specifikace, ve spustitelném diagramu jsou kontrolovány i chybějící atributy pro správné spuštění pomocí procesního engine. Jak bylo řečeno, obojí se provádí na úrovni Java tříd. [31]

3.4.7.2 Microsoft Visio

Placený nástroj *Microsoft Visio* obsahuje tlačítko, pomocí kterého je možné validovat BPMN diagramy. „*Abychom vám pomohli, ověřuje Visio váš diagram podle 76 logických pravidel souvisejících s vizuální správností vašeho diagramu a na základě standardu BPMN 2.0.*“ [32] (Přeloženo autorem)

Stejně jako předchozí nástroj, validuje *Visio* diagramy na úrovni vlastního objektového modelu. Zabývá se výhradně stylovými pravidly, která je možné libovolně vypínat nebo si nakonfigurovat vlastní. Všechny pravidla se týkají pouze základních vlastností prvků, hran a diagramu a byly řečeny v předchozích kapitolách. Jedná se o klasický koncept rozdělení úrovně závažnosti na chybu a varování.

Zajímavá je možnost konfigurace vlastních pravidel, kterou většina ostatních nástrojů neumožňuje. Je možné importovat sadu vlastních pravidel v podporovaném formátu nebo přímo v nástroji si napsat vlastní pravidla v jazyce C#. *Microsoft Visio* k tomuto účelu vydal celou příručku. [33]

3.4.7.3 Enterprise Architect

Enterprise Architect je společně s *Microsoft Visio* jedním z nejpoužívanějších modelovacích nástrojů, zejména v Enterprise/Business sektoru. Základem nástroje je plná podpora OMG UML standardu, doplněk tvoří mnoho dalších modelovacích standardů včetně BPMN 2.0. Jedná se proto o jeden z nejkomplexnějších nástrojů, který podporuje například i generování kódu přímo z UML diagramu.

Opět je možné spustit ruční validaci BPMN diagramu pomocí tlačítka. V nástroji *Enterprise Architect* jsou validační pravidla rozdělena do 12 kategorií, kterých je možné vypínat dle preferencí. Pravidla jsou spíše jednoduchého typu, částečně převzatá z notace UML. Výhodou je, že každá chybová hláška má své ID, které je možné podrobněji dohledat v dokumentaci nástroje. Tedy například u chyby „MVR05002“ „(Message Flow není validní v rámci hranice Pool)“ jsou podrobnosti snadno vyhledatelné. [34]

3.5 OpenPonk

OpenPonk je platforma pro metamodelování a modelovací pracovní nástroj implementovaný v dynamickém prostředí Pharo, zaměřený na podporu činností kolem softwaru a podnikového inženýrství, jako je modelování, provádění, simulace, generování zdrojového kódu atd.[2]

V současné době je nástroj rozšiřován a vyvíjen Centrem pro konceptuální modelování a implementaci na Katedře softwarového inženýrství FIT ČVUT. [6]

Platforma OpenPonk navazuje na nástroj OpenCABE, který vyvinul v roce 2011 Mgr. Martin Podloucký. Jedná se o nástroj pro modelování diagramů v jazyce BORM. OpenCABE je založený na platformě Eclipse. Základní myšlenky tohoto nástroje byly použity pro tvorbu nové platformy již založené na čistě objektově orientovaném jazyce Pharo. Nástroj DynaCASE vznikl v roce 2014 jako studentský týmový projekt. Později byl rozšířen a přejmenován na dnešní název OpenPonk. [6]

OpenPonk je zveřejněn pod open-source licencí MIT. Celý nástroj je rozdělen do několika oddělených balíčků, tzv. pluginů. Tím je umožněna snadná rozšiřitelnost a také uživatelská variabilita. Instalace vždy obsahuje jádro – základní sadu funkcionalit obsahující jak základní vlastnosti modelů, tak možnosti zobrazení a vykreslení. Další balíčky se vztahují k jednotlivým notacím, případně dalším rozšířením. Mezi podporované notace v současné době patří UML, OntoUML, BORM, FSM a Petriho sítě. [35]

V současné chvíli (duben 2023) je poslední hlavní zveřejněnou verzí verze 3.0.4 z roku 2021, která je doporučená pro Pharo 8. Ve vývoji je nová verze platformy určená pro Pharo 11. Nová verze obsahuje nové uživatelské rozhraní založené na knihovně Spec2. Také je implementováno

nové vykreslování diagramů založené na knihovně Roassal3. Tyto dvě knihovny vyžadují velký zásah do struktury platformy OpenPonk. Balíček pro modelování a validaci BPMN 2.0 diagramů bude implementován již pro novou verzi nástroje OpenPonk.

3.5.1 Pharo

Pharo je čistě objektově orientovaný programovací jazyk a zároveň výkonné prostředí (IDE s Operačním systémem sloučené do jednoho), zaměřené na jednoduchost a okamžitou zpětnou vazbu. Pharo boří hranici mezi programem a vývojovým prostředím. Celé prostředí Pharo funguje na principu virtuálního stroje, který představuje uzavřené prostředí Pharo. Je možné přímo použít kód pro vizuální reprezentaci datových struktur již během ladění, za běhu programu upravovat zdrojový kód nebo zdrojová data. V prostředí je možné vytvářet vestavěné aplikace dle vlastních potřeb, nebo je také zveřejnit a nechat upravovat dál dalšími vývojáři. [36]

Pharo je open source implementace klasického programovacího jazyka Smalltalk. Smalltalk je interpretovaný, dynamicky typovaný, čistě objektový jazyk, který byl vyvinut v roce 1972 společností Xerox PARC. Nejslavnější se stala verze Smalltalk-80, která tvoří základ mnoha komerčním a open source verzím, jako je právě Pharo. Mezi další známé hlavní implementace patří například Squeak, VisualAge Smalltalk, či Amber.

Hlavním rysem jazyka Smalltalk je čistá objektová orientovanost. Všechny datové typy jsou objekty, dokonce i blok kódu tvoří objekt. Každý objekt má tři základní vlastnosti: umí udržet svůj stav, umí obdržet zprávu od jiného objektu včetně sebe, může zaslat zprávu jinému objektu nebo sobě. Stav objektu je ostatním objektům skryt, obecně jsou všechny hodnoty Smalltalku ve stavu *protected* (viditelné pouze děděným objektům).

Celé prostředí Smalltalk je ukládáno pomocí *image souborů* (snímek paměti) a je spouštěn celý virtuální stroj, který je specifickým prostředím jazyka. To umožňuje přímo překládat zdrojový kód do mezikódu (*bytecode*) a rychle jej pomocí virtuálního stroje interpretovat. Celé prostředí se tak jednoduše skládá pouze ze tříd zapsaných v jazyce Smalltalk, tím je zaručena snadná správa. Je proto možné upravovat běžící program za běhu, prakticky okamžitě. Další výhodou vlastního virtuálního stroje a uložení práce v podobě *image* je snadná přenositelnost mezi různými platformami, stačí nainstalovat základní aplikaci prostředí, která je specifická pro platformu, snímky samotné však již ne a je možné je přenášet. [37]

Základní syntaxe jazyka Smalltalk obsahuje pouze pět vyhrazených klíčových slov: *true, false, nil, self* a *super*. Také Pharo se pyšní jednoduchou syntaxí, která samozřejmě vychází z původní. Všechna klíčová slova, a tedy základní syntaxe jazyka Pharo, se podle autorů jazyka vejdou na pohlednici, viz 3.5. [36]

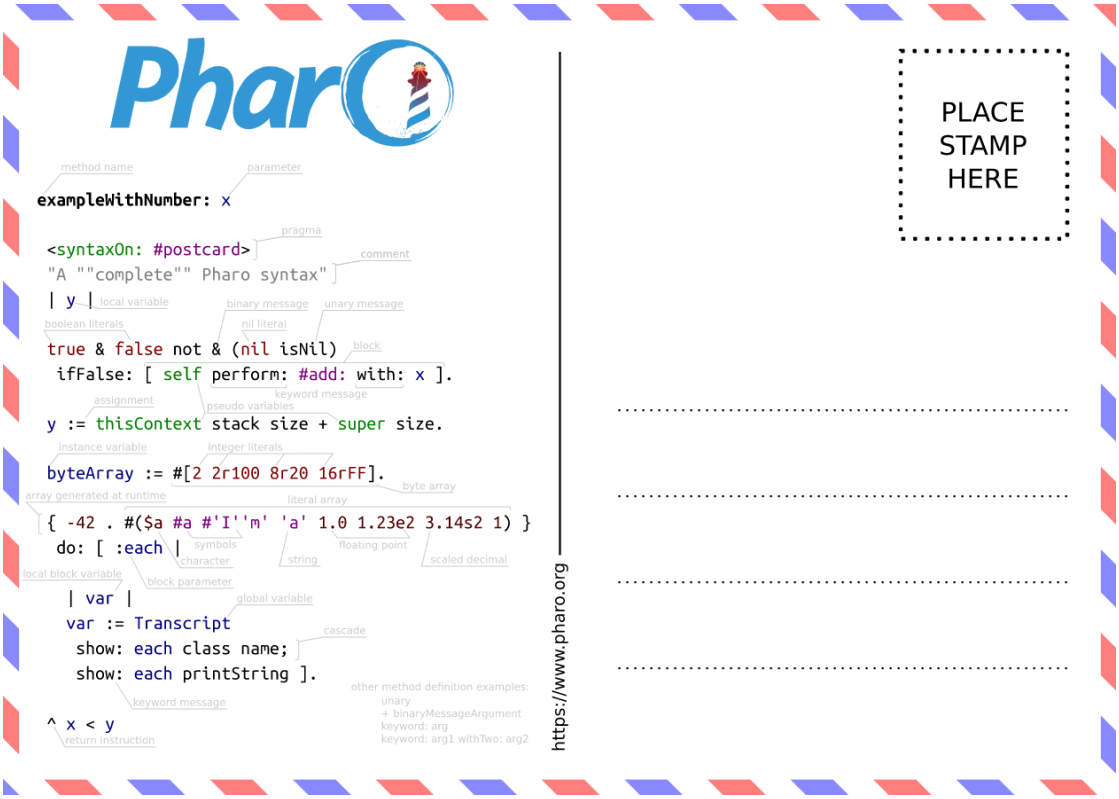
Aktuální hlavní verzí jazyka Pharo je verze 10, která vyšla v roce 2022. Již je připravována verze 11, která by však neměla obsahovat zásadní změny. Ve verzi 10 je standardně integrována vizualizační knihovna Roassal 3. V předchozích verzích jazyka byla podporována pouze knihovna Roassal 2. Pharo je relativně nový nástroj, který se stále vyvíí. Zároveň má silnou a aktivní komunitu uživatelů, která se snaží také o tvorbu dokumentace a dalších publikací. [38]

Platforma OpenPonk tak díky dynamickému prostředí Pharo umožňuje dalším uživatelům vyvíjet další funkcionality nebo si upravit již existující funkcionality dle vlastního vkusu. Obojí bez nutnosti složitých kompilací projektu, přímo za běhu celého prostředí.

3.5.2 Spec 2

„Spec2 není jen nová verze, ale kompletní přepsání a redesign Spec1. Na rozdíl od Spec1 jsou ve Spec2 všechna rozložení (layouts) dynamická. To znamená, že zobrazené prvky můžete měnit za běhu. Je to radikální vylepšení oproti Spec1, kde byla většina rozvržení statická a vytváření dynamických elementů bylo těžkopádné.“ [39]

Změna této verze znamená v kódu jádra OpenPonku mnoho změn. Nástroj je však navžen tak, že v jednotlivých pluginech není potřeba dělat zásadní opravy.



Obrázek 3.5 Syntaxe jazyka Pharo zobrazená na pohlednici. [36]

3.5.3 Roassal 3

Roassal 3 je vizualizační engine vytvořený v prostředí Pharo. Využívá se zejména k vizualizaci datových sad. Je možné vytvářet nejrůznější grafy, či interaktivní vizualizace. OpenPonk z tohoto nástroje používá zejména jednodušší objekty a tvary, které je možné vrstvit přes sebe, nastavovat jim různé vlastnosti a spojovat je pomocí hran.

Oproti verzi dva přináší Roassal 3 nejen velké množství nových funkcionalit, ale také velké změny v původních, včetně přepracování všech tříd a metod. Nová verze umožňuje jednodušší práci s již vytvořenými prvky, lepší možnosti jejich vrstvení a nastavování vlastností.

Vzhledem k novým možnostem byla přepracována architektura OpenPonku. Došlo k úplnému oddělení controlleru od vrstvy pro zobrazení prvků, pojmenované jako elementy diagramu. Bylo umožněno také hromadné přesouvání prvků nebo zjednodušena práce s hranami, jejich přesouvání a lámání podle předurčených bodů. [38]

3.6 Analýza současné implementace

Součástí OpenPonk verze 3.0.4 je modelář pro jazyk BPMN z bakalářské práce [1]. Cílem této práce je modelář pro BPMN rozšířit pro možnost modelování BPMN 2.0 diagramů. Současná verze implementace obsahuje pouze část jazyka BPMN 1.0, také je vytvořená pro původní verzi platformy OpenPonk, která již nebude podporována kvůli zastaralé architektuře.

Dalším cílem této práce je vytvořit možnost validování BPMN 2.0 diagramů, tj. pomocí sady pravidel ověřit, zda je daný diagram zakreslen sémanticky správně. Podobným tématem se zabýval Ing. Marek Bělohoubek ve své bakalářské práci [40], ve které vytvořil pro platformu

OpenPonk nástroj pro verifikaci OntoUML diagramů.

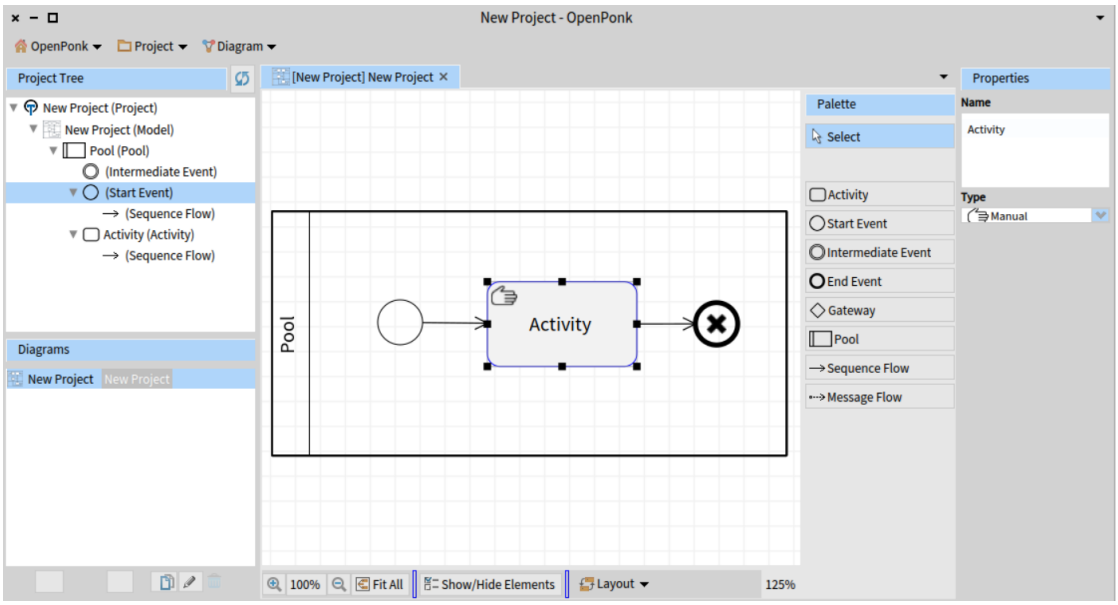
Následující analýza přiblíží a zhodnotí současnou implementaci BPMN modeláře a OntoUML verifikací.

3.6.1 Analýza současné implementace BPMN modeláře

Současná implementace BPMN rozšíření pro platformu OpenPonk [1] je vytvořena pro OpenPonk v. 3.0.4 běžícím ve Pharo 9. Implementace se skládá ze tří balíčků *OpenPonk-BPMN*, *OpenPonk-BPMN-XML* a *BaselineOfBPMN*.

Balíček OpenPonk-BPMN obsahuje základní modelář pro tzv. level 1 paletu BPMN 1.0 (označení dle prvního vydání knihy BPMN Method and Style [3]). Celkem se jedná o 25 různých prvků jazyka BPMN, které vychází ze šesti základních uzlů: *Activity, Start Event, Intermediate Event, End Event, Gateway* a *Pool*. Tyto uzly doplňují jejich podtypy pomocí ikon, které jsou umístěny uvnitř prvku. Uzly je možné spojovat pomocí dvou typů hran *Sequence Flow* a *Message Flow*. *Pool* je možné zvětšovat dle potřeby a umisťovat prvky přímo do něj. Bohužel není možné vkládat do bazénu dráhy, tento nedostatek by měl být v práci odstraněn.

Obrazovka modeláře (Obrázek 3.6) se skládá z několika panelů včetně plochy pro modelování. V horním menu se nacházejí tlačítka pro uložení diagramu, exportování, nastavení apod. V levé části jsou k dispozici informace o všech prvcích ve vytvořeném diagramu. V pravé části se nachází dva panely – první s paletou prvků ve formě tlačítek, druhý panel obsahuje podrobnější informace k vybranému prvku (na obrázku zvýrazněná Activity). První panel s paletou prvků byl v následující verzi OpenPonku přesunut na levou stranu obrazovky, jedná se o přehlednější zobrazení vůči panelu s vlastnostmi vybraného prvku, který byl takto špatně viditelný.



Obrázek 3.6 Vzhled BPMN modeláře z bakalářské práce. [1]

Implementace je založena na třívrstvé architektuře Model-View-Controller (MVC). Nejedná se o čisté MVC, tedy vrstvy nejsou striktně odděleny. Například informace o tvaru a umístění prvku jsou obsaženy v controlleru. Tyto nedostatky by měly být co nejvíce eliminovány v nové verzi. Architektura základních tříd, rozdělení do balíčků a napojení na jádro implementace OpenPonku je vidět na obrázku 3.7.

Úvodní třídou celého rozšíření je `BaselineOfBPMN`, která obsahuje informace o rozšířeních a jeho balíčcích. Třída obsahuje anotaci „baseline“, tato anotace je vyhledávána třídou jádra

implementace – `BaselineOfOpenPonk`, která registruje všechny dostupné pluginy (rozšíření). V případě BPMN rozšíření obsahuje *Baseline* informace o potřebné verzi jádra, XML doplňku a jaké balíčky pro BPMN mají být nahrány.

Další nastavení pluginu obsahují třídy `OPBPMNPlugin`, `OPBPMNNavigatorAdapter` a třída `OPBPMNLayouter`. `OPBPMNPlugin` slouží jako moderátor celého rozšíření, obsahuje informace o základních třídách z každé vrstvy MVC – *modelClass*, *diagramControllerClass* a *layouterClass*. Třída `Layouter` definuje rozložení zobrazení. V tomto případě je využito rozšíření Roassal 2, které umožňuje umisťování prvků na plátně. Prvky jsou vytvářeny pomocí základních tvarů, které jsou taktéž definovány v Roassal 2. Do třídy `Layouter` jsou navíc přidány dvě metody pro vrácení všech uzlů a hran v diagramu. `NavigatorAdapter` definuje levý panel *ProjectTree* – názvy prvků, jejich řazení, vnoření prvků a zobrazení ikon.

Třídy modelu představují jednotlivé prvky jazyka BPMN a ukládají o nich potřebné informace. U uzlů mezi tyto informace patří název, podtyp, výchozí podtyp, příchozí a odchozí hrany, případně vnitřní elementy, či další atributy. Třídy všech uzlů dědí základní vlastnosti (informace o příchozích, odchozích hranách a název) od třídy jádra `OPModelObject`. V rozšíření je zastřešuje třída `OPBPMNNodeModel`, která definuje další z jmenovaných vlastností. Každý typ uzlu pak obsahuje vlastní podtřídu pro definování dalších individuálních vlastností. Základní vlastnosti hran jsou také definované v implementaci OpenPonku ve třídě `OPDirectedAssociation`. Hrana obsahuje informace o svém názvu, zdroji a cíly, ke kterým je připojena. V pluginu jsou hrany implementované jako podtřídy `OPBPMNEdgeModel`.

Třídy modelu jsou hlavními instancemi pro kontrolu vlastností diagramu. Model obsahuje vlastnosti o sobě a připojených elementech, proto ve většině případů není třeba zkoumat vzhledovou stránku diagramu, ale pouze tyto sémantické vlastnosti.

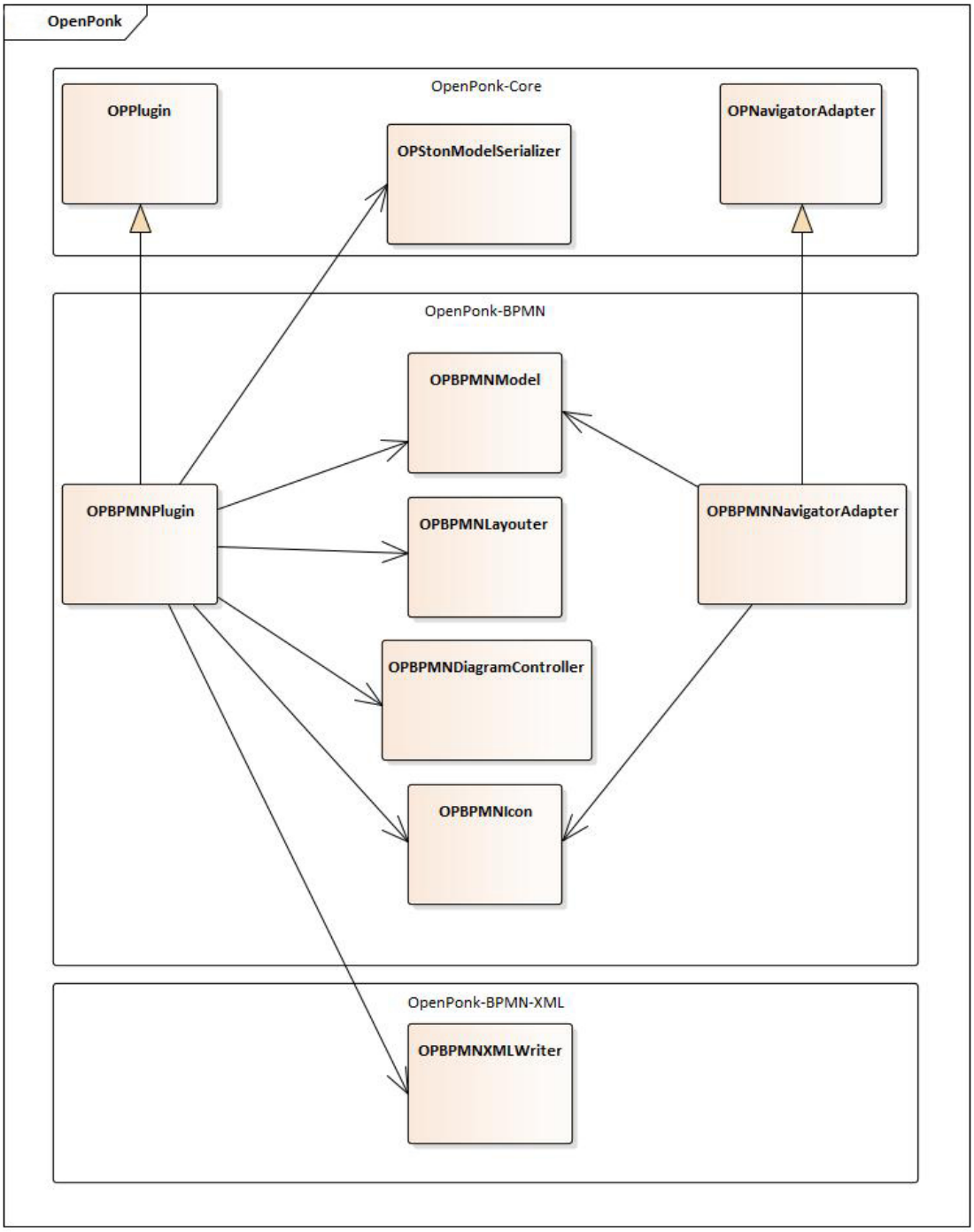
Třídy controlleru jsou párovými třídami modelu, ke každému modelu je definovaný právě jeden controller. Základní vlastnosti jsou opět definovány z jádra OpenPonku.

Controller pro všechny uzly s názvem `OPBPMNNodeController` dědí své vlastnosti od třídy `OPElementController`. Jsou zde definované vlastnosti pro změny a vykreslování uzlů. Metoda `buildEditorForm:` slouží k vytvoření pravého bočního panelu obrazovky (obrázek 3.6), jedná se o individuální nastavení vlastností vybraného uzlu. Například je pro všechny k dispozici textové pole pro vyplnění názvu. U některých uzlů, např. aktivit, je v panelu také *Select box* pro vybrání podtypu.

Další metody by se daly seskupit jako spojující a slouží k určení jak a s čím je možné prvek spojovat – `addAsTargetFor:` se ptá prvku, do kterého má být uzel umístěn, pokud je to možné, vytvoří se. `canBeSourceFor:` určuje pro jaké prvky může být uzel zdrojem (pouze hrany). `canBeTargetFor:` určuje pro jaké prvky může být uzel cílem (hrany a vnořené uzly), například bazén může být cílem pro všechny ostatní uzly a také pro *Message Flow*.

Druhou skupinu metod je možné označit názvem konstrukční, starají se o vytváření a aktualizace uzlů, patří sem – `createRoassalShape` vytváří základní tvar uzlu, nejčastěji obdélník, či kruh. Tyto tvary patří do rozšíření Roassal 2. `renderFigureIn:` vytváří celý prvek a předává ho diagramu pro umístění, v této metodě se k základnímu tvaru přidávají ještě další jako například popisek, či ikona. Na konci metody jsou také nastaveny vlastnosti prvku pro možnost editace na plátně, pro každý prvek může být vybrána libovolná kombinace následujících možností – *RTFocusable* pro možnost výběru, *RTDraggable* pro možnost přemístění pomocí myši, *RTResizable* pro možnost zvětšení uzlu. V této implementaci obsahují možnost zvětšování *Activity* a *Pool*. Ostatní uzly je možné označit a přemístit. `refreshFigure` slouží k aktualizaci vlastností prvku z modelu.

Controller hran je potomkem `OPDirectionalRelationshipController`, který definuje vlastnosti pro možnost vykreslení hran. Na rozdíl od uzlů, kde je uložen pouze cíl, zde se ukládá zdrojový i cílový prvek. Hrana je zakreslena do diagramu pouze tehdy, když má validní zdroj i cíl. Také je během zakreslování do diagramu při najetí myši na zdrojový/cílový prvek označen prvek zelenou, či červenou barvou, zda je, či není možné mu hranu přiřadit. O vytvoření tvaru se stará metoda `createEdgeFrom:to:`. Obě hrany v implementaci obsahují v této metodě vlastnost



Obrázek 3.7 Diagram základních tříd BPMN modeláře z bakalářské práce. [1]

withBorderAttachPoint, která určuje, že začátek a konec hrany vychází z okraje prvku, nikoliv z jeho středu.

`OPBPMNDiagramController` je párovou třídou k `OPBPMNModel`. Obě třídy společně představují kořenový prvek diagramu. Dalo by se říci, že jsou plátnem celého diagramu. `OPBPMNModel` obsahuje kolekci všech modelů prvků, umí oddělit hrany od uzlů. `OPBPMNDiagramController` ob-

sahuje kolekci všech controllerů a také informace, které prvky je možné do diagramu umístit, zajišťují metody `controllerFactory` a `elementsToShowInside`. `updateView` slouží k obnovení všech prvků, dochází k upozornění všech controllerů v diagramu. Třída také obsahuje metodu `initializePalette:` pro vytvoření palety prvků napravo od plátna (v novějších verzích vlevo).

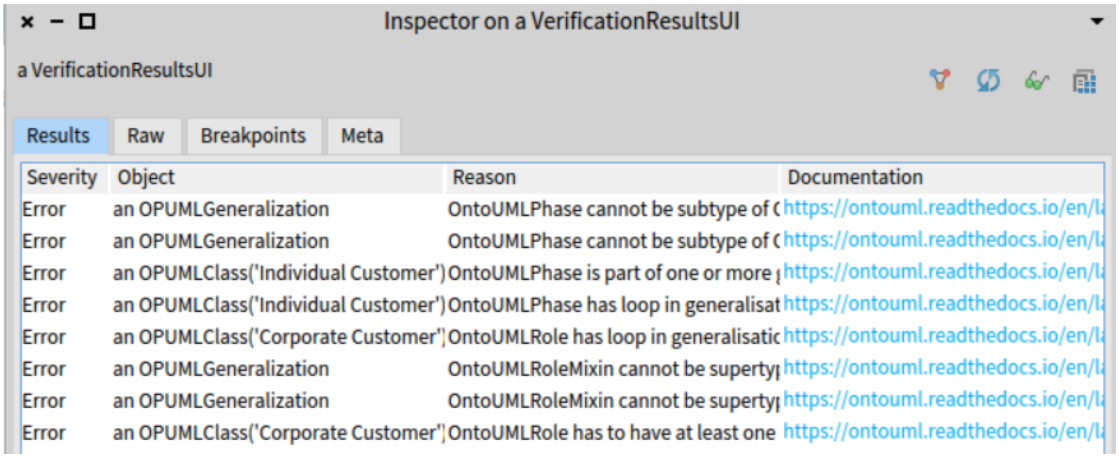
3.6.2 Analýza implementace verifikace OntoUML

Jak již bylo řečeno, inspirací pro validace BPMN diagramů může být rozšíření pro verifikaci OntoUML diagramů [40]. Toto rozšíření obsahuje mnoho funkcionalit, jako například automatické stahování verifikačních pravidel ze serveru dokumentace k jazyku OntoUML. Bohužel tuto funkcionalitu není možné využít vzhledem k povaze specifikace jazyka BPMN. Další funkcionalitou je detekce tzv. antipatternů, opět se jedná o specifikum OntoUML. Je ale možné se inspirovat základní architekturou rozšíření. V následujících odstavcích bude popsána zjednodušená architektura OntoUML verifikací, která je vyobrazena na obrázku 3.9.

Rozšíření je postaveno na hlavní třídě `VerificationController`. Tato třída zajišťuje celý proces verifikace/validace. Přímo z uživatelského rozhraní je volána metoda této třídy `verify:`, ta přebírá jako jediný argument kořenový prvek celého modelu. Jejím úkolem je vytvořit kolekce všech verifikací a všech uzlů diagramu, poté nad každým uzlem v cyklu spouští všechny verifikace voláním metody `verifySingleObject:withVerifications:`.

Jednotlivé verifikace tvoří samostatné třídy, jejichž společnou nadtřídou je `Verification`, která je pouze jako abstraktní. U každé verifikace je nejprve kontrolováno, zda s ní je možné verifikovat daný uzel, poté je spuštěna metoda `verifyObject:withModel:`. Tato metoda již provádí samotnou kontrolu pravidla. Pravidla je vhodné volit atomicky, tak aby kontrolovala pokud možno jen jeden uzel. Zároveň, aby jedna třída přesně odpovídala jednomu pravidlu pro snazší orientaci v kódu a správě chybových hlášek. Toto rozšíření také disponuje funkcionalitou pro vypínání a zapínání různých skupin pravidel. Pravidla jsou logicky rozdělena podle vztahů mezi prvky v OntoUML.

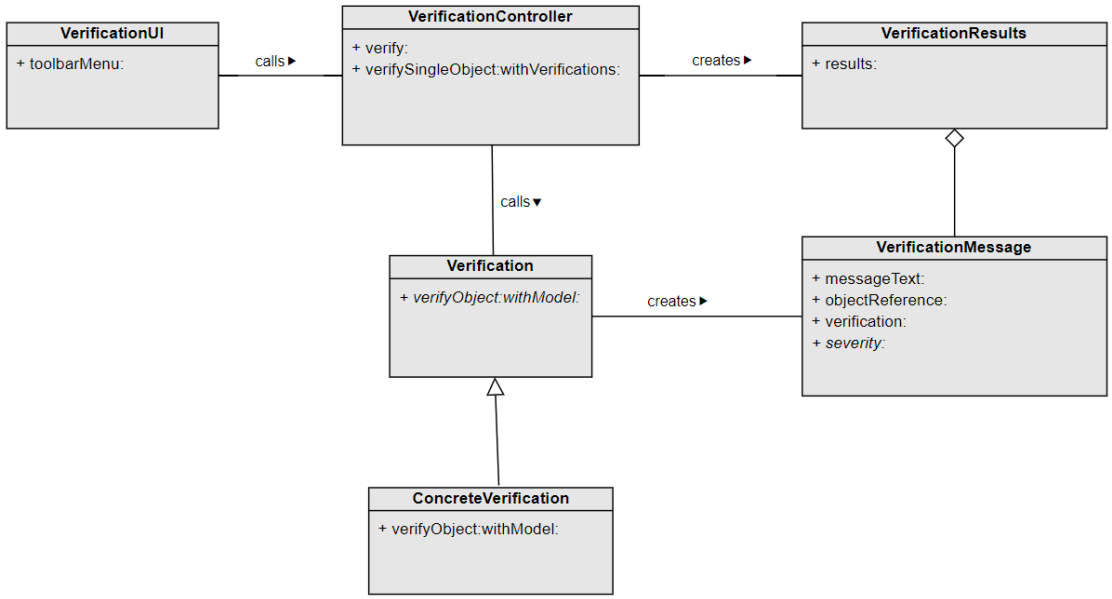
Výstupem verifikací jsou chybové hlášky, které jsou postupně sbírány do kolekce. Tato kolekce je zastoupena vlastní třídou `VerificationResults`, aby byla umožněna snadná manipulace s výsledky. Každá chybová hláška je potomkem třídy `VerificationMessage` a představuje stupeň závažnosti chyby nebo varování. Taková hláška obsahuje informace o objektu, ke kterému se vztahuje, o pravidlu, které bylo porušeno, závažnost, text chyby a také odkaz na podrobnější dokumentaci, kterou je možné zobrazit v externím prohlížeči.



Obrázek 3.8 Okno *Inspector* s výsledkem verifikace z balíčku OntoUML. [40] (Obrázek autora)

Celá verifikace se spouští z horního menu kliknutím na tlačítko „Verify model“. Po provedení

kontrol je zobrazena obrazovka uživatelského rozhraní. Jedná se o tzv. *Inspector* (viz obrázek 3.8), který umožňuje podrobněji procházet vybraná data a zjistit tak příčinu chyby. Výsledky přehledně jsou zobrazeny v tabulce pomocí rozšíření *Spec 2*. Hlavním prvkem zobrazeného okna je `SpTablePresenter`, tato třída umožňuje tvorbu jednoduché tabulky s vlastní definicí sloupců. Také je možné jednotlivé řádky tabulky vybírat a navázat na výběr nějakou akci. V tomto případě je při výběru řádku obarven prvek, ke kterému se vztahuje dané porušení pravidla, zelenou barvou přímo v diagramu.



Obrázek 3.9 Zjednodušená architektura tříd OntoUML verifikačního balíčku. [40] (Obrázek autora)

Kapitola 4
Praktická část

Praktická část této práce se zabývá návrhem validačních pravidel jazyka BPMN. Další kapitolou je návrh a implementace BPMN 2.0 modeláře a rozšíření pro validace BPMN 2.0 diagramů. Poslední kapitolou této části práce je dokumentace a testování.

4.1 Návrh validačních pravidel pro BPMN 2.0

V teoretické části práce bylo ukázáno, že jazyk BPMN nemá pevně stanovená pravidla modelování. Existuje dokumentace k nejnovějšímu standardu BPMN 2.0.2 [4], která je spíše obecným popisem jazyka. Nestanovuje pevně daná pravidla, jak modelovat. Některá pravidla je možné z textu vyčíst, jedná se zejména o základní konstrukty jazyka BPMN. Podrobněji jsou tato základní pravidla popsána v kapitole 3.4.2.

Existuje proto mnoho různých interpretací pravidel pro jazyk BPMN. Nejčastěji používanými a implementovanými jsou pravidla od Bruce Silvera 3.4.3. Jedná se o sadu pravidel, která se postupem času stala neformální konvencí pro správné modelování v BPMN. Jsou podle ní vedeny kurzy jazyka BPMN, je implementována v nejznámějších nástrojích a dokonce byly některé připomínky autora přidány přímo do specifikace jazyka. Tato analýza vychází zejména z těchto pravidel z knihy *BPMN Method and Style* [3]. Některá pravidla jsou převzata z jiných v předchozích kapitolách analyzovaných zdrojů.

4.1.1 Určení závažnosti validačního pravidla

Pravidla jsou rozdělena do dvou stupňů závažnosti, jako je tomu ve většině ostatních nástrojů nejen u jazyka BPMN. Také pan Ing. Marek Bělohoubek ve své bakalářské práci *Verifikace OntoUML modelů na platformě OpenPonk* [40] rozděluje závažnosti pravidel stejným způsobem.

Navíc je přidán třetí stupeň závažnosti, který neumožňuje diagram porušující toto pravidlo nakreslit. *OpenPonk* má implementovanou základní kontrolu diagramu, která byla částečně použita již v bakalářské práci [1]. Jde o jednoduchou kontrolu umístění elementu, každý element obsahuje seznam ostatních, které je možné na něj umístit, ať už jde o hrany nebo další vnořené uzly v případě bazénu a podprocesu.

Zaručeno je označení, kterým jsou dále označena tato pravidla, jejichž kontrola je zaručena již při modelování.

Error (chyba) je závažnost, která se vztahuje na zásadní porušení konstruktů jazyka BPMN. Tato pravidla jsou jasně stanovena ve specifikaci standardu jazyka BPMN. Porušení tohoto pravidla neumožňuje správnou interpretaci nakresleného diagramu a je nutné chybu opravit. Prvek obsahující danou chybu bude v diagramu jasně označen červenou barvou.

Warning (varování) má nižší stupeň závažnosti. Jedná se o pravidla, která buď nejsou ve specifikaci jasně řečená, nebo jsou stanovena v některém z jiných zdrojů jako „Best Practice“ modelování BPMN diagramů. Důvodem pro stanovení těchto pravidel je jasná a jednoznačná čitelnost diagramu. Dalšími důvody jsou dodržování stanovených nepsaných konvencí v BPMN modelování, označení kritických míst diagramu méně zkušeným uživatelům a nápověda pro zlepšení čitelnosti diagramu. Prvek obsahující danou chybu bude v diagramu jasně označen oranžovou barvou.

4.1.2 Provedení validace

U modelování BPMN diagramů nemá smysl provádět kontrolu správnosti za běhu. Jednou z technik modelování BPMN diagramů je postupování z vyššího úrovně abstrakce k podrobnějším detailům. Modely z jednotlivých úrovní abstrakce je vhodné si ukládat do zvláštních souborů. To umožňuje různé přístupy k procesu podle toho, kdo jej zrovna pozoruje. Při tomto postupu není model v každém kroku zcela validní. Například nejsou stanoveny typy aktivit, jejich popisky apod. Na vyšší úrovni detailu je model přesnější a modelovaný tak, aby odpovídal pravidlům jazyka BPMN. Z těchto důvodů není žádoucí osobu, která se soustředí na modelování, v každém kroku kontrolovat. [3]

Validace BPMN diagramů dávají smysl zejména ve finále návrhu diagramu, kdy je nutné doladit všechny potřebné detaily, případně odstranit neplatné konstrukty jazyka. V nástrojích podporující spouštění BPMN diagramů jsou často kontrolovány i spustitelné atributy, aby se předešlo chybám za běhu procesu. V případě této práce je kontrolována pouze sémantická stránka diagramu, čili zda je vizuálně nakreslená reprezentace validní.

Validace diagramu bude spuštěna stisknutím tlačítka *Validate Diagram* umístěném v horním menu nástroje pro modelování. Po provedené validaci bude otevřeno dialogové okno, ve Pharo nazývané jako *Inspector*, ve kterém budou přehledně v tabulce vypsaná všechna porušení pravidel. Tabulka bude obsahovat název prvku, který pravidlo porušil, závažnost a popis chyby/pravidla. Při kliknutí na porušení pravidla bude v diagramu vyznačen prvek, kterému porušení pravidla náleží. *Inspector* zároveň umožňuje nahlédnout do instance třídy modelu odpovídající porušení pravidlo, což usnadňuje řešení závažnějších problémů, které nejsou vidět při pohledu na diagram.

4.1.3 Validační pravidla

Nejprve jsou uvedena pravidla týkající se jednotlivých prvků, nakonec jsou uvedena složitější pravidla týkající se celého procesu. Narozdíl od pravidel v *BPMN Method and Style* [3], zde nejsou uvedena pravidla pro jednotlivé hrany, ale jsou zapracované pod dané uzly. Pro hrany budou uvedena pouze zaručená pravidla jejich vlastností. Validace bude probíhat tak, že se projdou všechny uzly. Pod uzlem můžou být teprve kontrolovány hrany, které jsou vzhledem k danému uzlu příchozí nebo odchozí. U každého pravidla je uveden kód chyby, název pravidla, závažnost a zdůvodnění proč je toto pravidlo vybráno jako vhodné. Podle kódu chyby bude možné dohledat podrobnější informace, chyba popisuje dvěma písmeny název prvku, kterého se týká, číselné označení určuje číslo pravidla.

4.1.3.1 Sequence Flow

SF01 (Zaručeno) *Sequence Flow* musí být připojena k flow uzlu (*Activity*, *Event* nebo *Gateway*) na obou koncích. – Pravidlo 1. BPMN Method and Style, základní pravidlo každého *Flowchart*.

SF02 (Zaručeno) Sequence Flow nesmí překročit hranice procesu ani podprocesu – Vyplývá ze základní definice sekvenčního toku. V OpenPonku zaručeno kontrolou, že zdroj i cíl toku musí mít stejného rodiče.

4.1.3.2 Message Flow

MF01 (Zaručeno) *Message Flow* nesmí spojovat prvky ve stejném procesu. – Pravidlo 9. BPMN Method and Style.

4.1.3.3 All Nodes

AN01 (Error) Všechny flow uzly mimo *Start Event* musí mít příchozí hranu. – Pravidlo 2. BPMN Method and Style, které dovoluje navíc diagramy bez *Start Event*, stejně jako specifikace jazyka. Například bpmnlint obsahuje v pravidlech, že proces musí mít *Start* i *End Event*.

AN02 (Error) Všechny flow uzly mimo *End Event* musí mít odchozí hranu. – Stejné jako předchozí bod.

AN03 (Error) Defaultní odchozí hranu smí mít pouze *Exclusive, Inclusive* nebo *Complex Gateway* a *Activity*. Mohou mít nejvýše jednu defaultní odchozí hranu. – Pravidlo 8. BPMN Method and Style. Doplněné možností použití dle dokumentace. Z hlediska významu defaultní hrany nedává smysl použití více těchto hran, proto závažnost chyby.

AN04 (Warning) Odchozí *Message Flow* by měla být popsána názvem zprávy – Pravidlo 57. BPMN Method and Style.

AN05 (Warning) Všechny uzly (mimo Start Event bez typu, Parallel Gateway) by měly obsahovat popisek. – Pravidla 40., 42., 43., 46., 49. BPMN Method and Style.

AN06 (Error) Odchozí hrana nesmí být podmíněná, pokud je jedinou odchozí hranou uzlu. – Pravidlo 6. BPMN Method and Style.

4.1.3.4 Start Event

SE01 (Zaručeno) *Start Event* nesmí mít příchozí *Sequence Flow*. – Pravidlo 13. BPMN Method and Style, z definice *Start Event*.

SE02 (Zaručeno) *Start Event* nesmí mít odchozí *Message Flow*. – Pravidlo 14. BPMN Method and Style, z definice *Start Event*.

SE03 (Error) *Start Event* typu *Message* musí mít příchozí *Message Flow*.

SE04 (Error) *Start Event* s příchozí *Message Flow* musí být typu *Message* nebo *Multiple*. – Pravidlo 15. BPMN Method and Style, závažnost chyby, protože by jinak nebylo jak zprávu zpracovat.

SE05 (Error) *Start Event* nesmí být typu *Error*, s výjimkou použití v podprocesu. – Pravidlo 16. BPMN Method and Style, závažnost chyby, jiné použití nedává smysl.

SE06 (Error) *Start Event* v podprocesu nesmí mít typ, pokud není spuštěn na základě nějaké události tzv. *Event*. – Pravidlo 17. BPMN Method and Style. Toto pravidlo bude těžké kontrolovat, protože je nutné znát celý kontext procesu.

4.1.3.5 End Event

EE01 (Zaručeno) *End Event* nesmí mít odchozí *Sequence Flow*.– Pravidlo 18. BPMN Method and Style.

EE02 (Zaručeno) *End Event* nesmí mít příchozí *Message Flow*. – Pravidlo 19. BPMN Method and Style.

EE03 (Error) *End Event* typu *Message* musí mít odchozí *Message Flow*.

EE04 (Error) *End Event* s odchozí *Message Flow* musí být typu *Message* nebo *Multiple*. – Pravidlo 20. BPMN Method and Style, závažnost chyby, protože by jinak nebylo jak zprávu vytvořit.

4.1.3.6 Boundary Event

BE01 (Zaručeno) *Boundary Event* nesmí mít příchozí *Sequence Flow*. – Pravidlo 23. BPMN Method and Style.

BE02 (Error) *Boundary Event* musí mít právě jednu odchozí *Sequence Flow*. (S výjimkou *Compensation Event*.) – Pravidlo 21. BPMN Method and Style.

BE03 (Warning) *Error Boundary Event* v podprocesu musí obsahovat odpovídající *Error Throw Event*. – Pravidlo 24. BPMN Method and Style. Stanoveno pouze jako varování, protože proces může standardně proběhnout i pouze pokud neexistuje párový *End Event*.

BE04 (Warning) *Escalation Boundary Event* v podprocesu musí obsahovat odpovídající *Escalation Throw Event*. – Pravidlo 26. BPMN Method and Style. Stanoveno pouze jako varování, protože proces může standardně proběhnout i pouze pokud neexisutje párový *Intermediate/End Event*.

BE05 (Error) *Error Boundary Event* nesmí být *Non-Interrupting* – – Pravidlo 25. BPMN Method and Style.

4.1.3.7 Intermediate Event

IE01 (Error) *Intermediate Event* typu *Message Receive* musí mít příchozí *Message Flow*.

IE02 (Error) *Intermediate Event* typu *Message Send* musí mít odchozí *Message Flow*.

IE03 (Error) *Intermediate Event* s příchozí *Message Flow* musí být typu *Message Receive (Catch)* nebo *Multiple*. – Pravidlo 27. BPMN Method and Style, závažnost chyby, protože by jinak nebylo jak zprávu zpracovat.

IE04 (Error) *Intermediate Event* s odchozí *Message Flow* musí být typu *Message Send (Throw)* nebo *Multiple*. – Pravidlo 28. BPMN Method and Style, závažnost chyby, protože by jinak nebylo jak zprávu zpracovat.

4.1.3.8 Gateway

GW01 (Zaručeno) *Gateway* nesmí mít příchozí ani odchozí *Message Flow*. – Pravidla 33. a 34. BPMN Method and Style.

GW02 (Error) Rozdělující *Gateway* musí mít více než jednu odchozí *Sequence Flow*. Spojující *Gateway* musí mít více než jednu příchozí *Sequence Flow*. – Pravidlo 35. BPMN Method and Style spojené s pravidlem 16. bpmnlint.

GW03 (Warning) *Gateway* musí mít buď více než jednu odchozí *Sequence Flow*, nebo více než jednu příchozí *Sequence Flow*. Neměla by být zároveň spojující i rozdělující. – Pravidlo 10. bpmnlint. Závažnost nastavena pouze jako varování, není obsaženo ve specifikaci jazyka.

GW04 (Error) *Event Gateway* může ve svých větvích obsahovat pouze *Catching Intermediate Event* nebo *Receive Task*. – Pravidlo 36. BPMN Method and Style.

GW05 (Error) *Sequence Flow* vedoucí z *Parallel/Event Gateway* nesmí být podmíněná. – Nesmí být typu *Conditional*, ani obsahovat popisek. Pravidlo 7. BPMN Method and Style.

GW06 (Error) U *Sequence Flow* vedoucí z *Gateway* není podmíněný atribut (diamant na začátku podmíněné hrany) zobrazovaný. – Pravidlo 50. BPMN Method and Style

GW07 (Warning) *Sequence Flow* vedoucí z *Gateway* (jiného typu než *Parallel* a *Event* a mimo defaultní hrany) by měly být popsány. – Pravidlo 51. BPMN Method and Style, jedná se o popisky, závažnost varování.

Tabulka 4.1 Porovnání s pravidly BPMN Method and Style – Číslo pravidla BPMN Method and Style; Číslo pravidla v této práce; Informace, zda bylo pravidlo implementováno; Poznámka ke způsobu implementace

Č. p. BPMN M&S	Č. p. v této práci	Implemenotváno?	Poznámka
1.	SF01	Ano	
2.	AN01	Ano	
3.	AN02	Ano	
4.	SF02	Ano	
5.	SF02	Ano	
6.	AN06	Ano	
7.	GW05	Ano	
8.	AN03	Ano	
9.	MF01	Ano	
10.	SE04, IE03, AC03, PR05	Ano	
11.	EE04, IE4, AC04, PR05	Ano	
12.	–	Ano	Vlastnost nástroje
13.	SE01	Ano	
14.	SE02	Ano	
15.	SE03, SE04	Ano	
16.	SE05	Ano	
17.	SE06		
18.	EE01	Ano	
19.	EE02	Ano	
20.	EE03, EE04	Ano	
21.	BE02	Ano	
22.	–	Ano	Vlastnost nástroje
23.	BE01	Ano	Zaručeno
24.	BE03	Ano	
25.	BE05	Ano	
26.	BE04	Ano	
27.	IE01, IE03	Ano	
28.	IE02, IE04	Ano	
29.	–	Ano	Vlastnost nástroje
30.	–	Ano	Vlastnost nástroje
31.	AN02	Ano	
32.	AN01	Ano	
33.	GW01	Ano	
34.	GW01	Ano	
35.	GW02	Ano	
36.	GW04	Ano	
37.	PR02	Ano	
38.	PR03	Ano	
39.	PR01	Ano	Bez podprocesu

4.1.3.9 Activity

AC01 (Error) *Activity* typu *Message Receive* musí mít příchozí *Message Flow*. – Pravidlo 15. bpmn-lint. BPMN Method and Style pravidla aktivit nezmiňuje, ale jejich použití je logické a vyplývá

z pravidel o *Message Flow*.

AC02 (Error) *Activity* typu *Message Send* musí mít odchozí *Message Flow*.

AC03 (Error) *Activity* s příchozí *Message Flow* musí být typu *Message Receive*.

AC04 (Error) *Activity* s odchozí *Message Flow* musí být typu *Message Send*.

AC05 (Warning) *Activity* by neměla sloužit ke spojování, či rozdělování toků. – Pravidlo 4. bpmnlint. Závažnost varování, protože použití konstruktu zmiňuje i dokumentace, ale znepřehledňuje čitelnost diagramu.

4.1.3.10 Process (Pool)

PR01 (Zaručeno) *Pool* v sobě nesmí obsahovat jiný *Pool*. – Pravidlo 39. BPMN Method and Style, zjednodušené.

PR02 (Error) Proces musí obsahovat alespoň jednu *Activity*. – Pravidlo 37. BPMN Method and Style.

PR03 (Error) Prvky nejvýše jednoho procesu mohou být obsaženy v jednom *Pool*. – Pravidlo 38. BPMN Method and Style.

PR04 (Error) Pokud diagram obsahuje jen jednoho participanta, nemusí být zakreslen. Pokud jich je víc, musí být všechny flow prvky zakresleny uvnitř bazénů. – Z BPMN specifikace.

PR05 (Error) *Pool* může být zdrojem *Message Flow* pouze jako *Black-Box*. – Pravidlo 10. BPMN Method and Style, rozděleno podle prvků.

PR06 (Error) *Pool* může být cílem *Message Flow* pouze jako *Black-Box*. – Pravidlo 11. BPMN Method and Style, rozděleno podle prvků.

PR07 (Warning) Proces by měl obsahovat *Start* a *End Event*. – Pravidlo 2. bpmnlint.

4.1.4 Porovnání validačních pravidel s BPMN Method and Style

V tabulce 4.1 je shrnuté namapování pravidel na pravidla BPMN Method and Style [3]. Stylová pravidla do tabulky nebyla zahrnuta, protože netvoří hlavní účel validace BPMN diagramů. Tabulka již obsahuje i sloupec, zda bylo pravidlo naimplementováno.

4.2 Návrh a implementace

Praktická část práce navazuje na implementaci z bakalářské práce [1]. Výsledný kód této práce obsahoval dva balíčky: *OpenPonk-BPMN* s implementací samotného modeláře a druhý balíček *OpenPonk-BPMN-XML* určený pro export a import diagramů do jiných nástrojů. Balíček pro export zůstane zachován v nezměněné funkční podobě pro export a import základních diagramů.

Balíček OpenPonk-BPMN musí být přepracován kvůli změnám v architektuře nástroje. Controllery jsou v nové verzi striktně odděleny od vrstvy View, to má za následek přepracování většiny metod v těchto dvou vrstvách. Následně bude rozšířen o další prvky BPMN 2.0 notace, tak aby nástroj obsahoval pokud možno, co nejvíce prvků a konstruktů z notace.

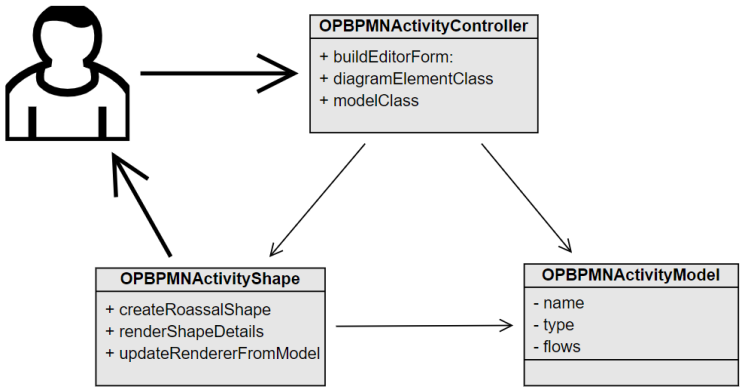
Také bude implementován třetí balíček *OpenPonk-BPMN-Validation*, který bude poskytovat validaci BPMN diagramů. Tento balíček je inspirován podobným rozšířením pro verifikaci OntoUML diagramů, které vypracoval Ing. Marek Bělohoubek ve své bakalářské práci [40]. Proces

kontroly diagramu je v této práci nazván validací, jelikož se jedná o vlastní kontrolu finálního výsledku diagramu. Pravidla nejsou nijak zanesena ve specifikaci jazyka, proto se nemůže jednat o verifikaci. Verifikace poskytuje potvrzení správnosti na základě specifikací stanovených pravidel.

4.2.1 Návrh a implementace modelovacího nástroje pro BPMN

Základní balíček BPMN modelovacího pluginu, OpenPonk-BPMN, pro platformu OpenPonk byl vytvořen v bakalářské práci. [1] Celkem bylo vytvořeno 25 různých prvků jazyka BPMN, tím byla pokryta většina tzv. Level 1 palety jazyka BPMN 1.0. Cílem této práce je pokrýt pokud možno celou paletu jazyka BPMN 2.0.

Všechny modelovací doplňky platformy OpenPonk jsou postaveny třívrstvé architektuře návrhového vzoru Model-View-Controller. Tato architektura umožňuje snadnou správu, kontrolu a rozšiřovatelnost. V původní bakalářské práci však nebyly striktně odděleny vrstvy View a Controller. Controller obsahoval údaje, jak daný prvek vykreslit, také se staral o propagaci změn. S přechodem OpenPonku na nový vykreslovací engine Roassal 3 bylo rozhodnuto také o změně v architektuře OpenPonku, a to striktní oddělení vrstev View a Controller. Spolupráci vrstev lze vidět na kontrétním příkladu pro prvek *Activity* na obrázku 4.1. Ke změně došlo během tvorby této diplomové práce, proto již byla zapracována.



Obrázek 4.1 Ukázka spolupráce konkrétních tříd včetně základních metod pro prvek *Activity* založené na architektuře MVC. (Obrázek autora)

Tato diplomová práce pracuje se zatím neoficiální rozpracovanou verzí platformy OpenPonk, s pravděpodobným názvem 4.0 (aktuálně poslední oficiální verze je verze 3.0.3 z roku 2021). Verze bude dostupná taktéž pro novou zatím neoficiálně vydanou verzi Pharo 11.

Implementace modeláře by kromě změn v architektuře měla obsahovat také mnoho dalších prvků notace BPMN 2.0:

- Bazén včetně drah.
- Expandovaný podproces včetně *Boundary Events*.
- Zabalený podproces včetně *Boundary Events*.
- Další typy aktivit včetně markerů a *Call Activity*.
- Další typy událostí včetně *Non-Interrupting*.

- Další typy bran.
- Defaultní a podmínkové *Sequence Flow*.
- Datové objekty včetně hran.
- Přidání možnosti ohraničení a poznámky z implementace jádra OpenPonku.

Po implemenentaci všech výše uvedených bodů by měla být pokryta celá paleta BPMN 2.0, kromě konverzací a choreografie, které se však v běžném procesním modelování nepoužívají. Navíc nejsou zatím ani vyučovány na předmětech softwarového inženýrství, kde se jazyk BPMN probírá. Nástroj bude pro studenty těchto předmětů k dispozici.

4.2.1.1 Model

Již existující třídy modelu není nutné moc měnit. Zůstává také dědičnost na jádro OpenPonku. Pro uzly byla navíc přidána třída `OPBPMNFlowNodeModel`, která tvoří nadtřídu pro *Activity*, *Events*, *Gateway* a *Sub-Process*. Neobsahuje však žádnou implementaci, jedná se o oddělení různých funkcí uzlů.

`OPBPMNPoolModel` nově obsahuje proměnnou s počtem drah a seřazenou kolekci názvů. K operacím nad kolekcí kromě getteru a setteru obsahuje i dvě specifické metody:

`laneNamesAt:put:` vloží název dráhy na specifickou pozici v kolekci. Parametry jsou číslo a text obsahující nový název.

`laneNamesFitToLanesCount` při změně počtu drah je třeba změnit také velikost kolekce názvů. Pokud je drah méně než dvě, je kolekce vyprázněna. V opačném případě je upravena velikost tak, aby počet názvů odpovídal počtu drah.

`OPBPMNCollapsedSubProcessModel` je nově vzniklou metodou, oba typy podprocesu byly odděleny, protože každý má jiné vlastnosti. Tento složený podproces může obsahovat marker nebo sloužit jako *Call Activity*, obě vlastnosti jsou zaneseny v modelu. Stejné vlastnosti byly přidány také do `OPBPMNActivityModel`.

`OPBPMNSubProcessModel` představuje expandovaný podproces. Může proto v sobě obsahovat jiné *Flow uzly*. Z tohoto důvodu obsahuje třída, podobně jako je tomu u bazénu, metody pro přístup k vnitřním uzlům a jejich odstranění: `nodes`, `removeAllNodes, \verb`removeNode—.

`OPBPMNStartEventModel` byla přidána možnost oddělení, zda je přerušující, či nepřerušující. Proto nově obsahuje boolean proměnnou `isNonInterrupting` a k ní patřičný getter a setter.

`OPBPMNBoundaryEventModel` je specifickou událostí, která je vázána na konkrétní prvek diagramu, proto oproti základnímu uzlu obsahuje navíc informaci, ke kterému uzlu náleží – subscribeDiagram. Také obsahuje možnost `isNonInterrupting`. *Boundary Event* neexistuje bez typu, proto byl jako výchozí typ zvolen *Error*.

`OPBPMNDataObjectModel` je vytvořen jako výchozí typ *DataObject*, dalšími typy jsou *DataInput*, *DataOutput* a *DataStore*.

Nadtřídy hran zůstaly beze změn. `OPBPMNFlowModel` dostala navíc typ, který je ve výchozím stavu nastaven na *Normal*, dalšími možnostmi jsou *Default* a *Conditional*. Přidanou třídou je `OPBPMNDataFlowModel`, která neobsahuje žádné speciální vlastnosti.

4.2.1.2 Controller

Tato vrstva prošla zásadní změnou. Všechny metody, které se týkaly vykreslování prvků byly přesunuty do vrstvy view, zde nazývané jako *DiagramElements*. Bylo odebráno několik metod: `renderFigureIn:`, `refreshFigure`, `createRoassalShape`, `createFigure` a `removeFigure`. Místo nich byla přidána pouze jedna metoda: `diagramElementClass`, která odkazuje na nově

vzniklé třídy view. Ostatní vlastnosti jako vytváření formuláře prvku a určení, zda může být prvek zdrojem/cílem jiného prvku, zůstávají nezměněné.

`OPBPMNPoolController` umožňuje být cílem pro ostatní controllery *Flow* uzlů a zdrojem pouze pro *Message Flow*. Formulář bazénu byl upraven tak, aby bylo možné pomocí něj přidávat jendotlivé dráhy. Po vytvoření obsahuje formulář informaci o počtu drah (výchozí počet: 1), pokud je počet drah zvýšen, zakreslí se do diagramu a ve formuláři se vytvoří tolik textových polí, kolik drah bylo vytvořeno. Formulář se obnoví po kliknutí zpět na bazén. Tak je umožněno pojmenovat každou dráhu jinak.

`OPBPMNCollapsedSubProcessController` také definuje vlastní formulář. Jako první obsahuje *Select box*, ve kterém je možné vybrat *Marker* podprocesu (sekvenční, paralelní, apod.). Poté je možné zaškrtnout *Check box*, zda se jedná o *Call Activity*. Posledním prvkem formuláře je tlačítko pro přidání *Boundary Event*, který se vytvoří do pravého dolního rohu podprocesu a není možné s ním pohybovat. Ve výsledné implementaci je možné přidat pouze jeden *Boundary Event*.

`OPBPMNSubProcessController` neboli expandovaný podproces umožňuje vložení prvků, controller proto obsahuje metody pro správu vnitřních elementů a může být cílem všech *Flow* uzlů. Formulář umožňuje stejné nastavení jako předchozí podproces.

`OPBPMNBoundaryEventController` také obsahuje vlastní controller, přestože jej nelze vytvořit samostatně. Jedná se o plnohodnotný element, se kterým není možné samostatně pohybovat. Ale je možné jej označit a vést z něj hranu. Proto je možné také nastavovat vlastnosti prvku ve formuláři, který zde obsahuje možnost nastavení typu a výběr, zda je událost přerušující, či nepřerušující.

`OPBPMNDataObjectController` obsahuje také formulář s možností změny typu. Může být zdrojem a cílem pouze pro *Data Flow*.

4.2.1.3 View

O rozvržení obrazovek se starají třídy jádra OpenPonku, které používají vizualizační engine Spec 2. Typ layoutu je definován ve třídě OPBPMNLayouter. Jednotlivé prvky jsou tvořeny z jednoduchých tvarů, které poskytuje rozšíření Roassal 3. Třídy pro jednotlivé prvky jsou seskupeny pod označením *DiagramElements*.

OpenPonk definuje základní vlastnosti všech prvků ve třídě `OPDiagramElement`. Prvek je definován jako tvar z rozšíření Roassal 3 – `roassalShape`. Může vlastnit další tvary, které tvoří kolekci – `ownedElements`. Tyto vlastněné tvary přebírají vlastnosti hlavního tvaru a jsou s ním přesouvány a mazány. `OPDiagramElement` obsahuje také další informace jako je třída modelu, rodičovský prvek, zdrojové a cílové hrany. Vyjmenované vlastnosti pokrývá sada metod:

`renderIn:` slouží k vytvoření elementu. Volá postupně metody k vytvoření základního tvaru, poté detailů a nakonec i vlastněných prvků. Element je ještě aktualizován pomocí informací z modelu a následně je vrácen celý element jako tvar Roassalu.

`createRoassalShape` slouží k vytvoření základního tvaru, který vrací v návratové hodnotě. Nutno definovat v podtřídě.

`renderShapeDetails` vytváří dodatečné tvary jako jsou ikony a popisky, tyto tvary jsou vázané na samotný `roassalShape`. Nutno definovat v podtřídě.

`updateFromRender` slouží k aktualizaci všech vlastněných prvků.

`updateFromModel` slouží k aktualizaci informací z modelu.

`updateFromSelf` slouží k aktualizaci sebe sama a vlastněných prvků.

Uzly BPMN rozšíření pokrývá třída `OPBPMNNodeShape`, která je podtřídou právě zmíněné `OPDiagramElement`. Obsahuje navíc proměnné pro ikonu a pro barvu obrysu prvku. Nějakou vnitřní ikonu obsahují všechny prvky kromě bazénu. Barva obrysu se používá pro zvýraznění validovaných prvků.

`renderShapeDetails` vytváří dodatečné tvary elementu. V této nadřídě je navíc definován popisek elementu jako `RSLabeled`, text je nastaven na černou barvu. Také je doprostřed tvaru přidána ikona, která je vytvořena pomocí `RSBitmap`. Pro celý element takto definovaný platí, že je možné ho v diagramu označit (`OPRSSelectable`) a pomocí stisknutí myši přesunout (`OPRSSelectionDraggable`). Implementace metody se nachází v ukázce kódu 4.1.

Všechny ikony, které se v rozšíření nacházejí, jsou definované ve formátu *base64*, jedná se o textový formát pomocí kterého je možné zakódovat bitmapové obrázky. Ikony byly do požadovaného formátu převedeny pomocí dekodéru [41]. Všechny použité ikony jsou z oficiálního fontu pro nástroj BPMN.io [42], font je zveřejněn pod Open Source licencí *OFL*. K použití již definovaného známého fontu bylo mnoho důvodů, zejména nutnost neporušit specifikaci jazyka, která vzhled prvků poměrně striktně definuje. Z uživatelského hlediska se jedná o příjemnou záležitost, protože je tento font použit ve většině populárních BPMN modelářů.

Výpis kódu 4.1 Ukázka implementace metody renderShapeDetails ve třídě OPBPMNNodeShape

```
renderShapeDetails
    | label selectable |
    label := RSLabeled new.
    label location below.
    label shapeBuilder labelShape color: Color black.
    roassalShape addInteraction: label.
    icon := RSBitmap new
                form: self typeIconForModel;
                yourself.
    self canvas add: icon.
    RSLocation new stick: icon on: self roassalShape.
    self roassalShape when: RSShapeRemovedEvent do: [icon remove].
    selectable := OPRSSelectable new.
    selectable highlightBorderColor: Color blue.
    roassalShape @ selectable.
    roassalShape @ OPRSSelectionDraggable
```

`createRoassalShape` je nutno definovat v podtřídě.

`setupHighlight:` slouží ke změně barvy obrysu prvku. Při označení prvku je pro zvýraznění použita modrá barva.

`typeIconForModel` vloží obrázek ikony podle typu, který je převzat z modelu.

`updateRendererFromModel` aktualizuje název elementu a jeho ikonu přímo z dat modelu. Nakonec je iniciováno znovunačtení prvku.

`updateRendererFromSelf` aktualizuje barvu prvku a barvu ohraničení podle aktuální uložených hodnot.

Konkrétní prvky notace BPMN vycházejí z této implementace. Dále budou zmíněny pouze metody, které se odlišují.

`OPBPMNGatewayShape` implementuje metodu `createRoassalShape`, ve které definuje vlastní tvar jako `RSPolygon` určený čtyřmi body, tak aby vznikl kosočtverec. Druhá metoda pro detail tvaru `renderShapeDetails` pouze mění barvu tvaru na světle modrou.

`OPBPMNStartEventShape` obsahuje metodu `createBorder`, která vytváří různá ohraničení prvku podle nastavení, zda se jedná o přerušující, či nepřerušující prvek. Je použit samostatný `RSBorder`, u kterého je možné nastavit obrys jako pole přerušovaných čar. Tvar prvku je obyčejný kruh z Roassal 3 rozšíření.

`OPBPMNIntermediateEventShape` definuje v detailech prvku navíc druhý vnitřní kruh, který společně s vnějším kruhem vytváří dvojitou čáru.

`OPBPMNEndEventShape` obsahuje také `RSBorder`, u kterého je nastavena větší tloušťka čáry.

`OPBPMNBoundaryEventShape` je definován stejně jako ostatní události s tím rozdílem, že není možné jej přesouvat, tj. nemá povolenou možnost `OPRSSelectionDraggable`.

`OPBPMNActivityShape` má základní tvar vytvořený jako `RSBox`. Třída také obsahuje metodu `createBorder` pro nastavení tloušťky ohraničení pro zobrazení jako *Call Activity*. Ikona typu není umístěna uprostřed ale v pravém horním rohu.

`OPBPMNCollapsedSubProcessShape` obsahuje podobně jako aktivita nastavení pro *Call Activity*. Vytvoření detailu prvku zde bylo nutné implementovat odlišně, jelikož je nutné do prvku umístit dvě ikony. První základní ikona je zobrazena vždy, a to dole uprostřed – metoda pro vložení ikony se jmenuje `typeBasicIcon`.

`OPBPMNSubProcessShape` obsahuje pouze tvar obdélníku s popiskem. Zajímavé jsou detaily elementu, s posunutím podprocesu musí být posunuty také všechny prvky v něm obsažené. Proto obsahuje vlastnost `OPRSSelectionDraggableBorder`, která zachycuje všechny prvky, které jsou umístěné na podprocesu. Podproces je také možné zvětšovat díky vlastnosti `OPRSResizable`.

`OPBPMNPoolShape` je nejsložitějším objektem celé implementace, protože přímo obsahuje také dráhy. Základním tvarem bazénu je `RSComposite`, jedná se o neviditelné obdélníkové ohraničení prvku, které zajišťuje, že všechny prvky v něm přidané mají stejné vlastnosti. To umožňuje vytvořit komplexní prvek, který je možné zvětšovat a zmenšovat, ale zároveň se přizpůsobují změně také vnitřní elementy a celý tvar se chová jako jeden. Bazén je složen ze dvou `RSBox`, které jsou umístěné vedle sebe. Metoda `addLanes:` vytváří dráhy. Každá čára oddělující dvě dráhy je složena ze tří tvarů. Jedná se o dva malé čtverce, které jsou spojeny hranou `RSLine`. Tato hrana má vlastnost přizpůsobení velikosti mezi dvěma body, které se mohou libovolně pohybovat, tím je zaručena možnost zvětšování objektu včetně roztažení hran. Dráhy rozdělují bazén na stejně velké části, v této verzi není možné s nimi samostatně pohybovat. Každá dráha obsahuje také popisek, který je vytvořen pomocí `RSLabel`. Text popisků je aktualizován z modelu.

`OPBPMNDataObjectShape` je podtřídou `OPShape`, je oddělen od ostatních uzlů, jelikož celý tvar prvku tvoří ikona. Ikona je obalena tvarem `RSComposite` aby bylo možné ikonu ohraničit a měnit barvu tohoto ohraničení.

Hrany jsou tvořeny z třídy jádra `OPShape`. Tato třída definuje vlastnosti základní hrany. Jedná se o definici zdroje a cíle hrany a také bodů zlomu, tzv. *waypoints*. Jedná se o novinku v této verzi. Hranu je možné v polovině zlomit pod libovolným úhlem, poté opět ve čtvrtině, apod. Popisek hrany je definován ve vlastní třídě `OPBPMNEdgeLabel`, obsahuje informace v jaké vzdálenosti od hrany se má nacházet. Hrana pak takto definovaný popisek obsahuje jako vlastněný element.

`OPBPMNFlowShape` definuje tvar pro *Sequence Flow*. Zbylé dvě hrany také vychází z této implementace, proto nebudou uvedeny.

`renderBasicShape` definuje základní tvar, který je tvořen z `OPRSPolyline`, která definuje jak je připojená ke zdroji a cíli, jaké má označení na začátku a na konci a také jestli je možné ji označit myší.

`headMarker` určuje tvar na konci hrany, u cílového prvku. Zde se jedná o šipku, která je definována čtyřmi body.

`tailMarker` určuje tvar na začátku hrany. Zde se střídají tři různé tvary podle typu hrany – žádný, šikmá čára a diamant.

`createOwnedElementsWithoutController` vytváří popisek hrany tak, že k němu není třeba controller a model.

`updateRendererFromModel` aktualizuje typ hrany z modelu.

4.2.2 Návrh a implementace validace BPMN

Validace BPMN diagramů se nachází ve zvláštním balíčku *OpenPonk-BPMN-Validation*, tím je pro uživatele zaručena volitelná možnost použití, pro vývojáře je výhodou přehlednost kódu a snazší udržitelnost. Balíček naopak požaduje instalaci základního OpenPonk-Core a rozšíření OpenPonk-BPMN. Implementace vychází ze základní analýzy práce Ing. Marka Bělohoubka [40], klíčové třídy jsou zachovány, pouze bylo nutné navíc přidat nějaké pomocné třídy a metody. Balíček pro validaci je plně oddělený a je možné jej neinstalovat, pokud uživatel nechce validace používat. Třídy v balíčku jsou rozdělené pod jednotlivé tagy (štítky) pro větší přehlednost. Podrobný diagram tříd a všech metod (kromě getterů a setterů) je na obrázku 4.2.

Validaci je možné spustit pomocí tlačítka „Validate model“, které se nachází v horním menu diagramu v záložce „Diagram“, sekci „Validations“. Mimo spuštění validace je zde také k dispozici tlačítko pro odebrání zvýraznění „Remove highlights“, které umožňuje ponechat otevřené okno s validacemi, ale odebrat zvýraznění, aby bylo možné pohodlně modelovat.

Základní třída `OPBPMNValidationController` plní úlohu moderátora celé validace. Spouští validace nad každým uzlem procesu a vrací výsledek uživatelskému rozhraní.

`OPBPMNValidationController` obsahuje pět metod:

`validate:` metoda je volána přímo z uživatelského rozhraní. Obsahuje jeden argument, kterým je kořenový prvek diagramu. Metoda nejprve vytvoří kolekci všech uzlů v diagramu, poté také kolekci všech validačních pravidel. Pro uložení výsledků je vytvořena proměnná *results*, která je instancí třídy `OPBPMNValidationResults`. Provedení samotných validací je zahájeno v metodě `validateProcess:`, ve které jsou provedeny validace celého diagramu. Nakonec je nad každým uzlem spuštěna metoda `validateSingleObject:...`, výjimku tvoří bazén a podproces, jejichž vnitřní prvky jsou dodatečně také přidány do kolekce uzlů pro validace, tak aby nebyl žádný prvek vynechán. Návratovou hodnotou celé metody je proměnná *results*.

`validateProcess:` slouží ke zkontrolování diagramu jako celku. Tedy obsahuje validační pravidla, která se nad celým diagramem kontrolují pouze jednou, nikoliv nad každým uzlem, jako ostatní. Těmito pravidly jsou PR02, PR03, PR04 a PR07. V metodě je zkontrolována kolekce všech uzlů a spočteny metriky nutné k posouzení těchto pravidel. Počítané metriky jsou: počet aktivit, počet aktivit mimo bazény, počet bazénů a počet *Start Events*. Na základě počtu aktivit je posouzeno pravidlo PR02. Pokud existují aktivity mimo bazény, ale zároveň i v nich, či ve více bazénech najednou, je porušeno pravidlo PR03 nebo PR04. Konečně pravidlo PR07 je porušeno, pokud proces neobsahuje právě jeden *Start Event*. Vstupním parametrem je kořenový prvek diagramu `OPBPMNModel`, návratovou hodnotou je proměnná *results* stejně jako v předchozí metodě.

`validateSingleObject:withValidations:withModel:withResults:` umí nad daným uzlem provádět všechny validace přes kolekci pravidel. Aby bylo možné kontrolovat prvek v kontextu celého modelu, je předána také reference na kořenový prvek diagramu. Pro spuštění každého pravidla je volána metoda `validate:withModel:` z abstraktní třídy `OPBPMNValidation`. Pro výstup je opět použita instance *results*.

`getValidations` je metoda, která vrací množinu všech validací. Jedná se o všechny třídy pod tagem *Validation-Classes*, není tak nutné nově vytvořené pravidlo přidávat do žádné kolekce, ale je generována automaticky ze všech obsažených tříd.

`getValidateObjects:` vrací množinu všech uzlů prvku, který je dodán jako vstupní parametr. Pokud jsou prvky vnořeny v jiném, je nutné metodu nad tímto prvkem zavolat znovu, tak je tomu u bazénu a podprocesu.

`OPBPMNValidation` zastřešuje jednotlivá pravidla, jejím úkolem je poskytnout jednoduchý způsob, jak spouštět jednotlivá pravidla. K pravidlům nejsou ihned vytvářeny instance objektů,

proto je první volanou metodou třídní metoda (Class side) `validate:withModel:`. Každé pravidlo navíc obsahuje kontrolu, zda s ním je možné daný prvek validovat.

Nejdůležitější metody třídy `OPBPMNValidation`:

`canValidate:` (Class side) metoda kontrolující zda dané pravidlo může kontrolovaný prvek validovat. Návratovou hodnotou je boolean true/false.

`validateClass` (Class side) metoda, která má jako návratovou hodnotu název třídy/nadtřídy modelu, kterou je schopné dané pravidlo validovat. Implementované v každém pravidlu.

`validate:withModel:` (Class side) obsahuje na vstupu dva parametry. Prvním je objekt k validování, neboli třída modelu daného uzlu. Druhým parametrem je kořenový prvek části validovaného diagramu, tedy buď instance `OPBPMNModel` nebo podprocesu, či bazénu. V metodě dojde ke kontrole, zda může být objekt validován. Poté až je vytvořena instance třídy a je provedena kontrola pomocí následující metody.

`validateObject:withModel:` (Inst. side) metoda obsažená v každém pravidle provádějící samotnou validaci. Obsahuje stejné argumenty jako předchozí metoda. Validace nejčastěji obsahuje sadu podmínek, které při porušení znamenají porušení pravidla. V tomto případě je návratovou hodnotou pole obsahující jednu, či více instancí `OPBPMNValidationMessage`, která obsahuje všechny náležitosti potřebné k popisu chyby. Pokud validační pravidlo není porušeno, je vrácena prázdná kolekce.

Příkladem implementace pravidel může být například třída `SE03MessageReceive`, která je samozřejmě potomkem třídy `OPBPMNValidation`. Tato třída, jak název napovídá, implementuje pravidlo SE03. Obsahuje pouze dvě metody, `validateClass:`, která obsahuje pouze návratovou hodnotu `^ OPBPMNStartEventModel`. Druhá metoda implementuje pravidlo, viz ukázka kódu 4.2.

Výpis kódu 4.2 Ukázka validace pravidla SE03. (*Start Event* typu *Message Receive* musí mít příchozí *Message Flow*.)

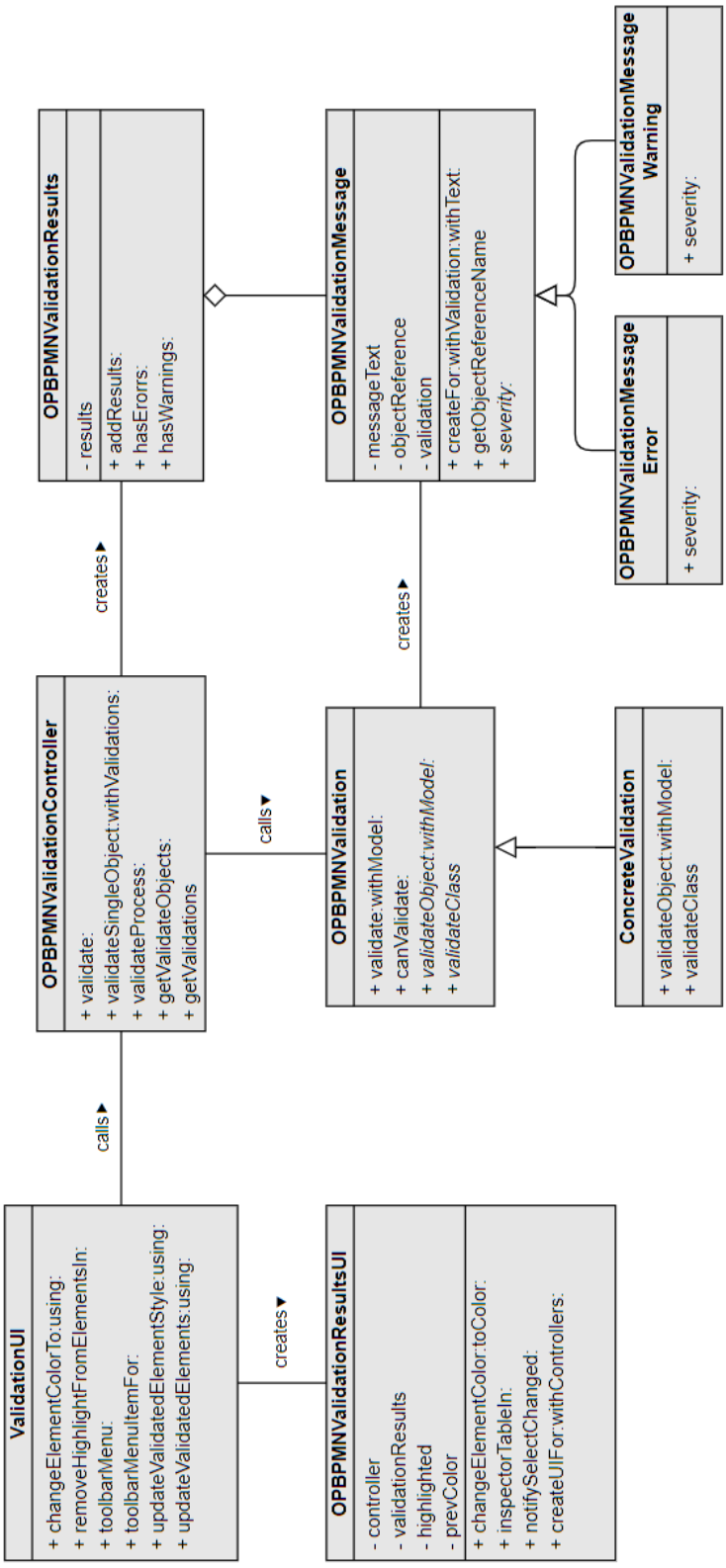
```
validateObject: aValidatedObject withModel: aValidatedModel

    aValidatedObject type = OPBPMNNodeType Receive
    & (aValidatedObject incoming size = 0) ifTrue: [
        ^ { (OPBPMNValidationMessageError
              createFor: aValidatedObject
              withValidation: self
              withText: '(SE03) ' , aValidatedObject class name,
                  ' of Message Receive type has to have
                  incoming Message Flow.') } ].
    ^ {  }
```

`OPBPMNValidationResults` je třída, která shromažďuje všechny výsledky validací. Obsahuje množinu výsledků a metody, které nad touto množinou pracují. Mezi ně patří inicializace, getter a setter. Mezi další metody patří:

`addResults:` přidá do množiny celou kolekci výsledků, případně jen jeden výsledek, pokud argumentem není kolekce.

`hasErrors:` obsahuje jeden argument, kterým je kontrolovaný uzel. Metoda vrací boolean, zda uzel obsahuje chyby, aby bylo možné jej zvýraznit správnou barvou.

`hasErrors:` totéž jako předchozí metoda.



Obrázek 4.2 Schéma tříd implementovaného balíčku pro validaci. (Obrázek autora)

`OPBPMNValidationMessage` poskytuje funkcionality pro uložení jednotlivých zpráv, tedy chyb (podtřída `OPBPMNValidationMessageError`) a varování (podtřída `OPBPMNValidationMessageWarning`). Ke každé zprávě je uložen text, reference na objekt, ke kterému se vztahuje a reference na validaci, která zprávu vyvolala. To umožňuje přesně dohledat, co chybu vyvolalo, v jakém prvku a proč. Obě podtřídy obsahují pouze metodu `severity:`. Hlavní třída obsahuje gettery a settery na tři již zmíněné proměnné, navíc obsahuje další metody:

`createFor:withValidation:withText:` metoda pro nastavení hodnot, použití této metody pro korektní vytvoření validace bylo možné vidět v ukázce kódu 4.2. Prvním argumentem je model uzlu, druhým odkaz na validaci a třetím je text chyby. Návratovou hodnotou je instance sebe sama.

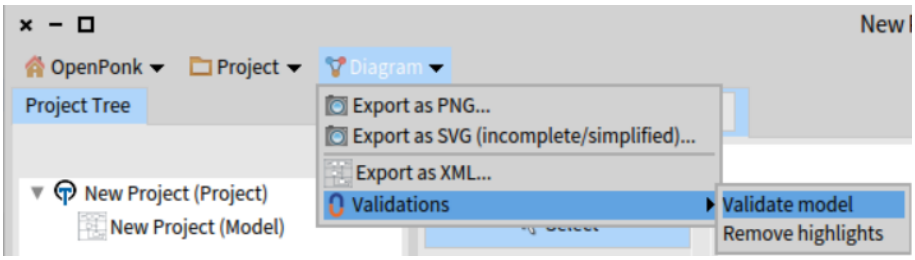
`severity:` metoda slouží k vypsání závažnosti směrem k uživateli. U chyb je vrácen text „Error“, u varování pak „Warning“.

Uživatelské rozhraní (UI) pro validace je vytvořeno pomocí dvou tříd. Konkrétně se jedná o úpravu menu a vložení potřebných tlačítek. Vytvoření dialogového okna inspectoru po dokončení validace a také jsou pomocí tříd UI obarvovány prvky, jejichž validace skončila s chybou. Poslední funkcionalitou je zvýraznění prvku podle výběru v tabulce s chybami. Zvýraznění obrysu prvku je nastaveno následovně:

- **Chyba** – Zvýraznění obrysu prvku červenou barvou.
- **Varování** – Zvýraznění obrysu prvku oranžovou barvou.
- **Výběr** – Zvýraznění obrysu prvku zelenou barvou. (Výběr chyby v seznamu zvýrazní prvek, kterého se týká zelenou barvou.)

`OPBPMNValidationUI` obsahuje úvodní metody pro spuštění validace:

`toolbarMenuItemFor:` slouží k přidání tlačítek do `OPEditorToolbarMenu`, záložky *Builder*, viz obrázek 4.3. Tlačítko validace obsahuje akci ke zvýraznění prvků, které budou obsahovat chybu po spuštění validace pomocí `OPBPMNValidationController`.



Obrázek 4.3 Toolbar menu s možností spuštění validace. (Obrázek autora)

`updateValidatedElementStyle:using:` slouží k obarvení obrysu prvku. Argumenty jsou reference na chybu, aby bylo možné určit závažnost a také controller daného prvku. Dle již popsaných barev. Původní barva obrysu je nastavena na černou.

`changeElementColorTo:using:` metoda použitá k zavolání view vrstvy daného prvku. Diagram controller každého uzlu poskytuje k tomuto účelu metodu `borderColor:`. Argumenty jsou barva a controller prvku.

`removeHighlightsFromElementsIn:` metoda, která slouží k odebrání všech zvýraznění prvků. Spouští se pomocí tlačítka „Remove highlights“. Jediným argumentem metody je controller kořenového prvku.

`OPBPMNValidationResultsUI` slouží k vytvoření okna inspectoru, ve kterém je vytvořená customizovaná tabulka výsledků se sloupci: závažnost, prvek, text zdůvodnění. V metodě je potřeba si pamatovat vybraný výsledek v tabulce, který má zároveň zvýraznit prvek zelenou barvou. Pro tyto účely je potřeba vědět i jeho původní barvu.

`inspectorTableIn:` vytváří tabulku výsledků pomocí třídy Spec 2, `SPTablePresenter`. Každý řádek představuje jedno porušení pravidel jednoho uzlu, tj. jedno pravidlo i jeden uzel mohou být v tabulce uvedeny víckrát. Při vybrání konkrétního řádku je zavolána metoda `notifySelectChanged:`.

`inspectorDocumentationReferenceIn:` vytváří v inspectoru záložku „Documentation“ pro odkaz na online dokumentaci.

`notifySelectChanged:` metoda sloužící ke změně výběru. Argumentem je řádek tabulky, ze kterého je možné vybrat konkrétní uzel.

`highlightElement:` provedení samotné změny barvy na daném uzlu. Barva zvýraznění vybraného prvku je změněna na zelenou. Je nutné uložit si původní barvu a změnit ji zpět u původního prvku (mohl být zvýrazněn kvůli chybě ve validaci např. červenou).

Výsledná implementace obsahuje navíc třídu `OPBPMNValidationFrameworkSettings`, která umožňuje možnost nastavení, zda bude validační pravidlo použito, či ne. Toto nastavení ale nebylo v práci použito, proto není třída podrobněji popsána. Jedná se však o možnost budoucího rozšíření.

4.3 Dokumentace a testování

Práce byla v průběhu testována jednotkovými testy. Tyto jednotkové testy nemají za úkol otestovat všechny funkcionality dané třídy, ale podchytit případné chyby fungování při změnách v jiných třídách, jelikož je tato práce vyvíjena současně s verzí jádra OpenPonku. Zároveň byl na konci práce proveden ruční test funkčnosti jednotlivých pravidel, z výsledků tohoto testování byla vytvořena stránka dokumentace. Bohužel nebylo provedeno uživatelské testování, uživatelsky bude aplikace testována v příštím semestru na předmětu BI-KOM na FIT ČVUT, jelikož se tento předmět se vyučuje pouze v zimním semestru.

Dokumentace je dostupná online na stránce github.com [43], jedná se o stejný repozitář, ve kterém je uložena platforma OpenPonk.

4.3.1 Jednotkové testování

Prostředí Pharo poskytuje základní rozhraní pro jednotkové testování. U všech tříd dědících od třídy `TestCase` se předpokládá, že obsahují nějaké jednotkové testy, ty je pak možné spouštět pomocí ikony pro celou třídu najednou. Jeden test je v kódu vytvořen pomocí klíčového slova `assert`, nejčastěji používané s rovností `equals`. Pokud je testovaná hodnota vyhodnocena jako správná, je test úspěšně dokončen, v opačném případě je uživateli na obrazovce zobrazena chybová hláška.

Testy pro jednotlivé balíčky jsou v nich odděleny pod tagem *Tests*.

4.3.1.1 Jednotkové testování OpenPonk-BPMN

Pro potřeby testování jednotlivých prvků v diagramech je v jádru OpenPonku vytvořena třída `OPElementControllerTest`. Tato třída poskytuje čtyři jednoduché metody pro kostru testování, je potřeba v každém potomkovi definovat: třídu controlleru, třídu modelu, třídu kořenového prvku diagramu a třídu diagram controlleru prvku. Pro potřeby BPMN rozšíření byla vytvořena

třída `OPBPMNControllerTest`, která doplňuje základní metody o tři jednoduché testy: test modelu, vytvoření controlleru a vytvoření modelu.

`OPBPMNNodeControllerTest` kontroluje základní vlastnosti uzlů. U některých specifických uzlů, jako například bazénu, mohou být navíc také specifické testy pro daný prvek.

`testAssignmentToParent` je test přiřazení modelu nadřazenému prvku. Tedy vytvoření a následná kontrola, zda nadřazený prvek obsahuje ve své kolekci prvků.

`testClassSideModelClass` je test, zda třída modelu odkazovaná ze třídy controlleru odpovídá skutečně vytvářené třídě.

`testCreateController` je test, zda testovaná třída controlleru odpovídá skutečně vytvářené třídě.

`testCreateModel` je test, zda testovaná třída modelu odpovídá skutečně vytvářené třídě.

`testFigureRefresh` je test, který vytvoří daný prvek a otestuje, zda se při změně popisku prvku popisek opravdu změní také ve třídě modelu.

Testovací konkrétních prvků notace BPMN obsahují vždy metody pro určení třídy controlleru, modelu, diagramElementu a cílového controlleru s modelem pro test vytvoření. Cílový prvek byl u většiny stanoven jako bazén, pouze *Boundary Event* může být vytvořen pouze v podprocesu. Tato sada testů neslouží k testování všech konkrétních vlastností daných prvků. Konkrétní vlastnosti se mnohem lépe testují vizuální kontrolou při modelování. Testy slouží ke kontrole, zda je možné všechny prvky bezpečně vytvořit i po provedení změn v jiných částech kódu.

`OPBPMNFlowControllerTest` kontroluje základní vlastnosti hran, konkrétně zda dojde k vytvoření hrany mezi dvěma prvky. V tomto případě je složitější samotná příprava testu, pomocí metody `setUp` jsou vytvořeny dva prvky, ke kterým je možné hrany připojit. Třída obsahuje čtyři základní testy vlastností hran.

`testCreateDiagramElement` testuje vytvoření prvku a zakreslení v diagramu.

`testCreateFigure` je komplexnější test, který zkoumá nejen správné vytvoření ale také, že jde opravdu o hranu, že je na obou koncích validně připojena, a že je zakreslena v diagramu.

`testCreateModel` testuje přiřazení hrany zdrojové i cílové modelové třídě.

`testFigureRefresh` je test, který vytvoří danou hranu a otestuje, zda se při změně popisku popisek opravdu změní také ve třídě modelu.

Podtřídami jsou třídy pro tři základní typy hran v implementaci. Oddělená je kontrola vytvoření značek na začátku a na konci hran.

4.3.1.2 Jednotkové testování OpenPonk-BPMN-Validation

Jednotkové testování v balíčku validace pokrývá nejdůležitější třídy rozšíření a jejich metody. Všechny testovací třídy jsou potomky `TestCase`, pro využití výhod jednoduchého testování.

`OPBPMNValidationControllerTest` testuje použití hlavních pěti metod. Zaměřuje se na jednoduchý test jedné až dvou validací. Cílem těchto testů je ověřit, zda vůbec nějaká validace probíhá, a zda vrací výsledky.

`testGetValidatedObjects` testuje zařazení objektů do testu.

`testGetValidations` testuje zařazení validací do testu. (Připraveno na možnost zapínání a vypínání různých pravidel v nastavení.)

`testValidate` testuje správnost výsledků u validování dvou jednoduchých objektů.

`testValidateProcess` testuje validaci procesu z pohledu celku, opět jsou vytvořeny pouze dva prvky – *Start Event* a *Activity*, které jsou postupně vkládány do modelu pro validace. Tímto jsou testována pravidla s označením „PR“.

`testValidateSingleObject` testuje jeden prvek a výsledek provedení validací vzhledem k tomuto prvku.

`OPBPMNValidationMessageTest` je třída testů k vytváření zpráv. V těchto testech dochází ke kontrole správnosti uložení textu, reference na prvek a typu zprávy. Třída obsahuje sedm testovacích metod, mezi ně patří také:

`testValidationError` testuje na jednoduché zprávě, zda je závažnost zprávy chyba.

`testObjectReference` testuje, zda je proměnná odkazující na objekt připojený ke zprávě, referencí na správný objekt.

`OPBPMNValidationResultsTest` slouží k testování kolekce chybových zpráv a jejich rozdělení na chyby a varování. Metody vytvářejí jednoduché zprávy a kontrolují jejich přiřazení do kolekce a rozpoznání typu.

`OPBPMNAbstractValidationTest` obsluhuje testování jednotlivých validací. Pro každé validační pravidlo jsou připraveny tři základní testy, které testují zda je možné pravidlo spustit. Nejsou testovány funkčnosti jednotlivých pravidel. Funkčnost je testována ruční kontrolou v následující kapitole. Tři testy pro každé pravidlo:

`canValidateTest` kontroluje, zda pravidlo může validovat třídu, kterou obsahuje v metodě `canValidate`.

`validatedClassTest` testuje, zda je dodaná třída validovatelná.

`newTest` testuje, že validační třída neimplementuje metodu `new`. Validace jsou vytvářeny pouze pomocí `basicNew`.

4.3.2 Testování validačních pravidel

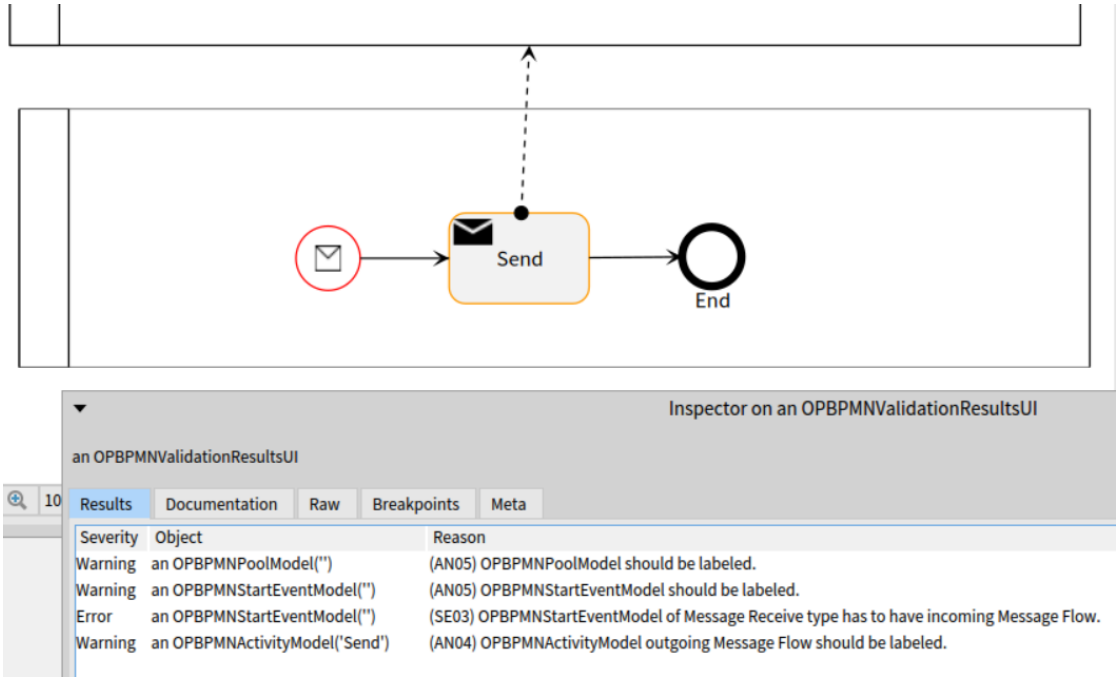
Pro každé pravidlo byly vytvořeny dvě jednoduché modelové situace. První, která pravidlo porušuje. Druhá, která by měla proběhnout bez chyb. Pro obě modelové situace byla spuštěna validace a zapsány výsledky.

Důležitým bodem testování byl fakt, že druhá modelová situace musela plně proběhnout bez chyb validace, tedy ani porušit jiná pravidla. Tento způsob odhalil velké množství chyb v kódu, protože se prakticky jednalo o 35 testů každého pravidla. Například při tvorbě testu pro pravidlo AN04 bylo nutné popsat oba bazény a zvolit správné typy prvků. Ukázku tvorby pravidel a jejich validaci poskytuje obrázek 4.4.

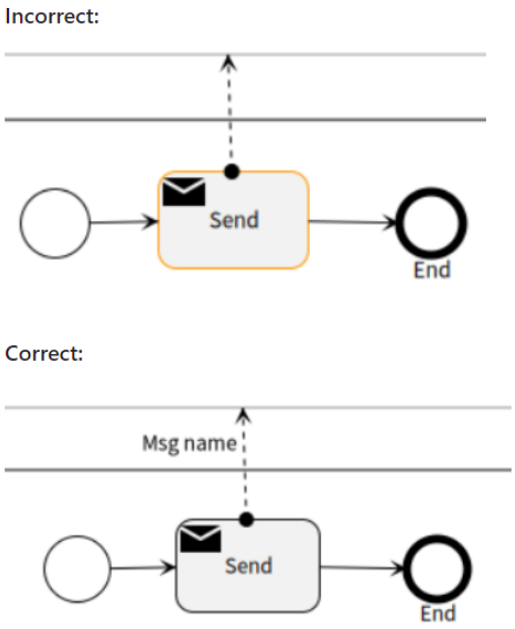
Obrázky výsledků byly zároveň použity jako obrázky pro dokumentaci. Samozřejmostí je, že jsou tímto způsobem testování vyzkoušeny pouze dvě možnosti, avšak plnohodnotné jednotkové testování by v tomto případě 35 tříd bylo velice náročné. Navíc by nedošlo k otestování funkcí uživatelského rozhraní, jako je změna barvy prvku při jeho chybě.

Následující bode je ukázkou provedených testů, všechny je možné nalézt v online dokumentaci [43] nebo v příloze práce.

AN04 (Warning) Odchozí *Message Flow* by měla být popsána názvem zprávy. – Viz obrázek 4.5, horní část ukazuje špatně zakreslený diagram porušující toto pravidlo, spodní obrázek korektní použití.



Obrázek 4.4 Ukázka tvorby validačního pravidla AN04 a provedení validace s chybami k odstranění. (Obrázek autora)



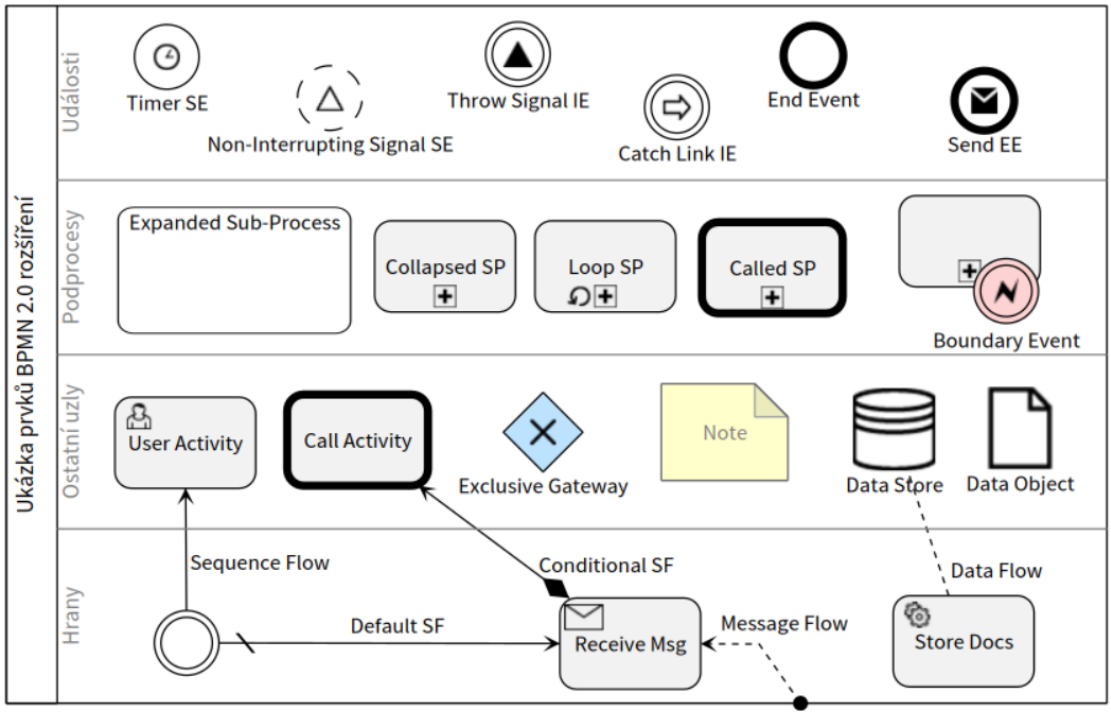
Obrázek 4.5 Test validačního pravidla AN04. (Obrázek autora)

4.4 Shrnutí výsledku implementace

Implementace BPMN modeláře navazovala na bakalářskou práci [1]. Výzvu ale představovaly změny v platformě OpenPonk, ze kterých můžeme jmenovat změnu architektury a rapidní změny

ve vykreslování prvků. Pro porovnání množství prvků obsahovala původní implementace 25 prvků jazyka, současná implementace poskytuje jen událostí přes 50 typů.

Implementace umožňuje téměř plnohodnotné procesní modelování BPMN 2.0. Z notace jazyka BPMN 2.0 byly vynechány pouze konverzace a choreografie, které jsou doplňujícími prvky notace (netvoří sadu procesního modelování, ale jiných typů diagramů). Ukázku možných prvků k vytvoření je možné si prohlédnout na obrázku 4.6.



Obrázek 4.6 Ukázka některých možných prvků k vytvoření v OpenPonk BPMN 2.0 rozšíření. (Obrázek autora)

Výsledná práce se bohužel neobešla bez dvou omezení: 1. Dráhy bazénu rozdělují bazén na stejně velké díly a nelze s nimi hýbat. Toto omezení může představovat problém zhoršení čitelnosti diagarmu v případě rozsáhlých diagramů, ve kterých může vystupovat několik osob. 2. Podproces může mít pouze jeden *Boundary Event*, který je umístěn v pravém dolním rohu podprocesu. Toto omezení může způsobit problém nemožnosti modelování některých diagramů, což se může stát v případě několika typů vrácených výjimek z podprocesu.

V druhé části implementace byl vytvořen doplněk pro validaci BPMN 2.0 diagramů. Návrh validačních pravidel vychází z pravidel používaných třemi z největších propagátorů notace BPMN: BPMN Method and Style [3], bpmnlint od společnosti Camunda [29] a Signavio Best Practice [25].

Výsledná sada pravidel obsahuje 35 pravidel, z tohoto počtu bylo implementováno 34 pravidel. Každé pravidlo obsahuje jeden ze 3 stupňů závažnosti (Zaručeno, Chyba, Varování), chybový kód a popis. Validaci BPMN diagramu je možné spustit stisknutím tlačítka „Validate model“, po provedení kontroly je uživateli zobrazeno okno s tabulkou nalezených chyb. V diagramu jsou barevně označeny prvky s chybou červeně, prvky s varováním oranžově a označený prvek zeleně. V záložce „Documentation“ je možné otevřít odkaz na dokumentaci k pravidlům.

Kapitola 5
Závěr

Cílem této práce bylo rozšířit modul pro BPMN modelování, který vznikl v rámci bakalářské práce [1]. Modul měl být rozšířen tak, aby pokrýval většinu prvků notace BPMN 2.0 a umožňoval automatizovanou kontrolu kvality BPMN modelů. Tento cíl byl splněn.

Výsledná práce pokrývá všechny prvky notace BPMN 2.0 určené k procesnímu modelování. Celkem se jedná zhruba o 80 prvků, které je možné spojovat pomocí 5 typů hran. Při modelování se uživatel bohužel setká s jedním omezením, tím je možnost použití maximálně jednoho *Boundary Event* v podprocesu. V praxi je však pravděpodobnost nutnosti použití více než jednoho *Boundary Event* velmi malá.

Oddělený balíček pro validaci BPMN diagramů poskytuje možnost automatizované kontroly sémantické správnosti BPMN modelu. Samotnou validaci je možné spustit stisknutím tlačítka „Validate model“, validace poskytuje kontrolu pomocí 35 validačních pravidel navržených v rámci této práce. Sada pravidel byla navržena na základě specifikace jazyka BPMN 2.0.2 a tří nejpoužívanějších sad pravidel ze známých komerčních nástrojů.

Po provedení validace je uživateli zobrazena tabulka s výsledkem kontroly. Tyto výsledky zobrazují závažnost chyby a její popis. Obsahují také možnost označení prvku, ke kterému se chyba vztahuje. Pro více informací k provedení validace a sadě pravidel je na obrazovce v záložce „Documentation“ k dispozici odkaz na online dokumentaci [43]. Oproti jiným nástrojům poskytuje tvorba v prostředí Pharo možnost dynamických změn a přizpůsobení vlastním potřebám také pro koncové uživatele.

Výsledný rozšířený modul byl přepracován tak, aby bylo možné jej publikovat v nové, zatím nevydané verzi platformy OpenPonk, pro prostředí Pharo 11. Oproti současné verzi 3.0.4 platformy OpenPonk, kompatibilní s Pharo 9, přináší nová verze zásadní změny architektury jádra implementace OpenPonku, rozšíření pro vykreslování a uživatelského komfortu. V nové verzi se uživatelé mohou těšit také na realizaci podnětů diskutovaných v rámci této práce.

Přínosem této práce je možnost publikovat BPMN 2.0 modul jako plnohodnotnou součást platformy OpenPonk, a tím rozšířit portfolio projektů a spoluprací, které jsou s platformou spojené. Zároveň je díky možnosti kontroly správnosti modelů vhodné použít tento nástroj při výuce jazyka BPMN na Katedře softwarového inženýrství FIT ČVUT.

Vzhledem k neustálému vývoji platformy OpenPonk bude potřeba i do budoucna BPMN 2.0 modul udržovat aktuální. Zároveň existuje mnoho možností dalšího rozšiřování modulu o neprocesní část notace BPMN, či další validační pravidla, která nikdy nedokáží pokrýt všechny možné chyby při modelování. Věřím, že se nám společně s Centrem pro konceptuální modelování a implementaci podaří BPMN 2.0 modul nadále udržovat a rozšiřovat.

Příloha A

Dokumentace sady validačních pravidel k vzniklému nástroji

Dostupné také na github.com. [43]

OpenPonk BPMN Validation rules

The best from BPMN Method and Style, bpmnlint validation rules and Signavio Best Practice.

Severity of rules:

- **Guaranteed** - You are not able to model this.
- **Error** - You should remove the Error because it can cause fatal problems in running process.
- **Warning** - Is better to make it clear but the model is still correct according to BPMN documentation.

Sequence Flow

SF01 (Guaranteed)

Sequence Flow must connect to a flow node (activity, gateway, or event) at both ends.

SF02 (Guaranteed)

Sequence Flow may not cross a pool (process) boundary.

Message Flow

MF01 (Guaranteed)

Message Flow may not connect nodes in the same process (pool).

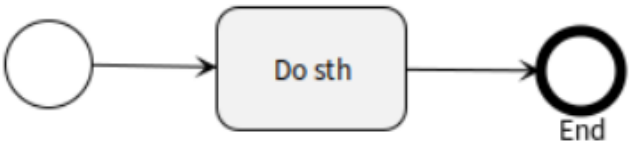
All Nodes

AN01 (Error)

All flow nodes other than Start Events, Boundary Events, and catching Link Events must have an incoming Sequence Flow.

Incorrect:

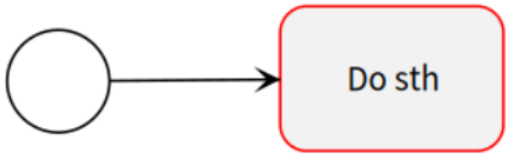
Correct:



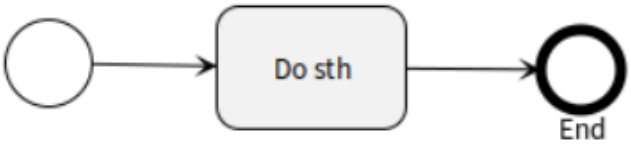
AN02 (Error)

All flow nodes other than end events and throwing Link events must have an outgoing sequence flow.

Incorrect:



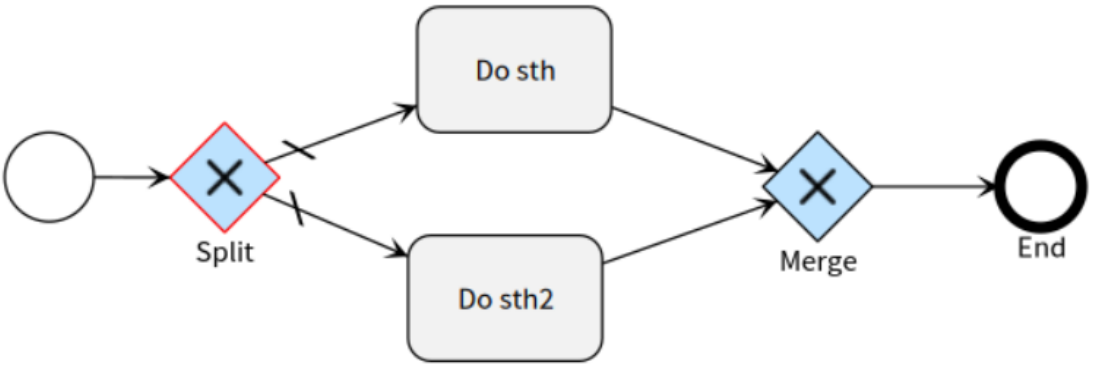
Correct:



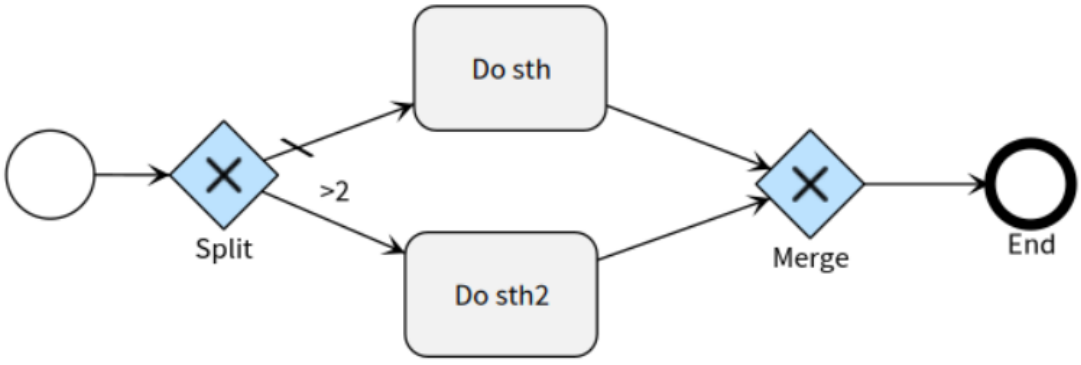
AN03 (Error)

Outgoing Default Sequence Flow can have only Exclusive, Inclusive, Complex Gateway or Activity. They can have only one Default Sequence Flow.

Incorrect:



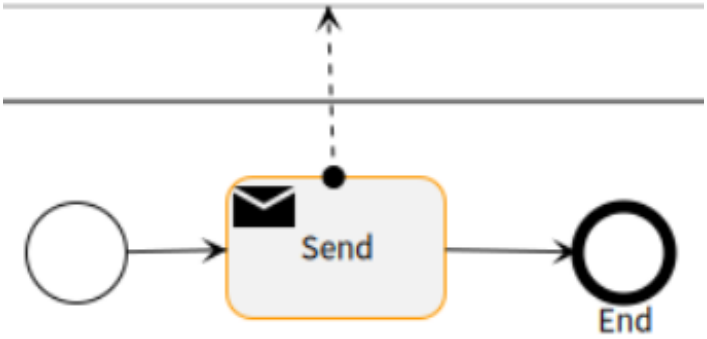
Correct:



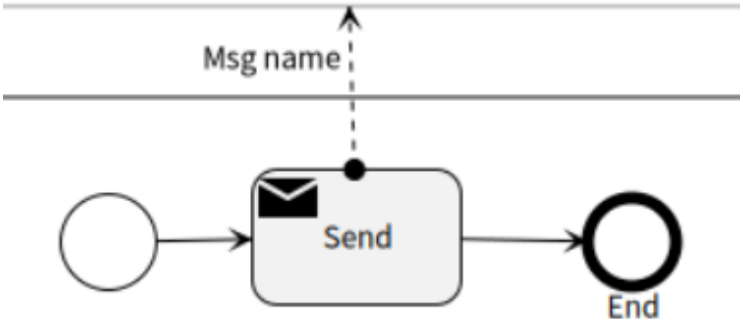
AN04 (Warning)

Outgoing Message Flow should be labeled by message name.

Incorrect:



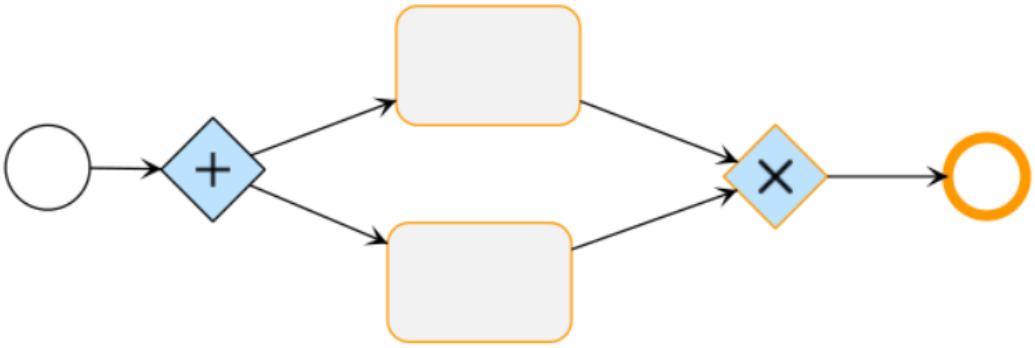
Correct:



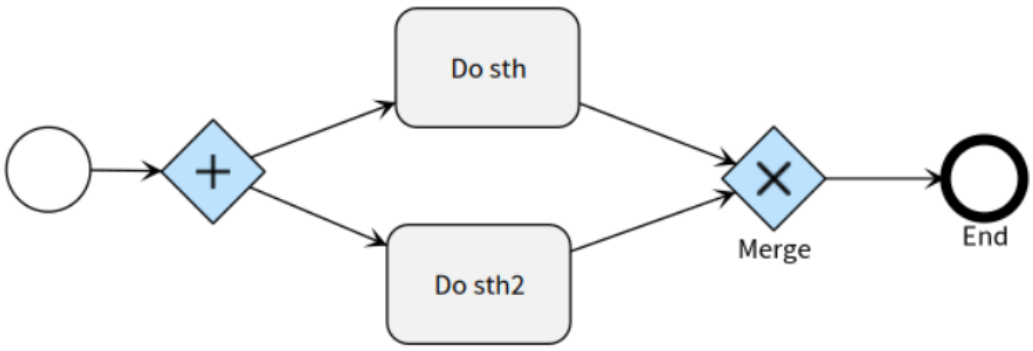
AN05 (Warning)

All nodes, except no type Start Event and Parallel Gateway, should be labeled.

Incorrect:



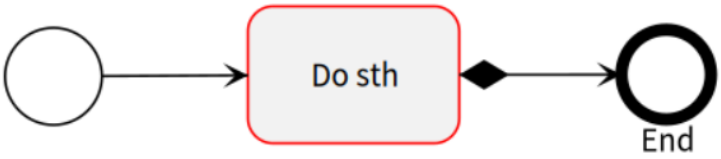
Correct:



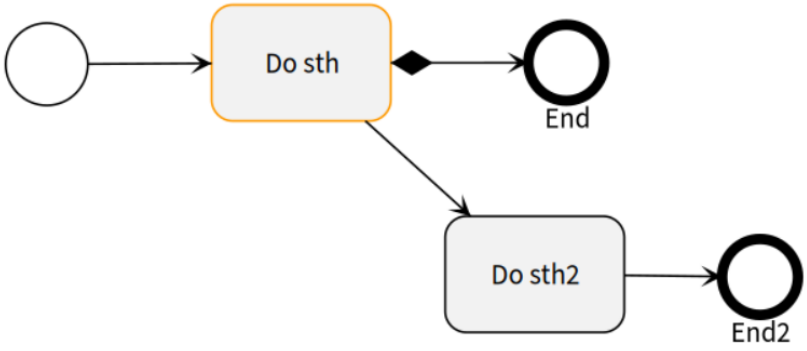
AN06 (Error)

A Conditional Sequence Flow may not be used if it is the only outgoing Sequence Flow

Incorrect:



Correct:
But you get Warning from the rule AC05.



Start Event

SE01 (Guaranteed)

Start Event must not have an incoming Sequence Flow.

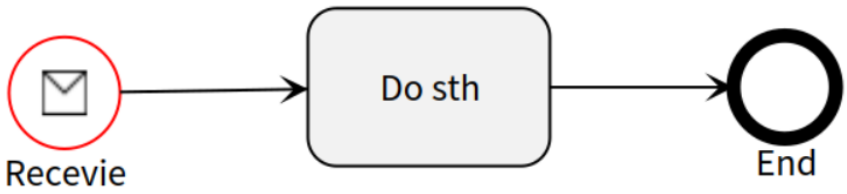
SE02 (Guaranteed)

Start Event must not have an outgoing Message Flow.

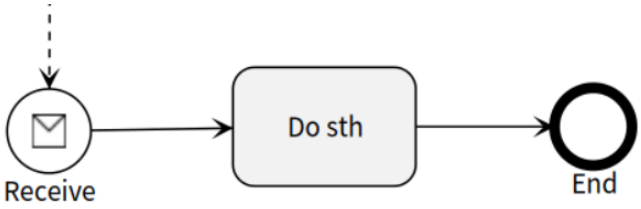
SE03 (Error)

Start Event of type Message must have incoming Message Flow.

Incorrect:



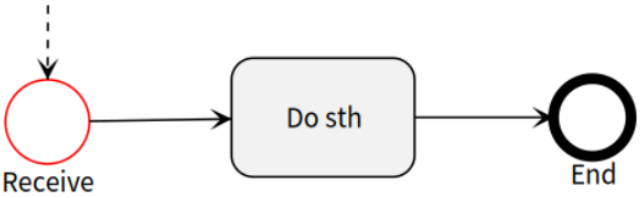
Correct:



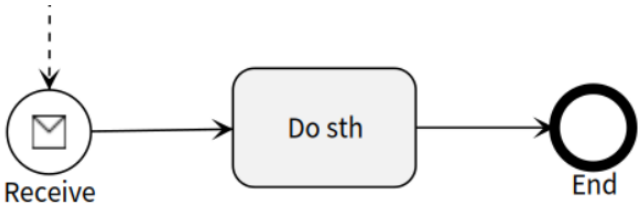
SE04 (Error)

Start Event with the incoming Message Flow must be of Message or Multiple type.

Incorrect:

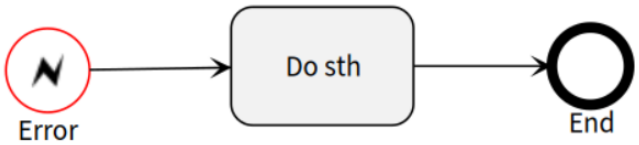


Correct:

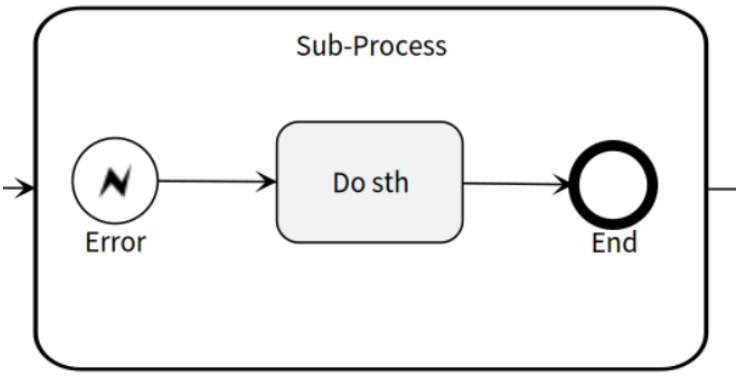


SE05 (Error)

Start Event may not have an Error trigger. Except Event Subprocess Start Event. **Incorrect:**



Correct:



SE06 (Error) - Not implemented yet.

Start Event in a Subprocess must not have a type unless it is triggered based on some Event from parent process

End Event

EE01 (Guaranteed)

End Event must not have an outgoing Sequence Flow.

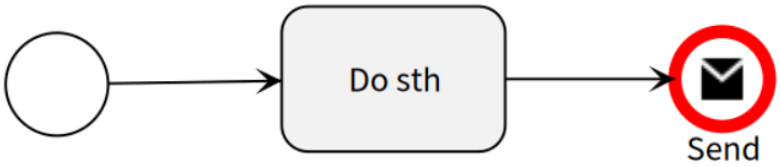
EE02 (Guaranteed)

End Event must not have an incoming Message Flow.

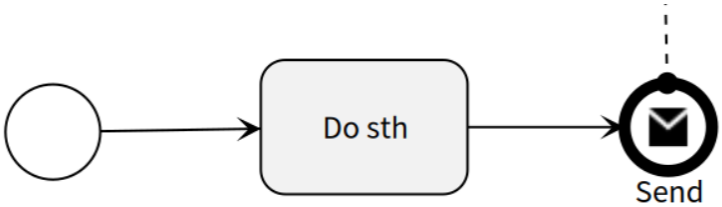
EE03 (Error)

End Event of type Message must have outgoing Message Flow.

Incorrect:



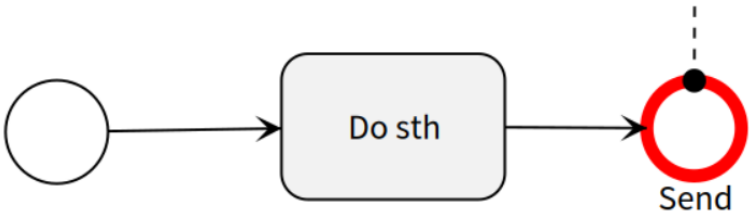
Correct:



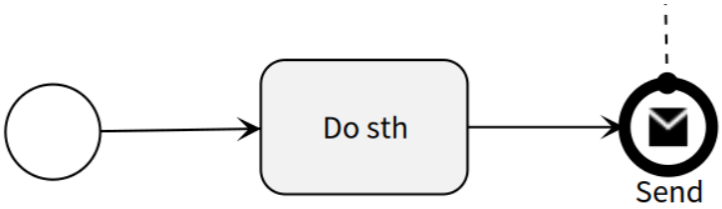
EE04 (Error)

End Event with the outgoing Message Flow must be of Message or Multiple type.

Incorrect:



Correct:



Boundary Event

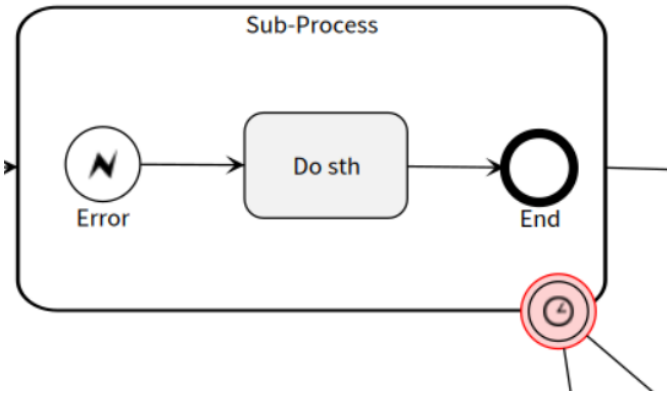
BE01 (Guaranteed)

Boundary Event must not have an incoming Sequence Flow.

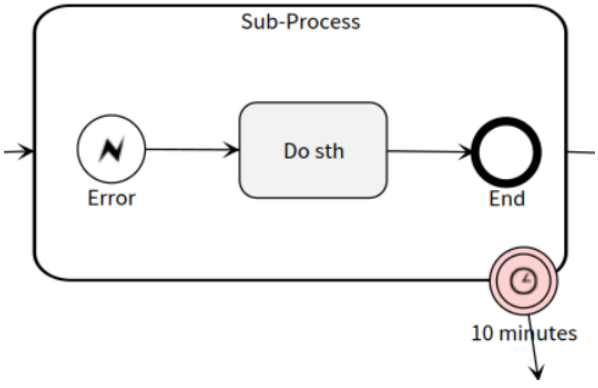
BE02 (Error)

Boundary Event must not have exactly one outgoing Sequence Flow. Except Compensation Boundary Event.

Incorrect:



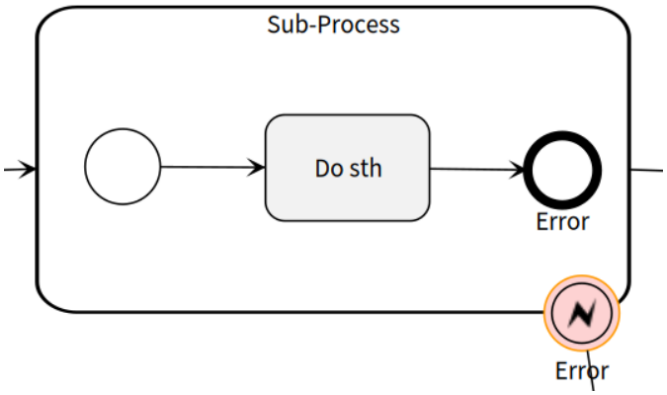
Correct:



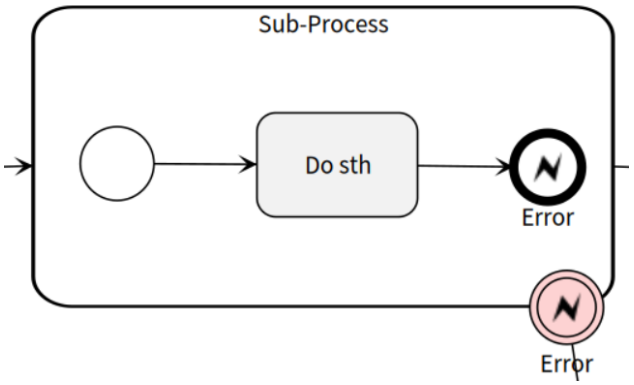
BE03 (Warning)

Error Boundary Event must contain a corresponding Error Throw Event in the Subprocess.

Incorrect:



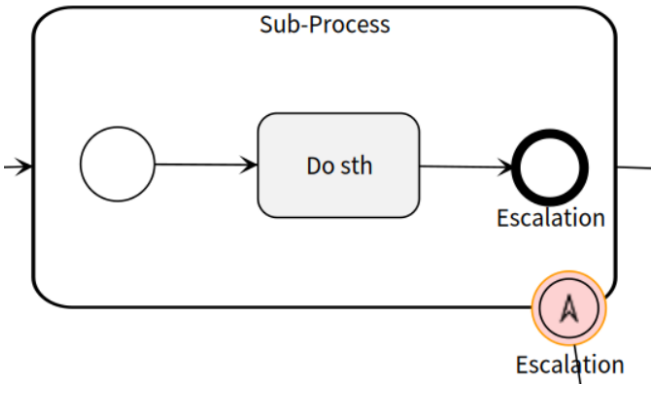
Correct:



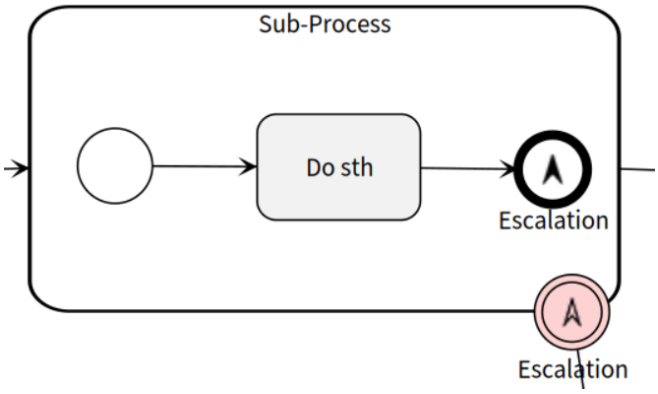
BE04 (Warning)

Escalation Boundary Event must contain a corresponding Escalation Throw Event in the Subprocess.

Incorrect:



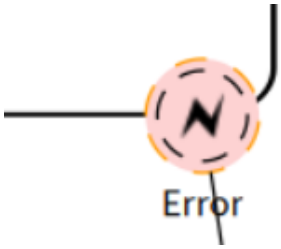
Correct:



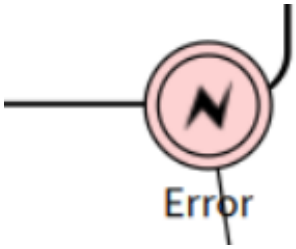
BE05 (Error)

Error Boundary Event may not be Non-Interrupting.

Incorrect:



Correct:

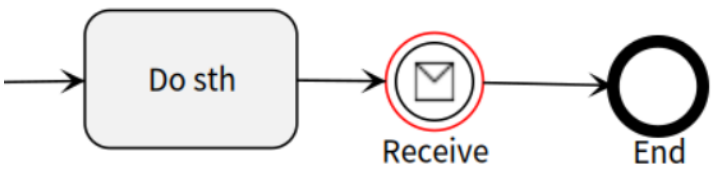


Intermediate Event

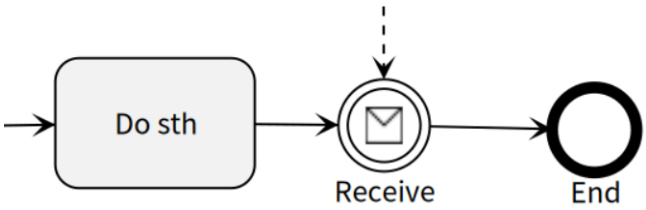
IE01 (Error)

Intermediate Event of type Message Receive must have incoming Message Flow.

Incorrect:



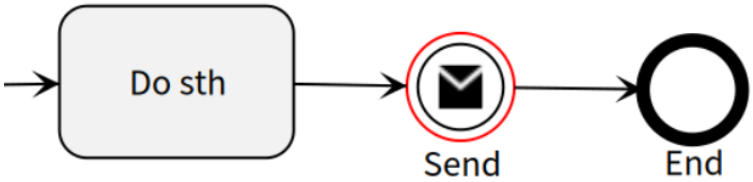
Correct:



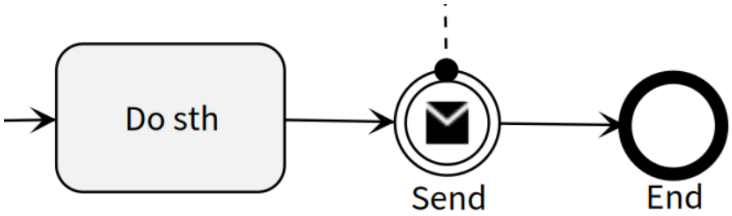
IE02 (Error)

Intermediate Event of type Message Send must have outgoing Message Flow.

Incorrect:



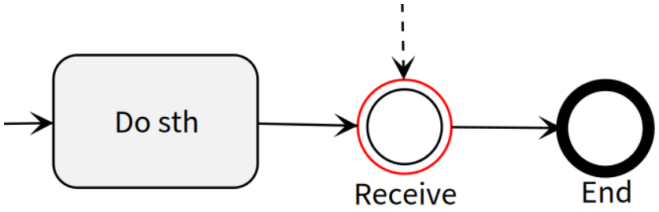
Correct:



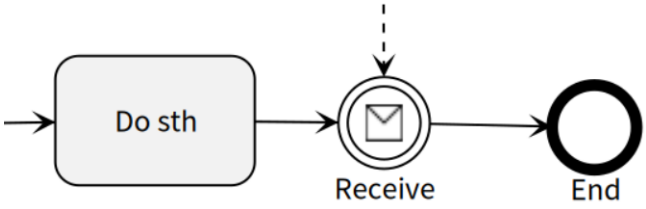
IE03 (Error)

Intermediate Event with the incoming Message Flow must be of Message Receive or Multiple type.

Incorrect:



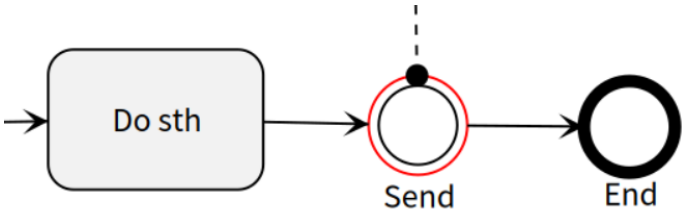
Correct:



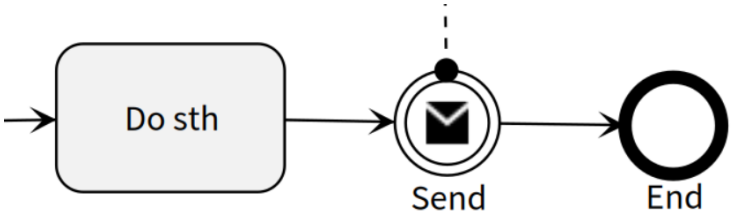
IE04 (Error)

Intermediate Event with the outgoing Message Flow must be of Message Send or Multiple type.

Incorrect:



Correct:



Gateway

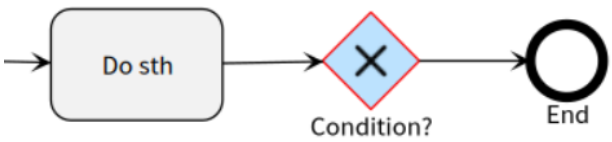
GW01 (Guaranteed)

Boundary Event must not have incoming or outgoing Message Flow.

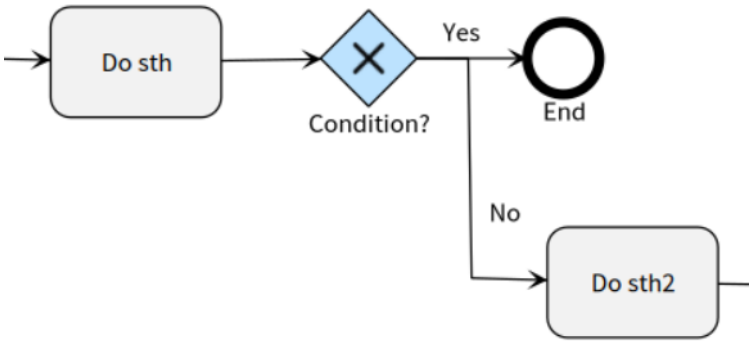
GW02 (Error)

Splitting Gateway must have more than one outgoing Sequence Flow. Merging Gateway must have more than one incoming Sequence Flow.

Incorrect:



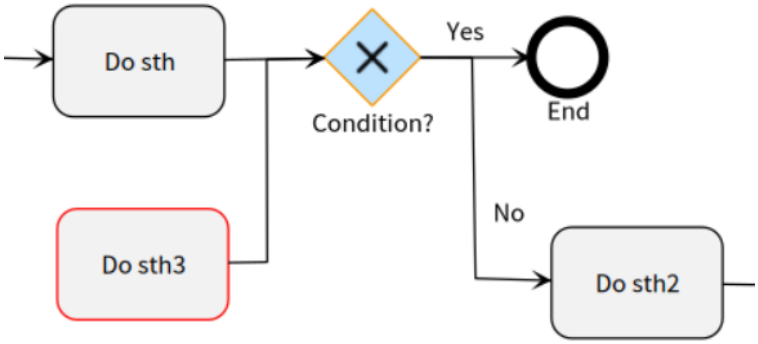
Correct:



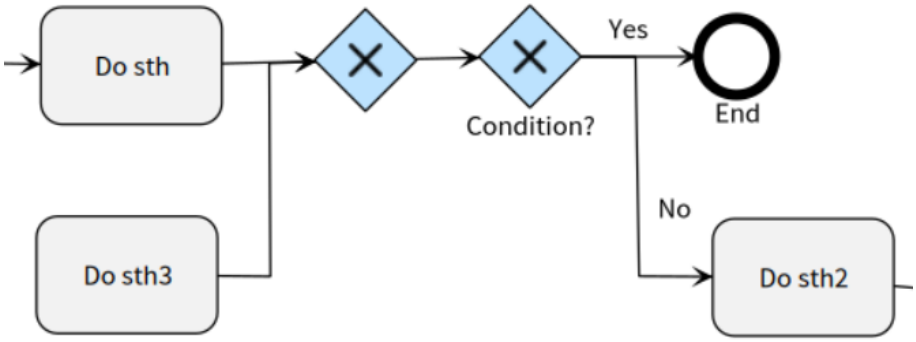
GW03 (Warning)

Gateway should not be both merging and spliting.

Incorrect:



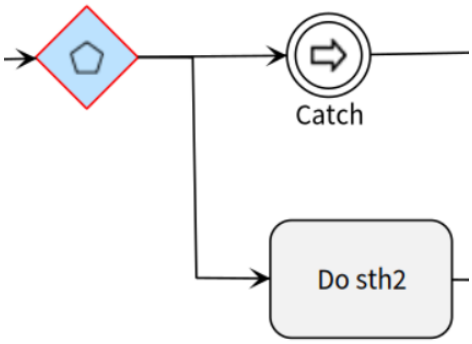
Correct:



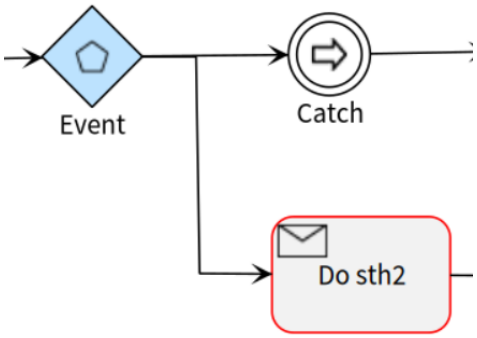
GW04 (Error)

Event Gateway can only contain Catching Intermediate Event or Receive Task in its branches.

Incorrect:



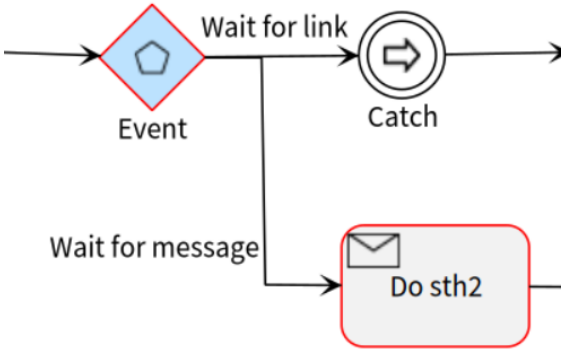
Correct:



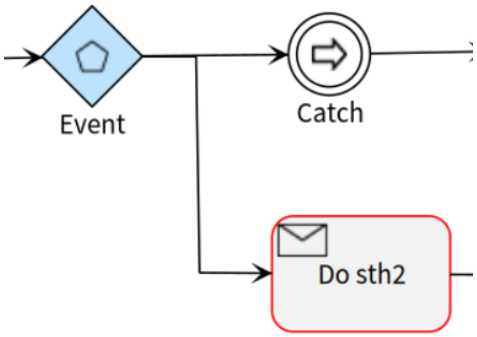
GW05 (Error)

Sequence Flow leading from Parallel/Event Gateway must not contain condition. (Label)

Incorrect:



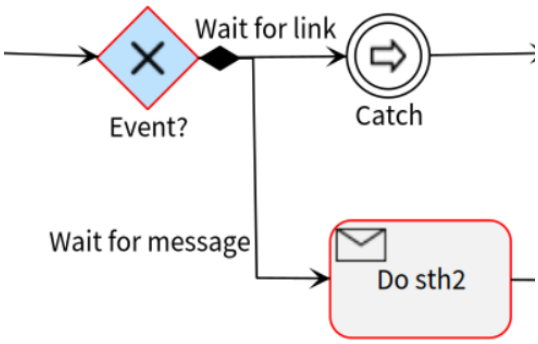
Correct:



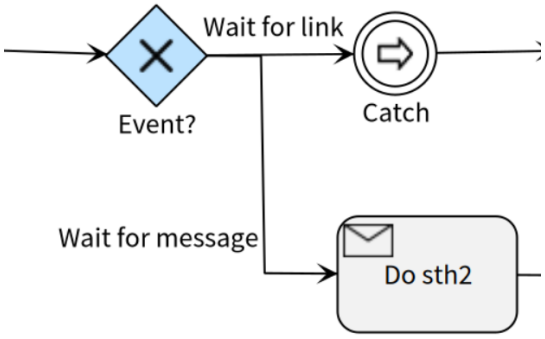
GW06 (Error)

Sequence Flow leading from Gateway must not be of conditional type. (Diamont at the beggining of Flow)

Incorrect:



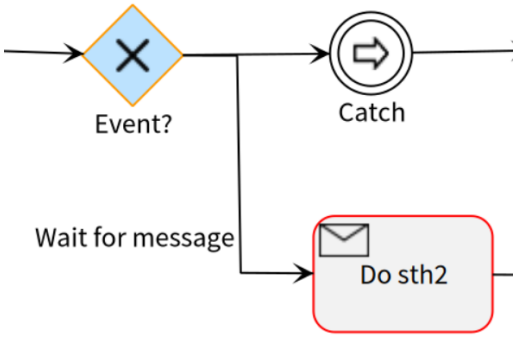
Correct:



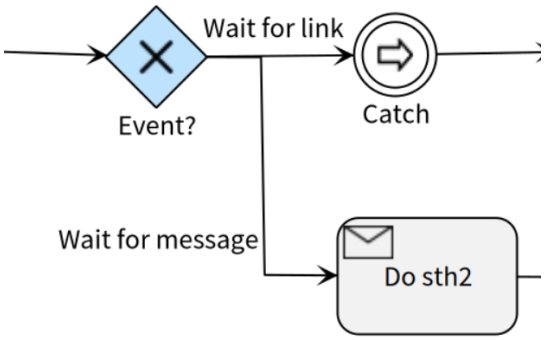
GW07 (Warning)

Sequence Flow leading from Gateway (of a different type than Parallel and Event and beyond default edges) should be labeled.

Incorrect:



Correct:

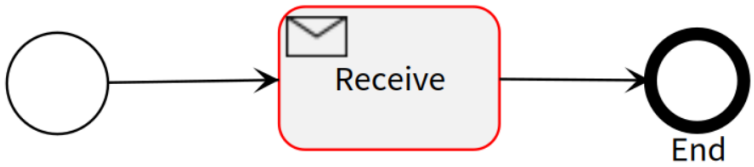


Activity

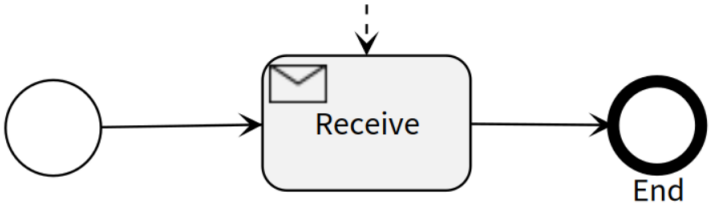
AC01 (Error)

Activity of type Message Receive must have incoming Message Flow.

Incorrect:



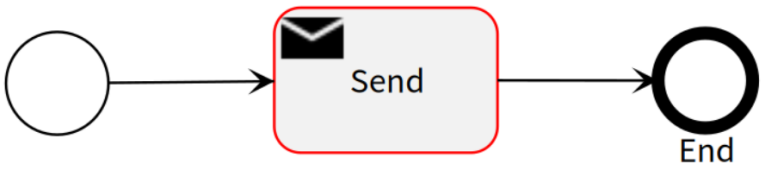
Correct:



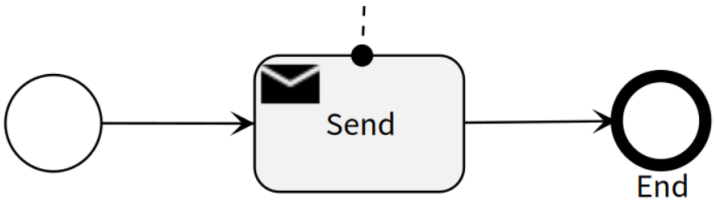
AC02 (Error)

Activity of type Message Send must have outgoing Message Flow.

Incorrect:



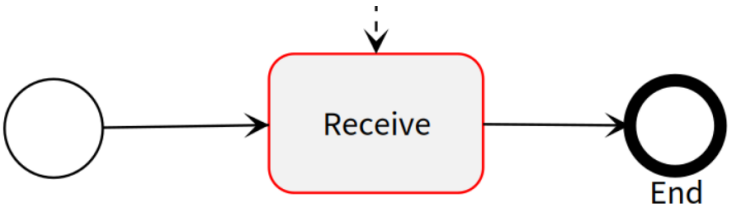
Correct:



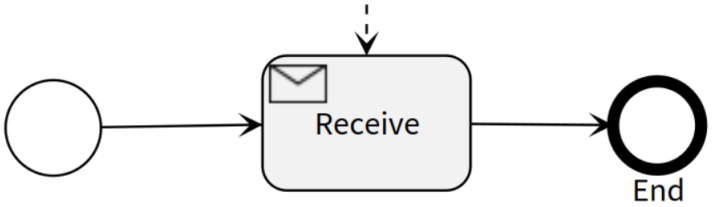
AC03 (Error)

Activity with the incoming Message Flow must be of Message Receive type.

Incorrect:



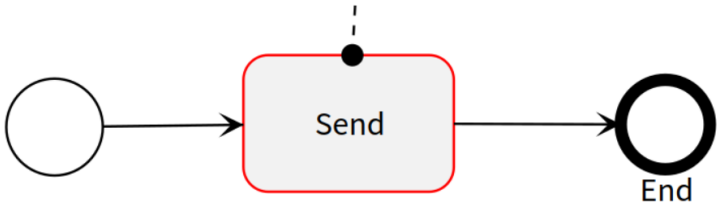
Correct:



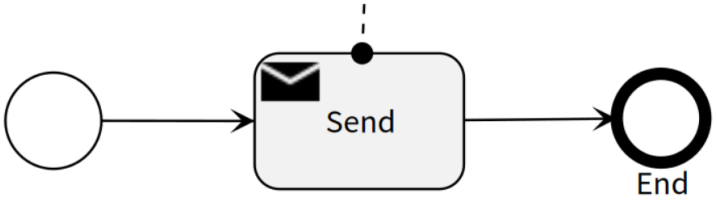
AC04 (Error)

Activity with the outgoing Message Flow must be of Message Send type.

Incorrect:



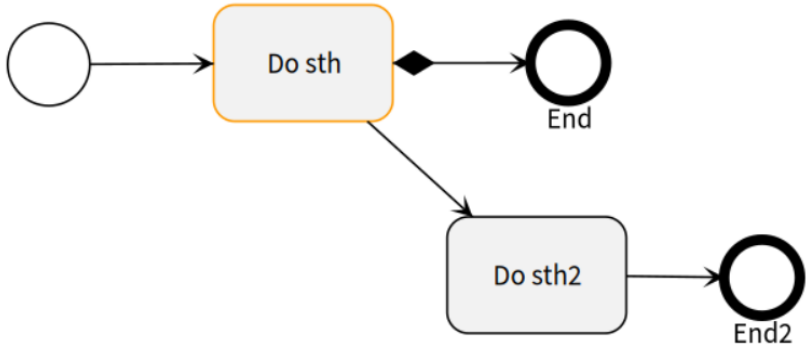
Correct:



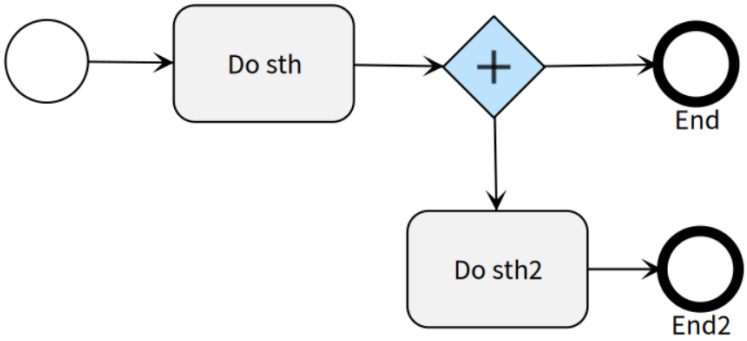
AC05 (Warning)

Activity should not be used to merge or split flow.

Incorrect:



Correct:



Process (Pool)

PR01 (Guaranteed)

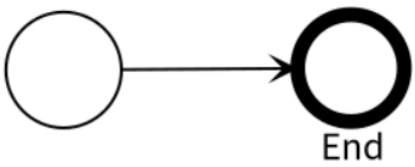
Pool cannot contain another Pool.

PR02 (Error)

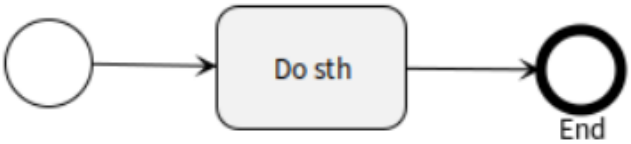
Process must have at least one Activity.

Class: PR2347ProcessErrors - called from OPBPMNValidationController - validateProcess:

Incorrect:



Correct:



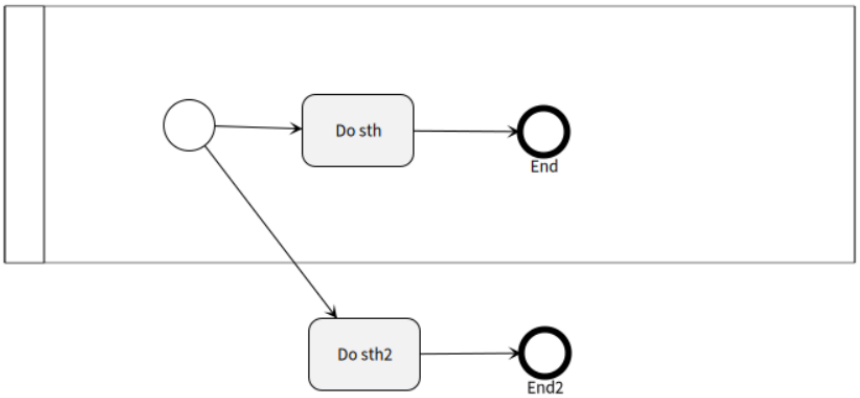
PR03 (Error)

Elements of at most one process can be contained only in one Pool.

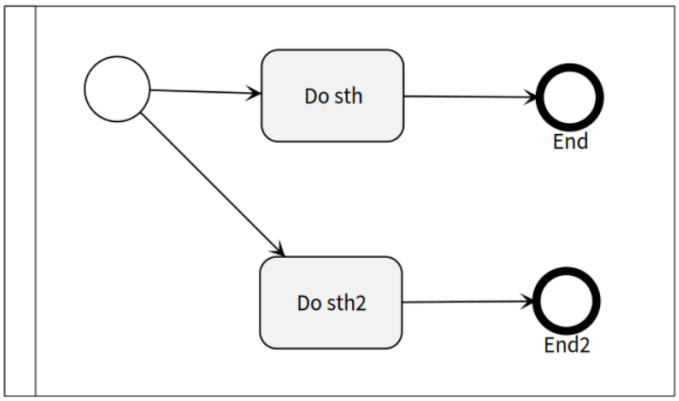
- You can get only one of PR03 and PR04 errors. - You get PR03 error if there is some Activity outside of Pool.

Class: PR2347ProcessErrors - called from OPBPMNValidationController - validateProcess:

Incorrect:



Correct:



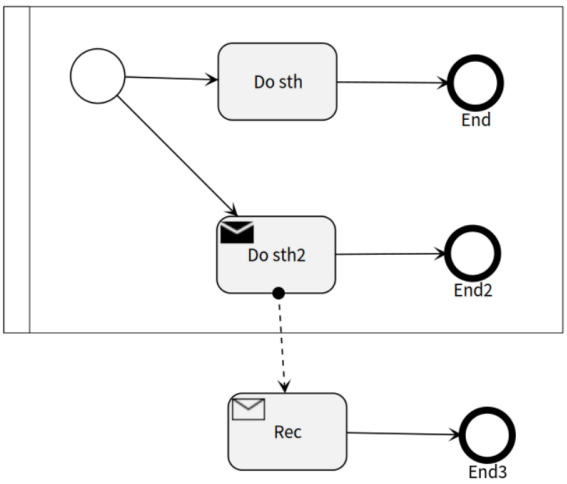
PR04 (Error)

If the diagram contains only one participant, it does not have to be drawn. If there is more, all elements must be drawn inside Pools.

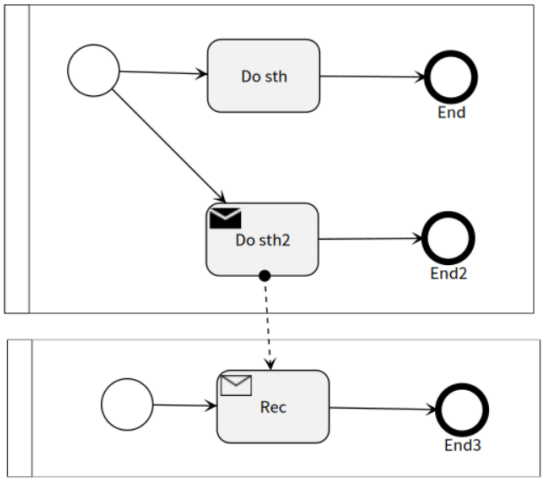
- You can get only one of PR03 and PR04 errors.

Class: PR2347ProcessErrors - called from OPBPMNValidationController - validateProcess:

Incorrect:



Correct:



PR05 (Error)

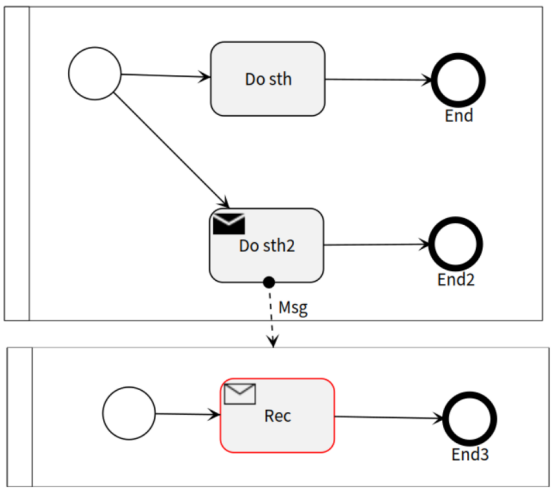
Pool can be source of a Message Flow only as a Black-Box. - Cannot contain elements.

Same as PR06

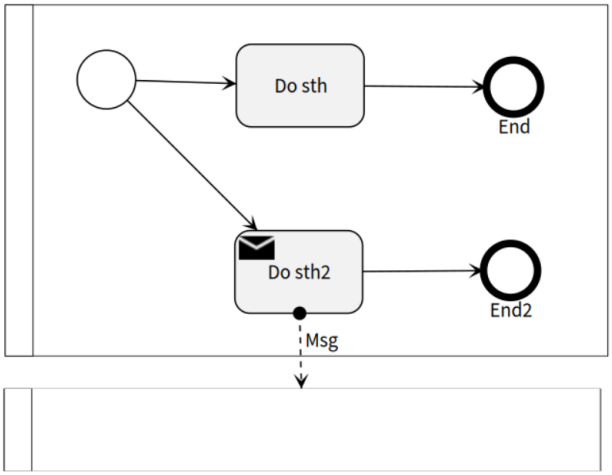
PR06 (Error)

Pool can be target of a Message Flow only as a Black-Box. - Cannot contain elements. Otherwise use Activity/Event as target.

Incorrect:



Correct:



PR07 (Warning)

Process should contain Start and End Event.

Class: PR2347ProcessErrors - called from OPBPMNValidationController - validateProcess:

Mostly you get AN01 or AN02 Error to this.

Bibliografie

1. PRIMUS, David. *Využití platformy OpenPonk pro orchestraci webových služeb pomocí BPMN* [online]. 2021. [cit. 2023-03-19]. Dostupné z: `https://dspace.cvut.cz/handle/10467/95376`. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce ING. ROBERT PERGL, PH. D.
2. *OpenPonk modeling platform* [online]. CCMI FIT ČVUT v Praze, 2022 [cit. 2023-04-10]. Dostupné z: `https://openponk.org`.
3. SILVER, Bruce. *BPMN method and style*. Second. USA: Cody-Cassidy Press, 2017. ISBN 9780982368114.
4. *Business Process Model and Notation v2.0.2* [online]. OMG, 2014 [cit. 2023-02-16]. Dostupné z: `https://www.omg.org/spec/BPMN/2.0.2/`.
5. *ISO/IEC 19510:2013 – Object Management Group Business Process Model and Notation* [online]. ISO, 2013 [cit. 2023-03-22]. Dostupné z: `https://www.iso.org/standard/62652.html`.
6. *Centrum pro konceptuální modelování a implementace* [online]. CCMI FIT ČVUT v Praze, 2018 [cit. 2023-04-10]. Dostupné z: `https://ccmi.fit.cvut.cz`.
7. GLIBRETH, Frank Bunker; GILBRETH, Lillian Moller. *Process Charts*. First. USA: American Society of Mechanical Engineers, 1921.
8. *Business Process Management Center of Excellence Glossary* [online]. Center of Excellence, 2009 [cit. 2023-03-26]. Dostupné z: `https://web.archive.org/web/20170131011831/https://www.ftb.ca.gov/aboutFTB/Projects/ITSP/BPM_Glossary.pdf`.
9. *Co je Workflow* [online]. ManagementMania, 2018 [cit. 2023-03-26]. Dostupné z: `https://managementmania.com/cs/workflow`.
10. AALST, W.M.P. van der; HOFSTEDE, A.H.M. ter; KIEPUSZEWSKI, B. Workflow Patterns. *Distributed and Parallel Databases*. 2003, s. 47. Dostupné z DOI: `https://doi.org/10.1023/A:1022883727209`.
11. *What Are Workflow Diagrams?* [Online]. IBM Cloud Education, 2021 [cit. 2023-03-26]. Dostupné z: `https://www.ibm.com/cloud/blog/workflow-diagrams`.
12. *Procesní řízení* [online]. BPM slovníček, 2007 [cit. 2023-04-07]. ISSN 1802-5676. Dostupné z: `http://bpm-slovnik.blogspot.com/2008/04/bpm.html`.
13. WESKE, Mathias. *Business Process Management*. First. Germany: Springer, 2007. ISBN 978-3-540-73521-2.
14. *What we do...differently* [online]. Object Management Group, 2023 [cit. 2023-04-28]. Dostupné z: `https://www.objectmanagementgroup.org`.

15. *BPM 2.0 Poster* [online]. BPM Offensive Berlin, 2015 [cit. 2023-03-21]. Dostupné z: http://www.bpmb.de.
16. *BPMN 2.0 Symbol Reference* [online]. Camunda, 2023 [cit. 2023-03-21]. Dostupné z: https://camunda.com/bpmn/reference/.
17. *BPMN Notation Overview* [online]. Visual Paradigm, 2022 [cit. 2023-03-21]. Dostupné z: https://www.visual-paradigm.com/guide/bpmn/.
18. *BPMN2.0 Event Examples* [online]. Sparx Systems, 2023 [cit. 2023-03-22]. Dostupné z: https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_simulation/bpmn2_0_events.html.
19. SUCHÁNEK, Marek. *OntoUML Pattern Catalogue* [online]. GitHub, 2023 [cit. 2023-04-28]. Dostupné z: https://ontouml.readthedocs.io/en/latest/patterns/index.html.
20. *Vertical pools* [online]. Bizagi Community — Bizagi Modeler, 2023 [cit. 2023-04-28]. Dostupné z: https://feedback.bizagi.com/en/topic/vertical-pools.
21. GEIGER, Matthias; HARRER, Simon; LENHARD, Jörg; WIRTZ, Guido. BPMN 2.0: The state of support and implementation. *Future Generation Computer Systems*. 2018, roč. 80, s. 250–262. ISSN 0167-739X. Dostupné z DOI: https://doi.org/10.1016/j.future.2017.01.006.
22. ALLANI, Olfa; GHANNOUCHI, Sonia Ayachi. Verification of BPMN 2.0 Process Models: An Event Log-based Approach. *Procedia Computer Science*. 2016, roč. 100, s. 1064–1070. ISSN 1877-0509. Dostupné z DOI: https://doi.org/10.1016/j.procs.2016.09.282. International Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016.
23. SILVER, Bruce. *BPMN Training* [online]. Method a Style, 2023 [cit. 2023-04-28]. Dostupné z: https://methodandstyle.com/training/bpmn-training/.
24. *SAP Signavio Process Manager* [online]. SAP Signavio, 2023 [cit. 2023-04-10]. Dostupné z: https://www.signavio.com/products/process-manager/.
25. *Guidelines by convention: Signavio Best Practice* [online]. SAP Signavio, 2023 [cit. 2023-04-10]. Dostupné z: https://www.modeling-guidelines.org/conventions/signavio-best-practice/.
26. *The dictionary* [online]. SAP Signavio, 2023 [cit. 2023-04-10]. Dostupné z: https://documentation.signavio.com/suite/en-us/Content/process-manager/userguide/intro-dictionary.htm.
27. *Camunda modeler* [online]. Camunda, 2023 [cit. 2023-04-10]. Dostupné z: https://camunda.com/download/modeler/.
28. *Web-based tooling for BPMN, DMN and Forms.* [Online]. Camunda, 2023 [cit. 2023-04-10]. Dostupné z: https://bpmn.io/.
29. *bpmnlint* [online]. Camunda, 2023 [cit. 2023-04-10]. Dostupné z: https://github.com/bpmn-io/bpmnlint.
30. *Activiti* [online]. Activiti, 2023 [cit. 2023-04-10]. Dostupné z: https://www.activiti.org.
31. *Activiti Designer BPMN features* [online]. Activiti, 2017 [cit. 2023-04-10]. Dostupné z: https://www.activiti.org/userguide/#eclipseDesignerBPMNFeatures.
32. HORN, Stephanie. *Introducing BPMN 2.0 in Visio* [online]. Microsoft, 2012 [cit. 2023-04-10]. Dostupné z: https://www.microsoft.com/en-us/microsoft-365/blog/2012/11/19/introducing-bpmn-2-0-in-visio/.
33. PARKER, David J. *Microsoft Visio 2013 business process diagramming and validation*. 1st. Packt Publishing, 2013. ISBN 9781782178019.

34. *Rules Reference* [online]. Sparx Systems, 2023 [cit. 2023-03-26]. Dostupné z: https://sparxsystems.com/enterprise_architect_user_guide/16.1/modeling_fundamentals/rules_reference.html.

35. UHNÁK, Peter; PERGL, Robert. The OpenPonk Modeling Platform. 2016. ISBN 97814503 45248. Dostupné z DOI: 10.1145/2991041.2991055.

36. *Pharo Features* [online]. Pharo, 2023 [cit. 2023-04-09]. Dostupné z: https://pharo.org/features.

37. GROVER, Jigyasa. *Wherre X=Smalltalk* [online]. Learn X in Y minutes, 2023 [cit. 2023-04-09]. Dostupné z: https://learnxinyminutes.com/docs/smalltalk/.

38. AKEVALION. *Creating a new visualization in Roassal* [online]. The Pharo Dev, 2022 [cit. 2023-04-09]. Dostupné z: https://thepharo.dev/2022/08/03/creating-a-new-visualization-in-roassal/.

39. JORDAN-MONTAÑO, Sebastian. *Dynamic layouts with Spec2* [online]. The Pharo Dev, 2021 [cit. 2023-04-07]. Dostupné z: https://thepharo.dev/2021/10/08/dynamic-layouts-with-spec2/.

40. BĚLOHOUBEK, Marek. *Verifikace OntoUML modelů na platformě OpenPonk* [online]. 2019. [cit. 2023-03-28]. Dostupné z: https://dspace.cvut.cz/handle/10467/83191. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce ING. ROBERT PERGL, PH. D.

41. HANKE, Dominik. *BASE64 Image* [online]. 2021. [cit. 2023-04-23]. Dostupné z: https://www.base64-image.de.

42. NIKKU. *BPMN 2.0 icon font* [online]. GitHub, 2023 [cit. 2023-04-23]. Dostupné z: https://github.com/bpmn-io/bpmn-font#readme.

43. PRIMUS, David. *OpenPonk BPMN Validation rules* [online]. GitHub, 2023 [cit. 2023-04-26]. Dostupné z: https://github.com/OpenPonk/OpenPonk-BPMN/blob/dev-validations-docs/docs/validation-rules.md.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `primus-dp` ..... složka se spustitelným programem
  - `openponk-OpenPonk-BPMN.bat` ..... skript pro spuštění
- `src`
  - `repository` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `dp-primus.pdf` ..... text práce ve formátu PDF