FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Využití platformy OpenPonk pro orchestraci webových služeb pomocí BPMN
Student:	David Primus
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2021/22

Pokyny pro vypracování

1. Seznamte se s notací BPMN a jejím uplatněním v orchestraci webových služeb.
2. Seznamte se s platformou OpenPonk.
3. Navrhněte architekturu modulu pro BPMN modelování v OpenPonk a implementujte prototyp BPMN modeláře zahrnující podmnožinu BPMN a potřebná rozšíření pro modelování orchestrace webových služeb.
4. Otestujte výsledek a demonstrujte jej na ukázkové studii.

Seznam odborné literatury

[1] https://openponk.github.io
[2] Silver, B. (2011). *BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A structured approach for business process modeling and implementation using BPMN 2.0*. Cody-Cassidy Press.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 24. ledna 2021

Bakalářská práce

VYUŽITÍ PLATFORMY OPENPONK PRO ORCHESTRACI WEBOVÝCH SLUŽEB POMOCÍ BPMN

David Primus

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
11. května 2021

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2021 David Primus. Všechna práva vyhrazena.
Tato práce vznikla jako školní díla na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bez uplatněných zákonných licencí nad rámec oprávnění uvedených v Prohlášení je nezbytný souhlas autora.

Odkaz na tuto práci: David Primus. *Využití platformy OpenPonk pro orchestraci webových služeb pomocí BPMN*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

1 Úvod 1

2 Cíle práce 3

3 Rešeršní část 5
3.1 Webová služba . . . 5
3.1.1 XML . . . 5
3.1.2 SOAP . . . 5
3.1.3 WSDL . . . 6
3.2 Servisně orientovaná architektura . . . 7
3.3 Orchestrace webových služeb . . . 7
3.3.1 Standardy k orchestraci webových služeb . . . 7
3.4 Podnikový proces . . . 9
3.4.1 Řízení podnikových procesů . . . 9
3.4.2 BPMS . . . 10
3.5 BPMN . . . 11
3.5.1 Základní prvky jazyka BPMN . . . 11
3.5.2 Použití jazyka BPMN . . . 13
3.5.3 Exportní formáty jazyka BPMN . . . 14
3.5.4 Nástroje pro jazyk BPMN . . . 15
3.6 OpenPonk . . . 17
3.6.1 Pharo . . . 17

4 Praktická část 19
4.1 Analýza a návrh . . . 19
4.1.1 Model . . . 19
4.1.2 View . . . 20
4.1.3 Controller . . . 20
4.2 Implementace . . . 21
4.2.1 Základní třídy . . . 21
4.2.2 Implementace modelu . . . 23
4.2.3 Implementace controlleru . . . 24
4.2.4 Balíček pro export do XML . . . 28
4.3 Testování . . . 31
4.4 Modelová studie . . . 31
4.4.1 Vytvoření modelové studie . . . 32

4.4.2 Ukázka modelové studie 32
4.5 Shrnutí implementace a možnost budoucího vývoje 34

5 Závěr **35**

A Export diagramu modelové studie **37**

Obsah přiloženého média **45**

Seznam obrázků

3.1 Ilustrace popisu webové služby na abstraktní (*Logical*) a konkrétní (*Psychical*) úroveň podle jazyka WSDL. [8] 6
3.2 Plan Do Check Act (PDCA) – životní cyklus business procesu. [21] 10
3.3 Základní prvky jazyka BPMN. [8] 11
3.4 Základní typy událostí. (Obrázek autora) 12
3.5 Základní typy bran pro větvení procesu. (Obrázek autora) 13
3.6 Podpora přenositelnosti BPMN diagramů napříč nástroji. [32] 16
3.7 Příklad BPMS nástroje – Camunda. [34] 16

4.1 Okno OpenPonku pro vytváření diagramů Petriho sítí. (Obrázek autora) 20
4.2 Struktura balíčků rozšíření pro BPMN. (Obrázek autora) 21
4.3 Struktura nejdůležitějších tříd a jejich rozdělení do balíčků. (Obrázek autora) . . 22
4.4 Vytvořený prvek aktivity s formulářem a paleta prvků pro BPMN. (Obrázek autora) 24
4.5 Struktura tříd controllerů. (Obrázek autora) 25
4.6 Vytvořené testovací třídy včetně výsledků testů – zelené úspěšné. (Obrázek autora) 31
4.7 Diagram procesu modelové studie. (Obrázek autora) 32
4.8 Průběh instance procesu modelové studie – stav po první službě. (Obrázek autora) 33
4.9 Průběh instance procesu modelové studie – stav po druhé službě. (Obrázek autora) 33
4.10 Paleta všech prvků v rozšíření platformy OpenPonk pro jazyk BPMN. (Obrázek autora) 34

Seznam tabulek

4.1 Implementovaná syntaktická pravidla 26

Seznam výpisů kódu

3.1 Ukázka XSD formátu pro ukládání BPMN diagramu. [26] 15
4.1 Ukázka vytvoření palety a dekodování obrázku v base64. 26
4.2 Ukázka vytvoření tvaru pro *Start Event* 27
4.3 Ukázka vytvoření jednoduchého XML v jazyce Pharo 29
4.4 Ukázka vytvoření diagramové části, včetně přístupu k souřadnicím prvku 30

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 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 10. května 2021 .....

Abstrakt

Platforma OpenPonk je nástroj pro konceptuální modelování vyvíjený v systému Pharo. Cílem této práce je rozšíření platformy o prototyp modeláře pro BPMN modelování. Dalším cílem je zajistit potřebná rozšíření pro orchestraci webových služeb pomocí jazyka BPMN, v této práci je toho docíleno vytvořením exportu do spustitelného XML formátu. Nakonec je výsledek demonstrován na modelové studii. Přínosem této práce je nezávislost konceptuálního modelu orchestrace na implementaci procesního stroje.

Klíčová slova modelovací plugin, orchestrace webových služeb, webové služby, BPMN, XSD, OpenPonk, Pharo

Abstract

The OpenPonk platform is a conceptual modeling tool developed in the Pharo system. The aim of this work is to extend the platform with a prototype modeler for BPMN modeling. Another goal is to provide the necessary extensions for the orchestration of web services using the BPMN language, in this work this is achieved by creating an export to an executable XML format. Finally, the result is demonstrated in a model study. The contribution of this work is the independence of the conceptual model of orchestration on the implementation of the process engine.

Keywords modeling plugin, web service orchestration, web services, BPMN, XSD, OpenPonk, Pharo

Seznam zkratek

BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
BPMS	Business Process Management Suite
MVC	Model-View-Controller
OMG	Object Management Group
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Kapitola 1
Úvod

I menší podniky dnes disponují složitými informačními systémy, které se často skládají z několika webových služeb a dalších komponent. S rostoucím počtem služeb přichází nutnost vzájemného propojování do funkčních celků. Pouhým statickým propojováním služeb často nejsme schopni využít jejich potenciál. Proto je zapotřebí spojovat služby dynamicky, dle aktuálních potřeb uživatelů. V současnosti je tento proces znám jako orchestrace webových služeb.

Pro větší přehlednost těchto systémů se používá několik modelovacích notací, jednou z nich je jazyk BPMN. Jazyk, který vznikl pro modelování podnikových procesů, se díky několika spustitelným formátům dá využít nejen k modelování, ale také k orchestraci webových služeb či podnikových systémů. Svůj význam dostávají spustitelné formáty jazyka BPMN také ve světě *Business Process Managementu*, ve kterém dochází k propojování teoretické analýzy a technické implementace pomocí automatizovaných nástrojů.

Platforma OpenPonk, která vznikla jako studentský týmový projekt na Katedře softwarového inženýrství FIT ČVUT a nyní je vyvíjená *Centrem pro konceptuální modelování a implementaci*, se zaměřuje právě na konceptuální modelování. Obsahuje podporu pro několik notací včetně UML, či Petriho sítí. Neobsahuje však možnost modelování podnikových procesů v BPMN. Účelem této bakalářské práce je rozšířit platformu OpenPonk o prototyp modeláře pro jazyk BPMN. Dále je také cílem zaměřit se na exportovatelnost procesních diagramů a jejich využití k orchestraci webových služeb.

Teoretická část práce (viz kapitola 3) se zabývá analýzou webových služeb, používaných technologií a možností propojování webových služeb v souvislosti s jazykem BPMN. Praktická část (viz kapitola 4) se zaměřuje na vývoj modeláře pro jazyk BPMN včetně možnosti exportu diagramu do spustitelného formátu. Závěrem práce je na modelové studii testováno využití jazyka BPMN k orchestraci webových služeb.

Tato práce navazuje na bakalářskou práci *Podpora standardu OpenPonk* [1] studenta Borise Anisimova, který položil základy modelovacího nástroje pro jazyk BPMN.

Kapitola 2
Cíle práce

Cílem této práce je vytvoření modulu pro platformu OpenPonk pro modelování orchestrace webových služeb pomocí jazyka BPMN a demonstrace jeho použití na ukázkové studii. V implementaci je rozšířena stávající platforma OpenPonk, vyvíjená *Centrem pro konceptuální modelování a implementaci*.

Teoretická část práce se zaměřuje na téma webových služeb a jejich orchestraci. Cílem této části práce je analyzovat používané technologie a nástroje a jejich využití ve spolupráci s notací BPMN. Dalším cílem teoretické části je seznámit se s platformou OpenPonk, její architekturou a možností rozšíření pro modelování.

Praktická část se zabývá návrhem architektury a implementací modeláře pro BPMN. Dalším cílem práce je implementace možnosti exportování modelu ve formátu vhodném k jeho dalšímu zpracování a následnému použití k orchestraci webových služeb. Nakonec je třeba implementaci otestovat a demonstrovat výsledek práce na ukázkovém projektu s několika webovými službami.

Přínosem této práce je rozvoj platformy OpenPonk, která díky tomuto rozšíření může zaujmout více uživatelů, a to nejen z oblasti konceptuálního modelování, na které je platforma primárně zaměřena. Tato práce cílí na webové inženýry, kterým může OpenPonk pomoci svou uživatelskou přívětivostí a možností uživatelské úpravy platformy. Například mohou jeden model exportovat do spustitelného formátu pro více různých vzájemně nekompatibilních procesních strojů, a to díky možnosti přímé úpravy zdrojového kódu.

Kapitola 3
Rešeršní část

Tato část bakalářské práce se zabývá základními pojmy a nastíní další znalosti potřebné k orchestraci webových služeb. Zaměří se především na jazyk BPMN, jeho napojení na svět webových služeb, použití a převod do spustitelného formátu. Na závěr bude představena platforma OpenPonk, její vývoj v objektovém jazyce Pharo, struktura a rozšiřitelnost.

3.1 Webová služba

„Webová služba je softwarový systém navržený pro interakci dvou strojů po síti. Má rozhraní popsané v strojově zpracovatelném formátu, konkrétně WSDL (Web Services Description Language). Ostatní systémy interagují s webovou službou způsobem předepsaným její definicí zprávou SOAP (Simple Object Access Protocol), typicky přepravených pomocí HTTP s XML serializací ve spojení s jinými webovými standardy.“ [2] (Překlad autora).

Webové služby jsou technologií pro vzdálené volání funkcí, tím se stávají základním stavebním blokem pro přechod k distribuované výpočetní technice. Webová služba musí být implementována konkrétním agentem. Agent odesílá a přijímá zprávy, zatímco služba je jen poskytovanou abstraktní sadou funkcí. Agent může být pokaždé napsán v jiném programovacím jazyce, zatímco služba a její rozhraní zůstávají stejné. Takto na sebe může být navázáno více různých služeb, které komunikují nejen s uživatelem, ale také mezi sebou. [3]

Oba protokoly SOAP a WSDL jsou provedeny v syntaxi jazyka XML, tak aby byly snadno strojově zpracovatelné a co nejméně závislé na zvolené verzi standardu XML. Touto nezávislostí vhodně zapadají do koncepce SOA (Service Oriented Architecture). [2]

3.1.1 XML

XML neboli Extensible Markup Language je jednoduchý značkovací jazyk, jehož základem je stromovitá struktura dokumentu s právě jedním kořenem. Jazyk byl vyvinut konsorciem W3C k umožnění serializace dat. XML se stalo velmi rozšířeným formátem zejména díky jednoduché syntaxi, snadné přenositelnosti a konverzi. [4]

3.1.2 SOAP

Simple Object Access Protocol je protokol pro výměnu zpráv založených na formátu XML. Vznikl v roce 1999 díky spolupráci firem DevelopMentor, Microsoft a UserLand. Dnes je SOAP specifikace spravována skupinou tvořící internetové protokoly z W3C konsorcia. [5]

Formát SOAP poskytuje prostředí pro tvorbu komunikace mezi webovými službami. Funguje na principu peer-to-peer, tedy jednotlivé služby spolu komunikují pomocí jednosměrných XML zpráv v modelu požadavek/odpověď.

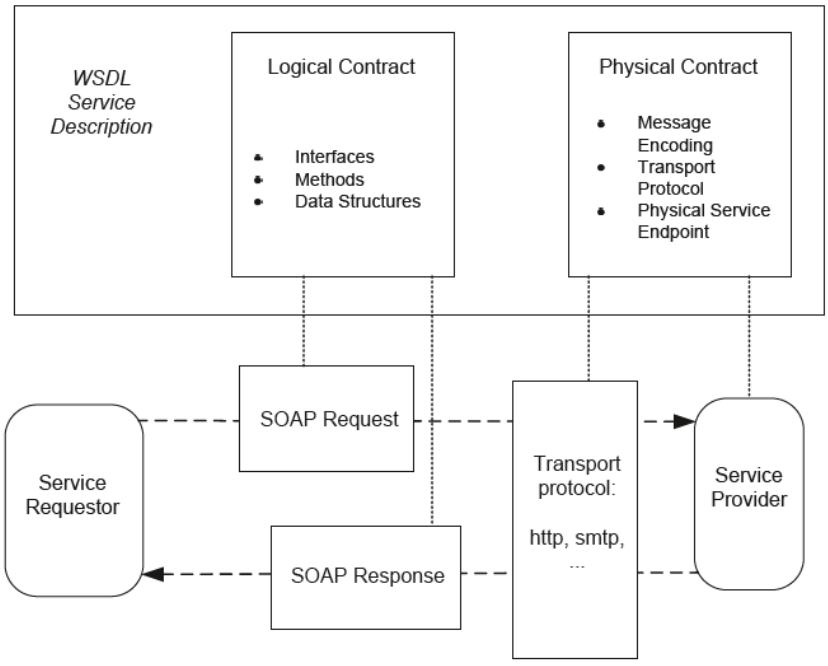
Jedna aplikace pošle prostřednictvím XML zprávy požadavek druhé aplikaci, ta požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká na požadavky klientů do okamžiku, kdy přes HTTP přijde SOAP zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď. [6]

Díky své jednoduchosti, rychlosti a použití HTTP pro přenos požadavků se stal SOAP velmi oblíbený. Infrastruktura webu a většiny společností dovoluje neomezenou komunikaci přes protokol HTTP (port 80), který SOAP využívá. Není tedy nutné složité navazování spojení ani přechod přes firewall. [5] [6]

3.1.3 WSDL

„Web Services Description Language Verze 2.0 (WSDL 2.0) poskytuje model a formát XML pro popis rozhraní webových služeb. WSDL 2.0 umožňuje oddělit popis abstraktních funkcí nabízených službou od konkrétních technických podrobností popisu služby, například „jak“ a „kde“ se tato funkce nabízí.“ [7] (Přeloženo autorem).

WSDL 2.0 popisuje webovou službu ve dvou úrovních – abstraktní (*Logical*) a konkrétní (*Psychical*), viz obrázek 3.1. Na abstraktní úrovni popisuje webovou službu jako službu, která odesílá a přijímá zprávy. Tyto zprávy jsou popsané rozhraní ve schématu XML dokumentu. Rozhraní seskupuje operace bez jakéhokoliv závazku k přenosu, či formě. Na konkrétní úrovni je popisována výměna zpráv a jejich kódování, včetně přidružených síťových adres odesílatele a příjemce. [8]



Obrázek 3.1 Ilustrace popisu webové služby na abstraktní (*Logical*) a konkrétní (*Psychical*) úroveň podle jazyka WSDL. [8]

3.2 Servisně orientovaná architektura

SOA (Service Oriented Architecture), neboli architektura orientovaná na služby, poskytuje odpovědi na problémy identifikované ve stávající softwarové architektuře. SOA umožňuje vývoj aplikací, které jsou pružnější a přizpůsobivější než aplikace vytvořené pomocí tradičních architektur. Aplikace založené na servisně orientované architektuře se proto mnohem snáze upravují a přizpůsobují. [9]

SOA není konkrétní technologie. Jedná se spíše o přístup k organizaci IT zdrojů, který má za cíl maximální zvýšení flexibility managementu podniku. SOA vnímá jednotlivé zdroje jako samostatné nezávislé komponenty, které spolu komunikují prostřednictvím služeb. Tyto komponenty v rámci SOA vytvářejí provázané business procesy, které zabezpečují šíření informací. SOA minimalizuje propast mezi modely obchodních procesů a skutečným aplikačním softwarem. Toho dosahuje zavedením nových technologií a jazyků, především jazyků BPEL a BPMN. Při návrhu je důležitá nezávislost business procesů na platformě, protože jen tak je možné dosáhnout požadované flexibility. [10]

Tam, kde se dodnes používají desktopové aplikace, je to většinou na určité platformě a napsané v určitém jazyce. Jelikož je celý systém provozován na jedné platformě, je jedinou cestou pro komunikaci API, na které se dají napojovat další systémy, či komponenty. To ale mnohdy v současné době nestačí. Dynamicky se mění požadavky na vyvíjené systémy a jejich funkčnost. Rok co rok se objevují nové technologie. Proto se přistupuje k přenositelnosti a flexibilitě. [11]

Aby se dnes systémy uplatnili na trhu, musí komunikovat i mezi různými verzemi, platformami apod., nestačí tedy definovat API jedné technologie. Ani vývojáři se dnes nespecializují na jednu jedinou technologii. Důvodem těchto změn je bezesporu snaha o kompletní pokrytí business procesů informačními systémy, čehož je dosahováno jen kombinací několika technologií. Tyto aspekty a některé další vedly k prudkému rozvoji SOA přístupu, kdy se hranice systému definují prostřednictvím specifikovaného formátu předávaných dat, které daná služba produkuje či přijímá.

V současnosti v drtivé většině případů ke specifikaci přenášených dat dochází prostřednictvím formátu XML, jelikož je velmi dobře strojově čitelný. Dalším důležitým krokem k rozšíření využití SOA bylo zavedení popisu existujících služeb. Každou již existující službu lze spouštět na serveru pomocí strojem čitelného XML kódu jazyka WSDL. [11]

3.3 Orchestrace webových služeb

Standardní technologie jako např. WSDL a SOAP, pracující s webovými službami popisují jednotlivá rozhraní atomických služeb, jejich lokalizaci a spuštění.

Co však tyto základní technologie neposkytují, jsou důležité detaily, které popisují chování služby jako součást více komplexní spolupráce. Když se jedná o spolupráci, která je kolekcí aktivit (metod, služeb) navržených tak, aby úspěšně plnila daný cíl, jedná se o tzv. business proces. A právě popis kolekcí aktivit, který tento business proces vytváří je nazýván orchestrace. [12]

3.3.1 Standardy k orchestraci webových služeb

Existuje několik standardů souvisejících s orchestrací webových služeb. Většinou jsou na bázi formátu XML a fungují jako standardní webová služba, která zprostředkovává a deleguje chování ostatních služeb. Tento koncept umožňuje i vzájemné asynchronní volání služeb, kterého není možné dosáhnout bez orchestrační komponenty. [13]

3.3.1.1 WS-BPEL

Web Services Business Process Execution Language, často označován pouze jako BPEL, je OASIS standard určený pro automatizaci business procesů. BPEL je založen na jazyce XML, definuje model a prostředky pro popis chování procesu a umožňuje vykonávání jednotlivých procesů, které se skládají z volání webových služeb. Jedná se o nejrozšířenější standard užívaný k orchestraci webových služeb. Jeho vznik ale nebyl vůbec snadný. IBM a Microsoft totiž definovaly vlastní jazyky (WSFL a Xlang) a teprve příchod dalších modelovacích jazyků v roce 2003 donutil velké společnosti ke vzniku společného jazyka – BPEL4WS. Tento standard byl svěřen konsorciu OASIS, které jazyk dále vyvíjelo až do dnešního WS-BPEL 2.0. [14]

BPEL pokrývá základní principy strukturovaného programování (podmínky, cykly, ošetření výjimek apod.). Nedostatkem jazyka BPEL je absence grafické reprezentace, ta je často nahrazována notací BPMN, nicméně mapování mezi těmito dvěma jazyky není jednoznačné. [15]

3.3.1.2 OWL-S

OWL-S je ontologie postavená na vrcholu Web Ontology Language, určená k popisu sémantických webových služeb. Umožňuje za určitých omezení uživatelům a softwarovým agentům automaticky objevovat, vyvolávat, skládat a monitorovat webové zdroje nabízející služby. [16]

3.3.1.3 WSCI

WSCI (Web Service Choreography Interface) je jazyk založený na XML, který popisuje pozorovatelné chování webové služby, neřeší však definici a implementaci interních procesů, které ve skutečnosti řídí výměnu zpráv. To zajišťuje jazyk WDSL, se kterým je WSCI využíván. Cílem WSCI je spíše popsat pozorovatelné chování webové služby pomocí rozhraní orientovaného na tok zpráv. Tento popis umožňuje vývojářům, architektům a nástrojům popsat a sestavit globální pohled na dynamiku výměny zpráv díky pochopení interakcí s webovou službou. [17]

3.3.1.4 Wf-XML

Wf-XML je navrženo jako rozšíření protokolu ASAP (Asynchronous Service Access Protocol). ASAP poskytuje základní možnosti pro kontrolu a monitorování asynchronních webových služeb prostřednictvím SOAP. WF-XML kontroluje operace jako: vytvoření instance služby, její nastavení, spuštění, zastavení a vzniklé chyby. V případě monitorování se jedná o sledování aktuálního stavu dané služby a historii jejího spouštění. Speciálním případem asynchronních služeb jsou obchodní procesy, tedy úroveň procesních strojů, tzv. *process engines*. [12]

3.3.1.5 BPMN

Jazyk BPMN není typickým představitelem technologie pro orchestraci webových služeb, i zde však našel své místo a je využíván dokonce několika způsoby.

Jako grafická reprezentace jazyka WS-BPEL je využívána pouze podmnožina BPMN, navíc specifikace převodu BPMN na WS-BPEL a naopak není zcela jednoznačná. Toho využívají velké společnosti, které si specifikaci upravují dle svých potřeb. Proto se může stát, že soubory formátu WS-BPEL generované z BPMN modelu nejsou napříč různými nástroji kompatibilní.

Využití BPMN jako vlastní orchestrační komponenty umožňují opět jen některé nástroje bez zaručení vzájemné kompatibility. Tyto nástroje jako např. *Oracle BPM Suite*, implementují vlastní process engine, jehož reprezentace není veřejně známa. Již zmíněný nástroj *Oracle BPM Suite* dokonce umožňuje přímé spuštění BPMN diagramu stylem „What you see is what you execute“ (WYSIWYE), volně přeloženo: „Co je vidět, to se spouští“. Jako interní reprezentace spouštěcího procesu je využita notace XSD založená na principech BPEL rozšířená o prvky specifické pro daný nástroj. [13] [18]

Samotné BPMN 2.0 specifikuje dva exportní formáty, které jsou přenositelné mezi platformami, ne však přímo spustitelné. Jedná se o formáty XSD a XMI. Oba formáty je možné nalézt v souborech s příponou *.bpmn*. [19]

Více o Exportních formátech a notaci BPMN bude popsáno v kapitole 3.5.

3.4 Podnikový proces

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. [8]

Každý jsme účastníci nejrůznějších procesů. Procesy existují bez ohledu na to, jak dobře jsou řízeny. Aby obchodní společnost dosáhla svých cílů, musí své procesy řídit tak, aby byly vykonávány pokud možno efektivně a účelně. Společnosti tedy mají snahu své procesy zlepšovat a optimalizovat. Společnost je tak efektivní, jak efektivní jsou její procesy. Jejich přesný popis je základem úspěchu, každý zaměstnanec má pak přesně dané činnosti a dochází k méně problémům. [20]

Komplexně definovaný business proces se zabývá otázkami jako: co, kde, kdy, proč a jak se práce provádí, a kdo je za její provádění zodpovědný. Dobře strukturovaná definice procesu poskytne správné množství přehlednosti a podrobností různým spotřebitelům těchto informací, potenciálně napříč všemi úrovněmi organizace. [21]

Existuje několik způsobů, jak business proces graficky zachytit. Jazyk BPMN byl k tomuto účelu vyvinut, někdy je však třeba zachytit procesy z jiného pohledu, proto jsou hojně používané i následující notace: ARIS, UML, IDEF, BSP, DEMO či Petriho sítě.

Dobře zmapovaný proces se dá díky moderním nástrojům dobře automatizovat (BPMS - Business Process Management Suite). Zjednodušeně řečeno lze modely procesů spouštět na serverech, či podobných prostředích a díky nim koordinovat přenos dat a chod ostatních aplikací, komponent, ale i činnost zaměstnanců. [20]

3.4.1 Řízení podnikových procesů

„Business Process Management, neboli řízení podnikových procesů, či procesní řízení, je založeno na pozorování každého produktu, který společnost poskytuje na trh. To je výsledkem řady provedených činností. Podnikové procesy jsou klíčovým nástrojem pro organizaci těchto činností a pro lepší pochopení jejich vzájemných vztahů.“ [8] (Překlad autora). V praxi dochází ke střetu dvou různých pohledů na tuto definici.

Na jedné straně jsou manažeři, kteří na procesní řízení pohlížejí jako na manažerskou disciplínu. Jedná se o soubor metod a nástrojů, které intuitivně zlepšují obchodní procesy. Tento přístup se zaměřuje zejména na části společnosti podílející se na tvorbě byznysu, od cílů, přes procesy, až po samotné zdroje. Zvláštní pozornost je kladena na samotné procesy, které by měly probíhat souvisle, a neměly by být trhány ani organizační strukturou společnosti. Častým důsledkem vlivu procesního řízení jsou právě změny organizační struktury a podpora týmové práce, která musí být provázena změnou firemní kultury s důrazem na zodpovědnost jednotlivce a managementu a sdílení znalostí. [22]

Na druhé straně jsou IT odborníci, kteří procesní řízení vnímají jako soubor technologií. Tyto technologie začaly vznikat v 90. letech minulého století s nástupem osobních počítačů. První generace BPM nástrojů podporovala jen modelování. Postupně přibývaly používané technlologie a vznikly workflow systémy 2. generace. Tyto systémy vznikající po roce 2005, dokáží uceleně podporovat celý životní cyklus podnikání a živě reagovat na změny. Soubor těchto technologií je často nazýván pojmem Business Process Management Suite (BPMS). [23]

Organizace s vyspělými schopnostmi BPM spravují své procesy v cyklech. Opakuje se dokola několik fází, které řeší plánování, návrh, implementaci, provádění, měření, kontrolu a neustálé zlepšování podnikových procesů. Nejčastěji je životní cyklus procesu rozdělen na čtyři fáze: Plan, Do, Check, Act (PDCA), které v padesátých letech minulého století poprvé definoval americký statistik Dr. W. Edwards Deming. Do češtiny je možné tyto fáze přeložit jako: naplánuj, proveď, ověř, jednej. Názorně je jednoduchost principu životního cyklu procesu znázorněna na obrázku 3.2. [21]

Obrázek 3.2 Plan Do Check Act (PDCA) – životní cyklus business procesu. [21]

3.4.2 BPMS

Zkratka BPMS vznikla z pojmu Business Process Maagement System druhé generace (BPMS 2.0). V současné době se tyto systémy častěji označují jako Business Process Management Suite. Pod tímto označením se však neskrývá jen jedna konkrétní implementace nebo technologie, ale soubor konceptů, nástrojů a standardů, které dokáží uceleně podporovat celý životní cyklus podnikání. Základním konceptem, na kterém je BPMS postavené, je koncept SOA, jehož principy byly stručně popsané v předchozí kapitole 3.2. Významnou roli hrají webové služby a podniková sběrnice služeb ESB (Enterprise Service Bus). Díky jejich vlastnostem jako jsou nezávislost na platformě, zapouzdřenost a přehledná dokumentace jedním popisem, která definuje možnosti jejich použití. [22]

BPMS jsou nástrojem procesní integrace, proto jsou podporovány z obou stran – z vrstvy managementu i z vrstvy IT expertů. Vrstva managementu potřebuje výkonné prostředky pro řízení změn podnikání. A současně vrstva IT expertů, kteří hledají účinné prostředky pro aplikační a systémovou integraci, a zároveň hledají prostředky pro znovupoužití jednou vytvořené implementace. Hlavními moduly BPMS dle [23] jsou:

- **Procesní engine** je aplikační prostředí, které interpretuje chování procesů, toto prostředí je popisované v jazyce BPEL.
- **Business Rules Engine** je schopen pro uložená pravidla dodat rozhodovací data a výsledek poskytuje jak pro řízení procesů, tak pro služby, které je vyžadují.
- **Workflow** zajišťuje lidskou interakci s procesy.
- **Business Activity Monitoring** zajišťuje sledování výkonnosti procesů v reálném čase.

3.5 BPMN

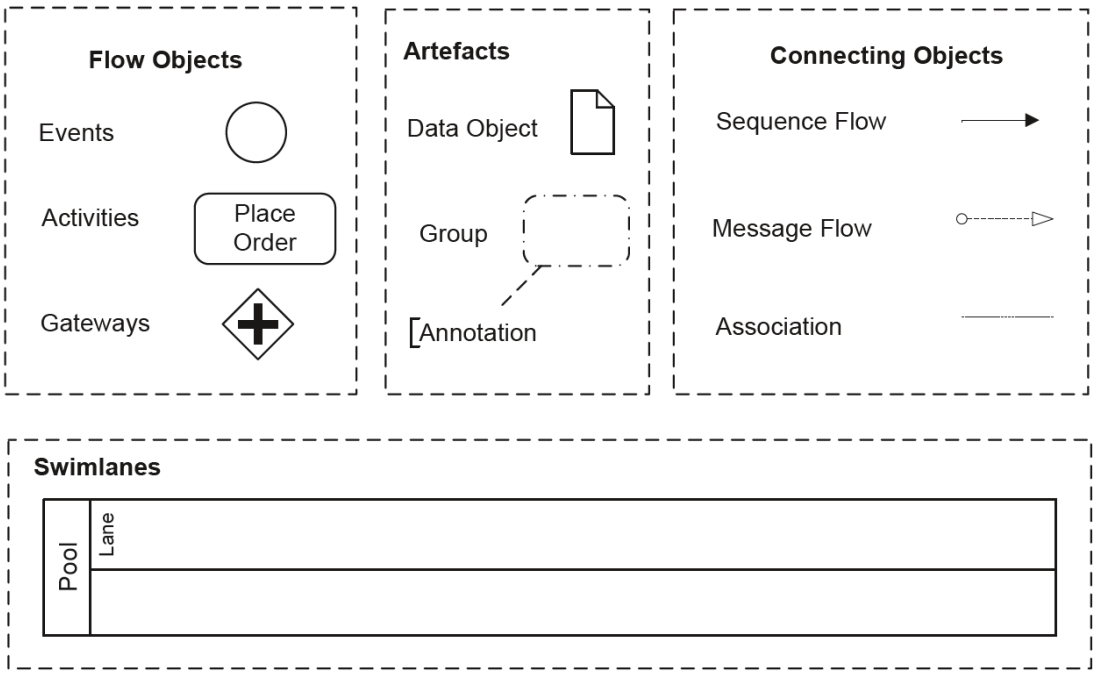
BPMN je standard zavedený pro grafickou reprezentaci business procesů. K dnešnímu názvu *Business Process Model and Notation* vedla dlouhá cesta, stejně jako v samotném vývoji jazyka BPMN. První požadavky na jazyk pro modelování business procesů vznikly v 90. letech, a tak bylo již v roce 2001 založeno konzorcium BPMI (Business Process Management Initiative). Hlavním cílem bylo vytvořit modelovací notaci čitelnou pro všechny účastníky životního cyklu procesu. V roce 2004 byla vydána první specifikace pod označením BPMN 1.0, tehdy zkratka BPMN znamenala Business Process Modeling Notation. [24]

V roce 2006 se BPMI spojilo s organizací OMG (Object Management Group). Práce na standardu pokračovaly a v roce 2007 byla vydána verze BPMN 1.1, která již byla plně podporována společností OMG. Nejvýznamnější posun byl zaznamenán s verzí BPMN 2.0, která byla představena v roce 2011 . Mimo nových grafických elementů jsou nově podporovány exportní a spustitelné formáty, jako je BPEL. Díky konverzím diagramu na spustitelné formáty se jazyk BPMN stal velkým hráčem v oblasti BPMS. [25] [24]

Aktuální verze BPMN 2.0.2 byla vydána v roce 2014, jejím cílem bylo upřesnit nejasnosti z verze 2.0, a to zejména v exportní části. [26]

3.5.1 Základní prvky jazyka BPMN

Základní prvky jazyka BPMN jsou definované standardem, kde každá implementace by měla využívat grafické prvky a tvary navržené v tomto standardu. Definované nejsou pouze barvy elementů, velikost textu a jeho umístění. V následujících kapitolách budou popsány prvky, které jsou podle Bruce Silvera [27] označeny jako Paleta level 1, tedy nejzásadnější prvky jazyka. Prvky jsou ve standardu rozděleny do čtyř přesně definovaných kategorií (viz obrázek 3.3) tak, aby bylo jejich použití a orientace v grafu co nejjednodušší: *Flow Objects* (tokové objekty), *Connecting Objects* (spojovací objekty), *Swimlanes* (plavecké dráhy), *Artifacts* (artefakty). [8]



Obrázek 3.3 Základní prvky jazyka BPMN. [8]

3.5.1.1 Tokové objekty

Tokové objekty jsou hlavními prvky jazyka BPMN. Definují kostru procesu a tok informací. Existují tři základní tokové elementy: *Activity*, *Event* a *Gateway*, každý z nich se dále dělí na několik dalších elementů, které již mají konkrétní význam a definují chování procesu. Konkrétní význam prvku je zachycen stylem čáry, která ohraničuje prvek a ikonou uvnitř elementu, která má za úkol dané prvky odlišit. Příkladem může být *Manual Activity*, která v sobě obsahuje ikonu v podobě lidské ruky. [24] [28]

Activity (činnost) je nejdůležitější součástí BPMN, protože každý proces je primárně tvořen z činností. Jedná se o jednotku práce, která posouvá „děj" životního cyklu procesu. *Activity* může být několika typů podle druhu činnosti, kterou vykonává. Nejzajímavější z pohledu webových služeb je *Service Activity*, která představuje činnost jedné webové služby.

Event (událost) se vyskytuje v každém diagramu. Účelem události je měnit stav procesu – začít, ukončit a pozastavit, viz obrázek 3.4. *Start Event* proces spouští, vyskytuje se proto jen na začátku procesu. Případná vnitřní ikona specifikuje, na základě jaké události je proces spouštěn. *Intermediate Event* pozastavuje proces do vzniku dané události, nejčastěji do přijetí zprávy, či uplynutí času, poté proces pokračuje. *End Event* ukončuje proces, proto je vždy posledním prvkem diagramu. [29] [30]

Obrázek 3.4 Základní typy událostí. (Obrázek autora)

Gateway (brána) definuje podmínku pro větvení nebo slučování toků procesu. *Gateway* zajišťuje větvení, jestliže obsahuje jeden vstupní tok a několik výstupních. Taková brána určuje jaké větve procesu se budou dále vykonávat. Ke slučování toků dochází, pokud brána obsahuje několik vstupních toků a jeden výstupní, v tom případě se podle typu *Gateway* čeká na dokončení jednoho, či více vstupních toků, a až poté proces pokračuje dále. Existuje pět typů *Gateway*: *Exclusive Gateway*, *Inclusive Gateway*, *Complex Gateway*, *Event-Based Gateway* a *Parallel Gateway*. Všechny typy *Gateway* jsou zobrazeny na obrázku 3.5.

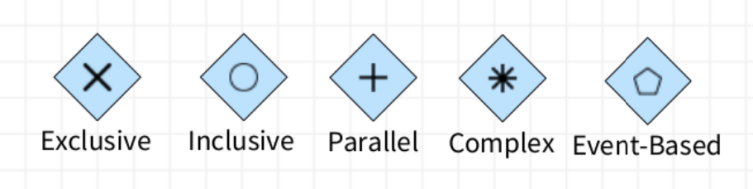
Exclusive Gateway představuje logický XOR, takže výstupem podmínky je vždy právě jedna větev.

Dalším typem je *Inclusive Gateway*, která imituje logický OR. Výstupem této brány může být více než jedna větev.

Podobným typem je *Complex Gateway*, ta je používána pro složitější větvení, proto vždy musí obsahovat výchozí cestu, kdyby ostatní podmínky nebyly splněny.

Event-Based Gateway, rozhoduje na základě vnější události. Příkladem může být čekání na příchozí zprávu po určitou dobu. Pokud zpráva nedorazí včas, je zvolena alternativní cesta.

Posledním typem je *Parallel Gateway*, tato brána neobsahuje podmínku, protože jsou vykonávány všechny její větve současně. Úskalím této *Gateway* je nutnost spojení běžících toků na jiném místě procesu, a s tím spojené čekání na provedení všech činností v jednotlivých větvích. Při špatném použití *Gateway* v návrhu procesu může při slučování toků nastat tzv. *Deadlock*, zablokování v jednom místě procesu bez možnosti pokračovat. [29] [28]



Obrázek 3.5 Základní typy bran pro větvení procesu. (Obrázek autora)

3.5.1.2 Spojovací objekty

Objekty v BPMN nejsou izolované, ale spojené tak, aby vytvořily tok. Základním typem je *Sequence Flow*, která spojuje tokové objekty. Jedná se o orientovanou spojnici dvou bodů tak, aby přesně udávala pořadí událostí v procesu. Existují syntaktická omezení, která definují základní pravidla vytváření toků. *Sequence Flow* nesmí vést do *Start Event* a nesmí vést z *End Event*. Další omezení zakazuje její použití mezi různými *Pool* (bazény). Ke komunikaci mezi bazény existuje *Message Flow*, která slouží pouze k tomuto účelu a nelze ji použít v rámci jednoho *Pool*. *Association*, neboli asociace, slouží k propojení artefaktů nebo textu s tokovými objekty. Asociace jako jediná nemusí být orientovaná, má pouze informativní charakter pro větší přehlednost diagramu. [24] [30]

3.5.1.3 Plavecké dráhy

Plavecké dráhy jsou grafický kontejner, který reprezentuje účastníky procesu. Dělí se na dva druhy *Pool* (bazén) a *Lane* (plavecká dráha). Účastníkem procesu může být konkrétní entita nebo role (např. plavec, plavčík). Bazén představuje takového účastníka tak, že ohraničuje tokové elementy – tu část procesu, která danému účastníkovi náleží. Pokud se uvnitř bazénu nachází několik drah, označuje bazén organizaci či oddělení, a dráhy označují (a ohraničují) konkrétní účastníky procesu. [30]

3.5.1.4 Artefakty

Artefakty doplňují model o upřesňující informace, a nemají tak vliv na samotný tok procesu. Mezi artefakty patří: *Data Object* (datový objekt), *Group* (skupina) a *Annotation* (anotace). [28]

3.5.2 Použití jazyka BPMN

Jak již bylo zmíněno v předchozích kapitolách, jazyk BPMN vznikl za účelem modelování business procesů, specifikace 2.0 jde však ještě dál, snaží se zachytit celý životní cyklus procesů a automatizovat jejich běh již od grafického návrhu. [25]

Na úvod notace BPMN poskytuje srozumitelné a snadno pochopitelné schéma procesu. To jej umožňuje navrhnout, nejčastěji v grafickém editoru, bez nutnosti programování. Navíc je možné modely vytvářet na různých úrovních abstrakce, nejprve začít se základními elementy a až poté postupně doplňovat implementační detaily. Jazyk také umožňuje nahradit část modelu dalším vnořeným diagramem, či úplně jinou reprezentací. Modely se tak snaží o co nejvěrnější zachycení reality. [26]

Častým důvodem návrhu obchodního procesu je optimalizace současného (fungujícího) procesu. Pomocí BPMN lze tento proces navrhnut a na teoretické rovině překreslit, či dokonce spustit v testovací prostředí tak, aby se stal co nejefektivnější.

Návrh procesu, většinou vytváří business analytik, který disponuje základními znalostmi notace. Model ale musí být natolik přehledný, aby jej mohli spravovat a hodnotit i ostatní zaměstnanci a obchodníci. [25]

Grafická reprezentace je podpořena vnitřní reprezentací, která má danou sémantiku. Interně je model BPMN 2.0 ukládán do strojově čitelného formátu XML (soubory .bpmn) či XMI. Formát dokumentu definuje specifikace, což umožňuje přenositelnost modelů mezi různými modelovacími nástroji, a také spustitelnost procesů, neboli *Process Execution*. [26]

3.5.3 Exportní formáty jazyka BPMN

Standard BPMN 2.0 definuje dva XML formáty pro ukládání procesů a zajišťuje jejich vzájemnou kompatibilitu pomocí XSLT (Extensible Stylesheet Language Transformations). Navíc standard podporuje export diagramu do formátů WS-BPEL a XPDL, které však nejsou určené k ukládání diagramů samotných. [26] [19]

Prvním je **XML Schema Definition** (XSD), tento formát umožňuje ukládání diagramu i sémantických pravidel, a to buď do jednoho společného souboru, nebo do dvou oddělených. Navíc je možné jej v některých nástrojích použít přímo jako *Executable file* (spustitelný soubor). Toto schéma nalezneme v souborech s příponou *.bpmn*. Díky svým výhodám se XSD stalo nejvíce používaným formátem pro ukládání BPMN procesů. Ukázka kódu 3.1 zobrazuje základní syntaxi dle BPMN 2.0 specifikace. [26]

Dle standardu nelze uložit do XSD nevalidní diagram. V praxi lze však nalézt odchylky od standardu, jednou z odchylek jsou právě nevalidní XSD soubory, kvůli umožnění průběžného ukládání v nástrojích. Validace bývá vynucena až při spouštění diagramu, či exportu do jiného formátu. [19]

Formátem **XML Metadata Interchange** (XMI) lze zachytit stejné informace jako pomocí XSD, proto je zajištěna jejich vzájemná kompatibilita. Jediným rozdílem je, že do XMI lze uložit i nekompletní, či nevalidní diagram. [19]

XML Process Definition Language (XPDL) vznikl už v roce 2002 a stal se exportním formátem pro BPMN 1.0. S nástupem BPMN 2.0 ztratilo XPDL svou pozici, stále je však podporováno většinou nástrojů. Pomocí XPDL je možné diagram procesu serializovat a zanést, včetně souřadnic prvků, do XML. Formát má dokonce vlastní grafickou reprezentaci, většinou se však pro zobrazení používá notace BPMN. [31]

Výpis kódu 3.1 Ukázka XSD formátu pro ukládání BPMN diagramu. [26]

```
<!--main.bpmn-->
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions
    xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
    targetNamespace="sample1.main" xmlns:main="sample1.main"
    xmlns:s1="sample1.semantic1">
<bpmn:import location="semantic1.bpmn" namespace="sample1.semantic1"
    importType="http://www.omg.org/spec/BPMN/20100524/MODEL" />
    <bpmn:import location="diagram1.bpmn" namespace="sample1.diagram1"
        importType="http://www.omg.org/spec/BPMN/20100524/MODEL" />
</bpmn:definitions>

<!--semantic1.bpmn-->
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions
    xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
    targetNamespace="sample1.semantic1"
    xmlns:s1="sample1.semantic1">
    <bpmn:process id="process1">
        <!-- content here -->
    </bpmn:process>
</bpmn:definitions>

<!--diagram1.bpmn-->
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/DI"
    targetNamespace="sample1.diagram1"
    xmlns:bpmndi="http://www.omg.org/spec/BPMNDI/1.0.0"
    xmlns:d1="sample1.diagram1" xmlns:s1="sample1.semantic1"
    xmlns:main="sample1.main">
    <bpmndi:BPMNDiagram scale="1.0" unit="Pixel">
    <bpmndi:BPMNPlane element="main:collaboration1">
        <!-- content here -->
    </bpmndi:BPMNPlane>
    </bpmndi:BPMNDiagram>
</bpmn:definitions>
```

3.5.4 Nástroje pro jazyk BPMN

Existují desítky nástrojů, které podporují notaci BPMN, ne všechny však podporují export diagramů do standardních XML formátů. Podporou nejen exportních formátu se zabývá skupina *BPMN Model Interchange Working Group (BPMN MIWG)*, která zveřejnila analýzu nástrojů, které podporují import a export BPMN diagramů ve formátu XSD, viz obrázek 3.6. Navíc je možné na jejich webu nový nástroj zaregistrovat a otestovat, zda vyhovuje BPMN 2.0 standardu. [32] Mezi rozšířené funkcionality nástrojů nepatří ani validace diagramů. Neexistuje nástroj, ve kterém by uživatel mohl měnit nastavení prvků, exportu, či dodefinovat validační pravidla.

Některé specializované nástroje umožňují dokonce spouštění samotných procesů, tedy generování tzv. *execution model*. Příkladem může být nástroj Camunda [33], tento nástroj nabízí plnou automatizaci procesů. Vytvořený BPMN model je možné nahrát na server, kde běžící instance Camundy dokáže orchestrovat všechny účastníky procesu od lidí, přes API, webové služby, až po umělou inteligenci, viz obrázek 3.7. Takto spuštěný proces běží zcela automaticky. Pouze když je třeba lidská interakce, je vytvořen „Human Task“.

BPMN Tools tested for Model Interchange

Search:

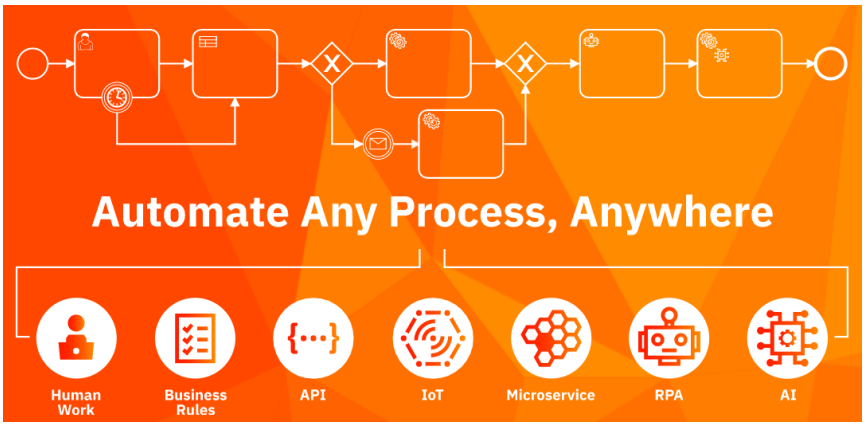
Tool Name & Version	BPMN 2.0	Import	Export	Roundtrip	Results Submitted	Part of Demos	License	Open Issues	Closed Issues
Yaoqiang BPMN Editor 2.2.18	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Open Source	1	5
camunda Modeler 2.4.0	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Open Source	8	5
Trisotech BPMN Web Modeler 4.1.8	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Commercial	0	1
W4 BPMN+ Composer V.9.0	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Commercial	0	0
Yaoqiang BPMN Editor 3.0.1	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Commercial	1	5
Signavio Process Editor 7.8.1	✓	✓ Diff	✓ Diff	✓	✓ Details	✓	Commercial	6	2
iGrafx Process 2013 for Six Sigma 15.0.4.1565	✓	✓ Diff	✓ Diff	✓	✓ Details	✗	Commercial	0	0
bpmn.io 0.0.1	✓	✓ Diff	✓ Diff	✓	◐ Details	✓	Open Source	0	0
ADONIS 5.1 UL5 (mfb_bpmn_1.1)	✓	✓ Diff	✓ Diff	✓	◐ Details	✓	Commercial	0	0
itp-commerce Process Modeler for MS Visio 6	✓	✓ Diff	✓ Diff	✓	◐ Details	✓	Commercial	0	0
Trisotech BPMN Modeler for MS Visio 4.0	✓	✓ Diff	✓ Diff	✓	◐ Details	✓	Commercial	0	0
Eclipse BPMN2 Modeler 0.2.6	✓	✓ Diff	✓ Diff	✓	◐ Details	✗	Open Source	15	0
IBM Process Designer 8.0.1	✓	✓ Diff	✓ Diff	✓	◐ Details	✗	Commercial	0	0
ARIS Architect 9.5.0	✓	✓ Diff	✓ Diff	✓	◐ Details	✗	Commercial	3	0
ARIS Business Architect 7.2.4	✓	✓ Diff	✓ Diff	✓	◐ Details	✗	Commercial	5	0
MID Innovator 11.5.2.30413	✓	✓ Diff	✓ Diff	✓	◐ Details	✗	Commercial	9	0
Oracle BPM Studio 12.1.3	✓	✓ Diff	✓ Diff	✗	◐ Details	✓	Commercial	0	0
IBM BlueWorks Live	✓	✗	✓	✗	✗	✗	Commercial	0	0
camunda-bpmn.js c906a7c941	✓	✓ Diff	✗	✗	✓	✓	Open Source	3	1
ARIS Express 2.4	✓	✗	✗	✗	✗	✗	Commercial	0	0
Bizagi Process Modeler 2.4	✓	✗	✗	✗	✗	✗	Commercial	0	0
Cameo Business Modeler 17.0.3 sp1	✓	✗	✗	✗	✗	✗	Commercial	1	0

Showing 1 to 22 of 22 entries

Obrázek 3.6 Podpora přenositelnosti BPMN diagramů napříč nástroji. [32]

Uživateli přijde upozornění na zpracování úkolu, po jeho vypracování a odeslání opět proces pokračuje. Samozřejmě nástroj Camunda dokáže měřit metriky efektivity procesu a za běhu procesy upravovat a validovat, včetně diagramů a hodnot. [34]

Takto komplexních nástrojů není mnoho, jelikož je nutné implementovat vlastní *process engine*, který běží na serveru. Tyto implementace jsou unikátní pro každý nástroj, proto jsou přísně chráněny jednotlivými společnostmi, mezi několik málo nástrojů k plné automatizaci BPMN procesů patří například – Camunda [33], jBPM [35], Flowable [36], Activiti [37], či Heflo [38].



Obrázek 3.7 Příklad BPMS nástroje – Camunda. [34]

3.6 OpenPonk

OpenPonk [39] je platforma pro konceptuální modelování implementovaná v jazyce Pharo. Platforma OpenPonk (původně pod názvem DynaCASE) začala vznikat v roce 2014 jako studentský týmový projekt. V současné době je nástroj rozšiřován a vyvíjen Centrem pro konceptuální modelování a implementaci na Katedře softwarového inženýrství FIT ČVUT. OpenPonk se zaměřuje na podporu modelování, simulace a generování kódu v oblasti *Business Engineering*. Hlavní předností platformy je snadná rozšiřitelnost a možnost vývoje uživatelských funkcionalit díky uveřejnění pod OpenSource licencí MIT. Nástroj v současné době (verze 2.0.2) podporuje notace UML, OntoUML, BORM, Petriho sítě a Konečné automaty. [40]

Platforma je založena na návrhovém vzoru MVC (Model-View-Controller). Tato třívrstvá architektura umožňuje snadnou rozšiřitelnost formou Pluginů. Výhodou tohoto systému je univerzálnost a možnost dát uživateli na výběr, jaké součásti chce, či nechce používat, a umožnit mu implementovat vlastní rozšíření. [41]

3.6.1 Pharo

Pharo je čistě objektově orientovaný jazyk založený na prostředí *Smalltalk*. Prostředí jazyka Pharo je v podobě virtuálního stroje, ve kterém lze programy spouštět, vyvíjet i ladit. Každá instance má podobu *Image* – obrazu paměti, ve které jsou ukládány veškeré zdrojové kódy a běžící procesy. Vlastní prostředí umožňuje programátorovi za běhu programu manipulovat s objekty a provádět dynamickou rekompilaci kódu. Takto je možné přímo upravit kterýkoliv prvek i samotné prostředí jazyka. Přestože jde o relativně mladý jazyk, podporovaný pouze malou komunitou, stalo se Pharo moderním, perspektivním a stabilním prostředím. [42]

Od roku 2017 je možné používat také verzovací nástroj Git, který je standardně používán při vývoji aplikací v běžně používaných jazycích. [42]

Kapitola 4
Praktická část

Praktická část této bakalářské práce se zabývá analýzou platformy OpenPonk, návrhem rozšíření pro BPMN a jeho implementací. Poslední část je věnována testování nově vzniklého nástroje a ukázce orchestrace webových služeb na modelové studii.

4.1 Analýza a návrh

Praktická část práce navazuje na bakalářskou práci studenta Borise Anisimova [1], který položil základy modulu pro jazyk BPMN v platformě OpenPonk. Jeho práce pro mě byla inspirací, ale nebylo možné využít jeho implementaci, která byla vázána na starší verzi platformy. Navíc byl zpracován jen malý rozsah několika základních prvků.

Platforma OpenPonk je postavená na návrhovém vzoru MVC, systém samotný je navržen tak, aby byl snadno rozšiřovatelný. Toho je docíleno pomocí rozsáhlého využití dědičnosti. Jádro OpenPonku nabízí balíčky obsahující základní třídy, které je možné dál rozšiřovat. Samotné funkcionality jádra jsou rozdělené tak, aby byla dodržena třívrstvá architektura.

Základem každého rozšíření je potomek třídy `OPPlugin`, tato třída pomocí svých metod identifikuje OpenPonku základní třídy z každé vrstvy architektury – `modelClass`, `layouterClass` a `diagramControllerClass`. Takto je vynuceno dodržení architektury a zároveň snadná integrace. Při inicializaci OpenPonk vyhledá všechny potomky třídy `OPPlugin`, tím zjistí jaká rozšíření má k dispozici.

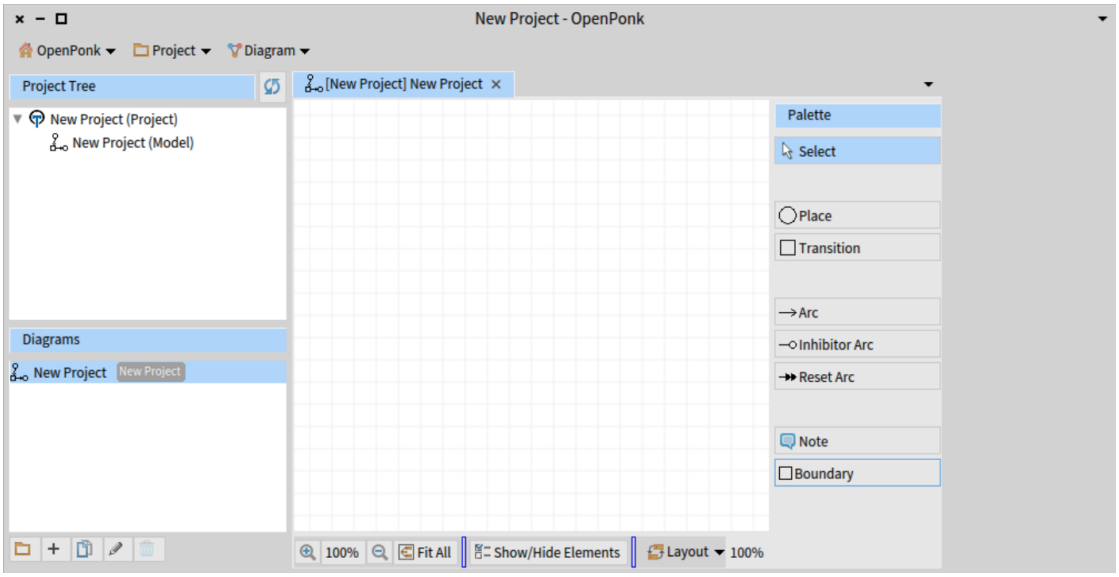
Rozšíření BPMN pro platformu OpenPonk bude v práci rozděleno do dvou balíčků. První z nich bude obsahovat rozšíření pro modelování pomocí BPMN, druhý balíček bude obsahovat třídy potřebné k exportu diagramu do formátu XML. Modelovací část implementace bude navazovat na existující rozhraní, které bude dále popsáno podle odpovídajících vrstev architektury Model-View-Controller.

4.1.1 Model

Model je oddělenou částí, každá třída modelu reprezentuje jeden prvek a jeho vlastnosti. Navíc je třeba jedna třída pro celkový model. OpenPonk nabízí základní rozhraní pro třídy modelu, tím je abstraktní třída `OPModelObject`. Ta zajišťuje základní vlastnosti prvku – jméno, ID, vlastníka a kolekci pro vnořené prvky `elements`. Zajímavostí je ID prvku, které je automaticky generované, tím je zajištěna jedinečnost v diagramu a lze se pomocí něj odkazovat při rekonstrukci diagramu ze souboru. Třída modelu jako taková nemá informace o existenci controllerů, či tvaru prvku.

4.1.2 View

Část View se stará o vykreslování elementů. Toho je docíleno pomocí rozšíření *Roassal* a *Trachel* [43], které jsou součástí Pharo. Při inicializaci nového projektu je vytvořeno nové okno obsahující plátno pro kreslení diagramu a paleta prvků. To je zobrazeno na obrázku 4.1, na kterém je vidět okno pro vytváření diagramů Petriho sítí. V levé části obrázku je zobrazen strom projektu, ve kterém je možné zobrazovat informace o vytvořených prvcích. Horní a dolní lišta obsahují základní ikony pro zobrazování, ukládání a otevírání projektu. [41]



Obrázek 4.1 Okno OpenPonku pro vytváření diagramů Petriho sítí. (Obrázek autora)

O okno diagramu se starají třídy `OPEditor` a `OPWorkbench`, ty k vytvoření okna využívají základní třídy Phara – `SpecLayout`. Naopak plátno (vykreslovací plocha) je instancí rozšíření *Roassal* a *Trachel*, kde jsou obě tyto rozšíření využívány najednou, protože každé nabízí jiné možnosti. Například *Trachel* podporuje akce pro kliknutí myší, *Roassal* zase pokročilé vizualizace.

Umístění prvku na plátno funguje velice jednoduše, uživatel si kliknutím na paletě prvků vybere prvek a dalším kliknutím jej umístí na plátno. Tento způsob je velice intuitivní, navíc jde prvky i později přesouvat, či některé zvětšovat. O vytvoření a zobrazení jednotlivých prvků se stará třída `OPCreationTool`, která pomocí tříd controlleru vykreslí daný prvek a uloží si jej do kolekce.

V rozšířeních není nutné tuto vrstu přímo implementovat, protože jádro OpenPonku obsahuje všechny metody k vykreslování prvků. Jediné co je třeba dodat, jsou definice plátna a palety prvků. Kliknutím na prvek palety se již volá nově vytvořený controller.

4.1.3 Controller

Obrazem modelu jsou controllery, které propojují vrstu view s modelem. Každé třídě modelu tak odpovídá jeden controller, který je zodpovědný za tvar a vykreslení prvku. [1]

Hlavní třídou v této vrstvě je třída `OPDiagramController`, která se stará o správné zobrazování všech prvků, a to nejen při jejich vytvoření, ale po celou dobu jejich existence. Třída `OPDiagramController` umožňuje také mazání prvků, skrývání a jejich pohyb. Dalo by se říci, že deleguje ostatní controllery. Její metoda `initializePalette:` navíc vytváří paletu prvků na pravé straně okna.

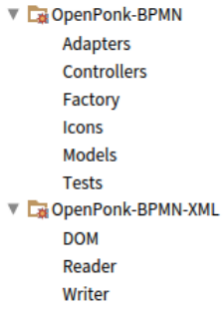
Každému modelu náleží jeden Controller, definuje totiž tvar prvku a některé další vlastnosti. OpenPonk nabízí abstraktní třídu, která definuje základní chování controlleru, tou třídou je `OPElementController`. Tvar prvku je vytvářen metodou `renderFigureIn:`, tuto metodu je samozřejmě nutné implementovat až v konkrétní podtřídě. Controller také sám vytváří instanci své modelové třídy, ke které slouží metoda `createModelIn:`.

4.2 Implementace

Implementační část je rozdělena do dvou částí. První je balíček `OpenPonk-BPMN`, který obsahuje všechny části potřebné k modelování BPMN diagramů. Druhý balíček `OpenPonk-BPMN-XML` je rozšířením modelovací části o export diagramů do XML formátu XSD, který je definován specifikací jazyka BPMN 2.0.

Struktura balíčků je dále pro přehlednost rozdělena pod jednotlivé tagy (štítky). Struktura rozšíření je podrobně znázorněna na obrázku 4.2. Nejdůležitější třídy rozšíření a jejich rozdělení do balíčků jsou znázorněny na diagramu 4.3.

Pro všechny názvy tříd v rozšíření je v souladu s architekturou platformy použita předpona „OPBPMN“. Pomocí předpony v názvech třídy je dosaženo nejen přehlednosti kódu, ale v jazyce Pharo je nutné, aby každá třída měla unikátní název napříč celým prostředím. K ukládání práce byl použit verzovací nástroj Git, který je přímo ve Pharo plně podporován.



Obrázek 4.2 Struktura balíčků rozšíření pro BPMN. (Obrázek autora)

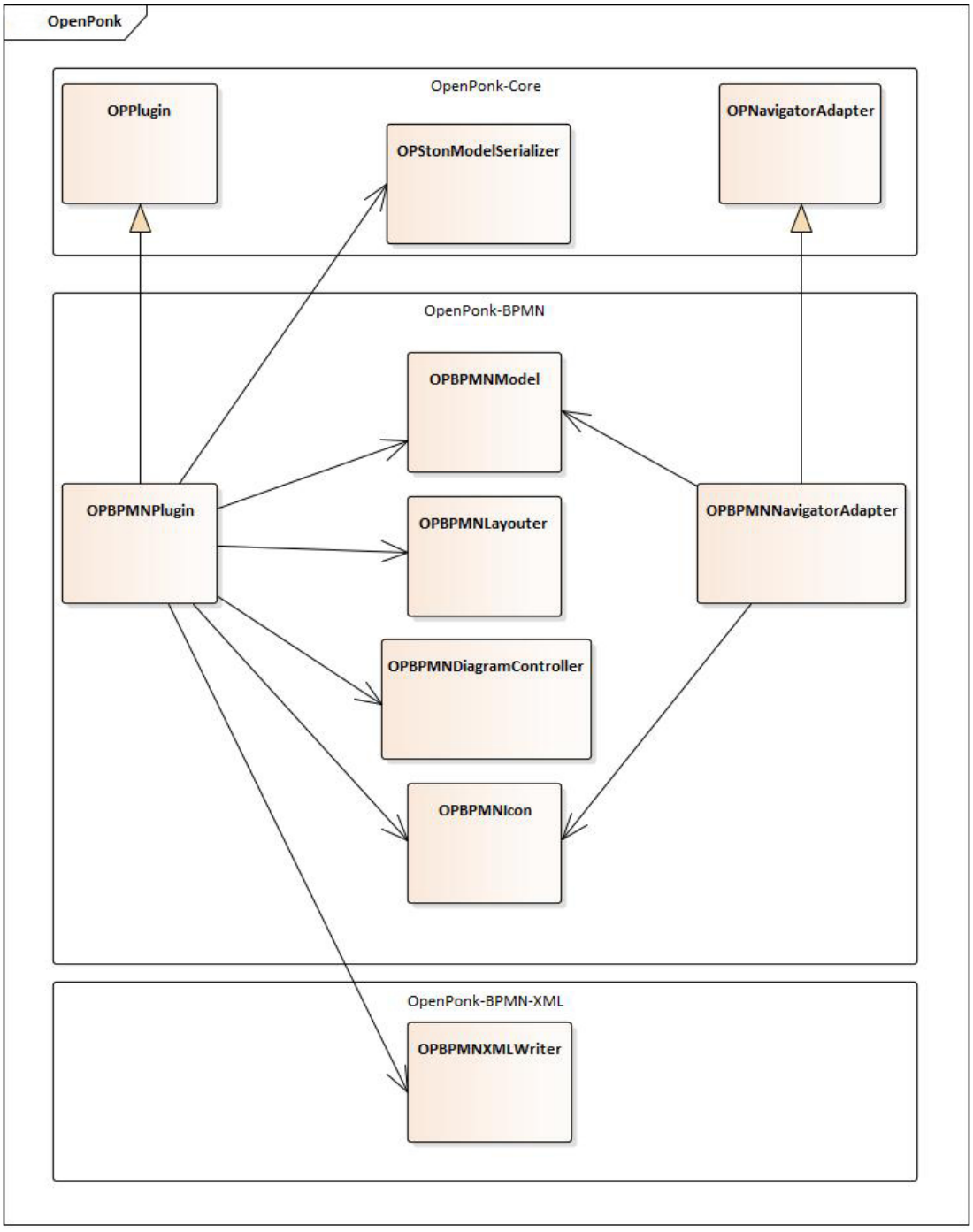
4.2.1 Základní třídy

Hlavní balíček rozšíření navazuje na existující rozhraní OpenPonku. Rozšíření se na ně napojuje pomocí potomka třídy `OPPlugin` – `OPBPMNPlugin`. Tato třída definuje základní vlastnosti rozšíření jako jsou název, ikona, další základní třídy pro jednotlivé vrstvy MVC, třídu pro serializaci diagramu k uložení a podporu skrývání prvků. Základními třídami jsou:

- `diagramControllerClass` → `OPBPMNDiagramController`,
- `layouterClass` → `OPBPMNLayouter`,
- `modelClass` → `OPBPMNModel`,
- `modelSerializerClass` → `OPStonModelSerializer`.

Třída `OPBPMNPlugin` obsahuje také dvě Class side metody, které volají druhý balíček umožňující export diagramu do XML. Jednou z nich je `toolBarMenu:`, která doplňuje základní implementaci panelu nástrojů o možnost „Export as XML“.

Druhou základní třídou je již zmíněný `OPBPMNLayouter`, který definuje jaké plátno má být vykresleno a jaké prvky je možné na něm vykreslit. V implementaci jsem využil *Roassal* implementaci plátna – `RTSugiyamaLayout`, která je používána i v některých dalších rozšířeních.



Obrázek 4.3 Struktura nejdůležitějších tříd a jejich rozdělení do balíčků. (Obrázek autora)

Pro identifikaci prvků, které se mají vykreslit bylo nutné implementovat metody `allEdges` a `allNodes`. Metoda `allEdges` vrací všechny controllery na hrany a `allNodes` vrací všechny controllery vložené na uzly. Bylo proto nutné vytvořit dvě třídy pro základní rozdělení controllerů, a to na `OPBPMNEdgeController` a `OPBPMNNodeController`. Metody tak vrací všechny prvky, které jsou potomky daného typu.

Třetí třída, `OPBPMNNavigatorAdapter`, definuje zobrazení umístěných prvků a navigaci mezi nimi, jedná se o panel, který se nachází v okně vlevo, viz obrázek 4.1. Do této třídy byly až na závěr implementace přidány všechny vytvořené prvky a jejich vazby.

4.2.2 Implementace modelu

Po vytvoření základních tříd bylo třeba doplnit logiku jednotlivých prvků, které se mají zobrazovat. OpenPonk nabízí základní implementaci, ale jelikož je každý prvek specifický, vznikla třída modelu pro každý prvek zvlášť. Hlavní třídou je `OPBPMNModel`, jejíž implementace dědí od základní třídy `OPModelObject`. Tato třída v sobě spravuje kolekci všech prvků, pomocí svých metod je opět rozděluje na hrany a uzly. Každému prvku tak ukládá povinnost implementovat metody `isEdge` a `isNode`, které vrací pravda/nepravda.

Logickým rozdělením ostatních modelů je dodržení již zavedené separace na uzly a hrany. `OPBPMNEdgeModel` dědí od `OPDirectedAssociation`, potomka `OPModelObject`. Toto rozšíření přidává atributy `source` a `target` a třídní metody pro vytvoření hrany ze zdroje k cíli `from:to:` a `from:to:named:`. Potomky `OPBPMNEdgeModel` jsou již konkrétní druhy hran. V této práci jsem implementoval základní dva druhy hran reprezentující průběh a tok procesu. Pro *Sequence Flow* vznikla třída `OPBPMNFlowModel` a pro *Message Flow* vznikla `OPBPMNMessageFlowModel`.

Implementace uzlů byla obtížnější. Do společné třídy `OPBPMNNodeModel`, dědící od známé `OPModelObject`, byla přidána kolekce `flows` pro hrany vedoucí do nebo z prvku a atribut `type`, který definuje poddruh prvku. Například *Activity* může být několika druhů jako jsou *Manual*, či Service. Pro tyto typy jsem vytvořil novou třídu `OPBPMNNodeType`, která obsahuje třídní metody vracející daný typ. Jedná se o implementaci výčtového typu, protože Pharo neobsahuje standardní *Enum* jako je tomu v jiných jazycích.

Podtřídy `OPBPMNNodeModel` jsou zástupci pro konkrétní prvky:

- `OPBPMNActivityModel`,
- `OPBPMNEventModel`,
  - `OPBPMNStartEventModel`,
  - `OPBPMNEndEventModel`,
  - `OPBPMNIntermediateEventModel`,
- `OPBPMNGatewayModel`,
- `OPBPMNPoolModel`.

Třídy se liší se pouze v implementaci metody `printType` vracející název prvku pro výpis. Jedinou odlišnou třídou je `OPBMNPoolModel`, která obsahuje metody pro přístup ke kolekci `elements`, protože vnořené prvky diagramu, tedy prvky vložené do *Pool*, se ukládají přímo do svého rodiče, ne do kořenového prvku jako ostatní.

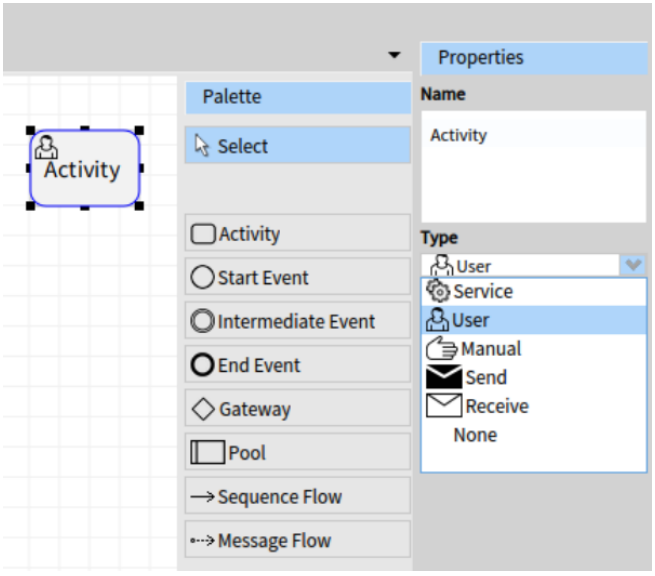
4.2.3 Implementace controlleru

Na diagramu tříd 4.5 jsou znázorněny všechny controllery včetně jejich propojení. Kořenovým controllerem je `OPBPMNDiagramController`, který odpovídá třídě modelu `OPBPMNModel`, každá další třída controlleru musí také implementovat metodu `modelClass`, která vrací odpovídající modelovou třídu. `OPBPMNDiagramController` je potomkem třídy `OPDiagramController`. Jedná se o nejdůležitější třídu uchovávající všechny informace o diagramu, od kolekce controllerů přes paletu prvků až po informace o okně a plátně samotném.

Metoda `controllerFactory`, třídy `OPBPMNDiagramController`, registruje všechny dostupné controllery a jejich modelové třídy. Metoda `updateView` je volána při každé změně na plátně a při načítání uloženého schématu, k načítání slouží také metoda `elementsToShowInside`, která zobrazuje všechny prvky. S implementací této metody jsem měl problémy při testování načtení diagamu ze souboru, protože musí vracet nejen své prvky z kolekce `elements`, ale také všechny vnořené prvky a jejich odchozí hrany, tedy prvky v *Pool*.

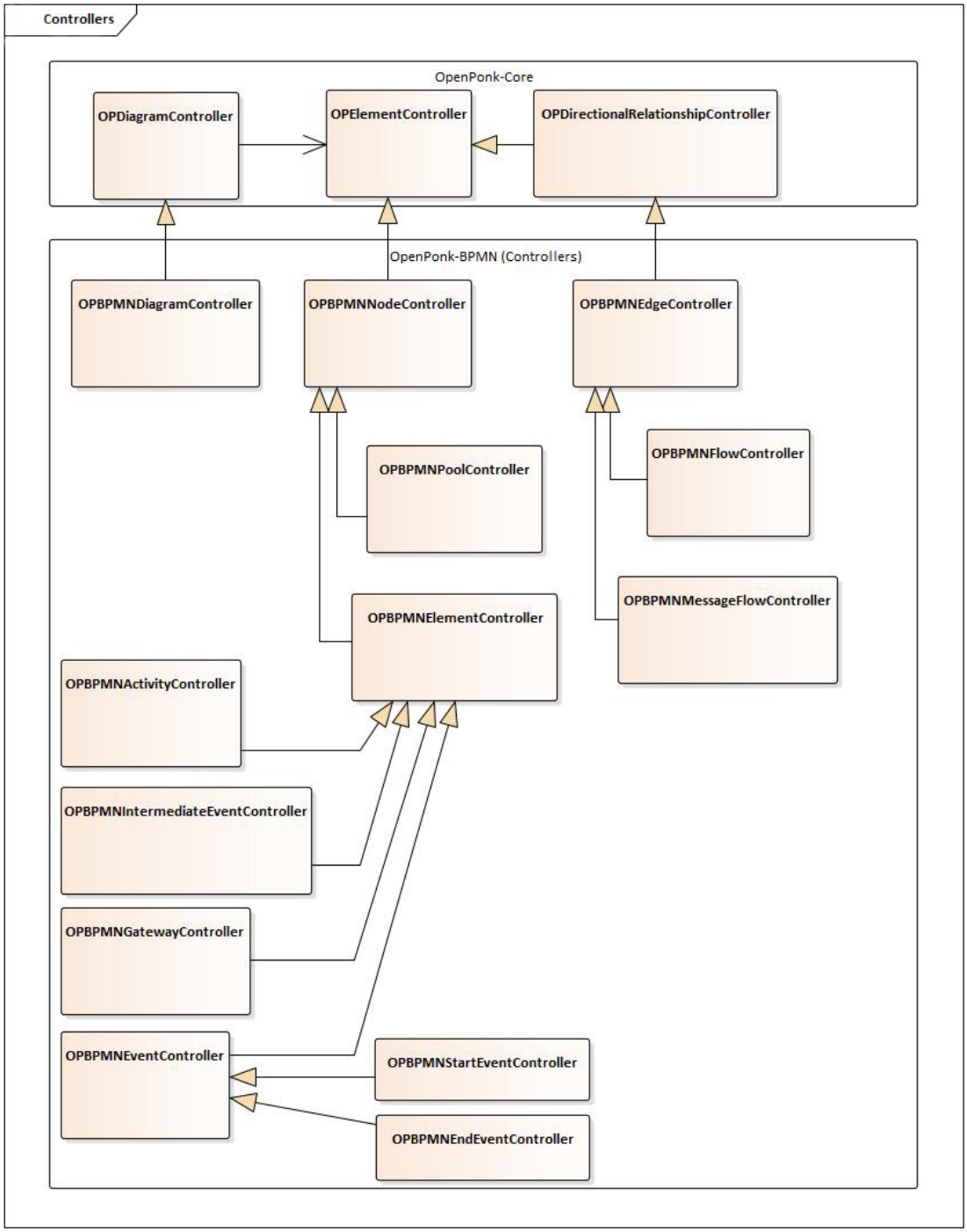
4.2.3.1 Paleta prvků

Paleta prvků je vytvářena metodou `initializePalette:` ve třídě `OPBPMNDiagramController`, pro každý prvek je v paletě zobrazené jméno a ikona pro větší přehlednost. Při vybrání prvku je vytvořen odpovídající controller. Paletu prvků lze vidět na obrázku 4.4 uprostřed. Chtěl bych upozornit na absenci plaveckých drah (*Lane*), které patří k základním prvkům BPMN. Tento prvek byl sice v rozšíření implementován, ale ne v dostatečně uživatelsky přívětivé podobě, proto byl z palety odebrán. Plavecké dráhy navíc mají pouze vizuální efekt, nemají vliv na průběh procesu ve spustitelné reprezentaci, který je pro tuto práci hlavním přínosem.



Obrázek 4.4 Vytvořený prvek aktivity s formulářem a paleta prvků pro BPMN. (Obrázek autora)

OpenPonk pro ukládání ikon a grafických prvků používá kódování base64, jedná se o převod binárních dat na tisknutelné ASCII znaky. Díky tomu mohou být ikony přímo integrovány v prostředí Pharo. Vzhledem ke standardu BPMN a nutnosti jej graficky dodžet jsem využil volně dostupnou sadu ikon *A BPMN 2.0 symbol font* [44] (uveřejněnou pod licencí SIL) používanou také v nástroji BPMN-iO [45].



Obrázek 4.5 Struktura tříd controllerů. (Obrázek autora)

Každou ikonu použitou v rozšíření bylo nutné převést pomocí photoshopu z formátu SVG do formátu PNG a zmenšit na požadovanou velikost 16x16 pixelů. Takto zmenšené ikony jsem zakódoval do base64 pomocí nástroje *Base64 Image Encoder* [46]. Pro přehlednost byla pro ikony vytvořena třída `OPBPMNIcon`, každé ikoně odpovídají dvě metody, jedna obsahuje zakódovaný obsah, druhá příkaz pro převedení zpět na obrázek. Obě metody jsou znázorněny v ukázce kódu 4.1, kde `activity` obsahuje dekódování obrázku a `activityContent` obsahuje část obrázku v kódování base64.

Výpis kódu 4.1 Ukázka vytvoření palety a dekodování obrázku v base64.

```
"OPBPMNDiagramController"
initializePalette: aPalette
  aPalette
    newCreationTool: 'Activity'
      factory: [ OPBPMNActivityController new ]
          icon: OPBPMNIcon current activity;

"OPBPMNIcon"
activity
  ^ Form
      fromBinaryStream:
          (self activityContent base64Decoded asByteArray readStream)

activityContent
  ^ 'iVBORw0KGgoAAAANSUhEUgAAABEAAAARCAYAAAA7bUf6AAAAAXNSR0I ...'
```

4.2.3.2 Controller uzlů

Společnou třídou zastřešující všechny controllery uzlů je `OPBPMNNodeController`, která dědí základní vlastnosti od `OPElementController`. Za zmínění stojí nejen metoda `buildEditorForm`, která vytváří formulář prvku pro zadání důležitých údajů, jako je název elementu. Například formulář pro aktivitu je zobrazen v pravé části obrázku 4.4. Po odeslání formuláře je volána metoda `refreshFigure`, která prvek aktualizuje.

Pro uzly jsou důležité dvě vlastnosti, které implementují syntaktická pravidla pro návrh diagramů. Metoda `canBeSourceFor:` pro hrany začínající v daném prvku, a `canBeTargetFor:` pro hrany končící v daném prvku, či pro prvky vytvářené uvnitř tohoto prvku. Při vytváření prvku navíc OpenPonk barevně odlišuje, kam lze, či nelze prvek umístit, takže není možné nakreslit diagram se zásadní syntaktickou chybou. Rád bych shrnul v následující tabulce 4.1, jaká syntaktická omezení se mi podařilo vytvořit, jedná se o základní pravidla vytváření BPMN diagramů.

Tabulka 4.1 Implementovaná syntaktická pravidla

Prvek	Může být zdrojem pro	Může být cílem pro
Activity	Message a Sequence flow	Message flow z jiného *Pool* a Sequence flow
Start Event	Message a Sequence flow	Message flow z jiného *Pool*
Interm. Event	Message a Sequence flow	Message flow z jiného *Pool* a Sequence flow
End Event	Message flow	Message flow z jiného *Pool* a Sequence flow
Gateway	Sequence flow	Sequence flow
Pool	Message flow	Message flow z jiného *Pool* a jiný uzel

Samotné vytvoření a vykreslení prvku má na starosti metoda `renderFigureIn`, nejprve je vytvořen základní *Roassal* tvar. Vytvoření tvaru jsem pro přehlednost přesunul do samostatné metody `createRoassalShape`, která je v `OPBPMNNodeController` pouze abstraktní. Poté je prvku

doplněn model, pole pro popisek a základní vlastnosti jako jsou `RTDraggable` – prvek lze posunout pomocí myši, či `OPRTFocusable` – při označení myší prvek změní barvu (na modrou). Obě metody pro vytvoření *Start Event* (bez vnitřní ikony), tedy `renderFigureIn` z `OPBPMNNodeController` a `createRoassalShape` z konkrétního `OPBPMNStartEventController`, jsou přiloženy v ukázce kódu 4.2.

Výpis kódu 4.2 Ukázka vytvoření tvaru pro *Start Event*

```
"OPBPMNNodeController"
renderFigureIn: aView
        self diagramController addingFigureFor: self.
        figure := self createRoassalShape elementOn: self model.
        figure addInteraction:
                        (OPRTMultilineLabeled new
                                view: aView;
                                below;
                                color: Color black;
                                yourself).
        figure @ RTDraggable.
        figure @ OPRTFocusable.
        figure renderIn: aView.
        ^ self diagramController addedFigureFor: self

"OPBPMNStartEventController"
createRoassalShape
        ^ RTRoundedBox new
                size: 36;
                borderRadius: 18;
                color: Color transparent;
                borderColor: Color black;
                yourself
```

Podtřídy `OPBPMNNodeController` jsem z praktických důvodů a také různých významů rozdělil na dva poddruhy:

OPBPMNPoolController v sobě může obsahovat všechny ostatní elementy, proto od nich byl logicky oddělen. Byla mu přidána kolekce `elements` a operace s ní spojené. Kolekce `elements` je typu `RTGroup`, aby bylo možné se všemi prvky nadále hýbat, přidávat je a mazat. Pro základní operace jsem vytvořil metody `addedNodeFigure:` a `removedNodeFigure:` pro přidání a odebrání prvku. Bohužel se mi nepodařilo dosáhnout toho, aby se při posunutí *Pool* posunuly i všechny ostatní vnořené prvky. Tento nedostatek bude třeba odstranit v další verzi rozšíření.

OPBPMNElementController zastřešuje ostatní prvky. Pro tuhle třídu je komplikované odebírání prvků a již v tabulce 4.1 vysvětlené metody `canBeSourceFor:` a `canBeTargetFor:`. Ve všech třech případech je nutné kontrolovat, zda vlastníkem prvku je *Pool* a při odebrání prvek odebrat i z něj. Všechny prvky navíc mohou obsahovat vnitřní ikony dle podtypu prvku (viz obrázek 4.4), tyto podtypy lze měnit pomocí formuláře vytvářeného metodou `buildEditorForm:`. Implementačně je `renderFigureIn` doplněna o vytvoření vnitřního elementu `RTBitmap`, který je při změně aktualizován metodou `refreshFigure`.

4.2.3.3 Controller hran

`OPBPMNEdgeController` je hlavním controllerem hran, dědí od již připravené třídy, kterou je `OPDirectionalRelationshipController`. Jedná se o rozšíření pro hrany vedoucí z nějakého prvku do jiného, proto jsou zde přidány atributy `source` a `target`, které představují zdrojový a cílový prvek. Oba dva prvky musí být upozorněni na přidání hrany, tedy ji přidat do své kolekce

hran. Druhy hran *Message Flow* i *Sequence Flow* vytvářím pomocí metody `createEdgeFrom:To:`, která je zde volána z `renderFigureIn`.

Hrany jsou opět tvarem z rozšíření *Roassal*, zde rozšířeného OpenPonkem o vytvoření hrany do stejného objektu jako je zdroj. Jedná se o třídu `OPRTStyledMultiLine`. Implicitní je však vytváření ze středu do středu prvku, proto je nutné přidání atributu `withBorderAttachPoint`, aby hrana začínala a končila na okraji prvku. Implementace pro *Message Flow* se liší pouze v přidání dvou atributů při vytváření *Roassal* tvaru:

```
tail: RTEmptyCircle;
dashedLine;
```

Hrana může mít také svůj popisek, se kterým jde narozdíl od popisků ostatní elementů hýbat, a to pro větší přehlednost v diagramu. Pohyb je umožněn pomocí řádku `RTDraggable;` při vytváření popisku `RTLabel`.

4.2.4 Balíček pro export do XML

Export diagramu je možné vytvořit v liště nástrojů v záložce „Diagram“, zde jsou k dispozici dvě možnosti „Export as PNG“ a „Export as XML“. Třídy pro exportování do XML (XSD formátu) jsou volány přímo ze třídy `OPBPMNPlugin` pomocí metody `exportPackage:`, které je pomocí parametru předána instance `OPBPMNDiagramController`.

Formát exportovaného souboru je univerzální bez vazby na process engine, tedy podle BPMN specifikace. Defaultně je nastavena hodnota atributu procesu na `isExecutable="false"`, tuhle hodnotu lze změnit ve formuláři kořenového elementu. Níže v praktické ukázce bude provedena změna ve zdrojovém kódu pro generování pro daný process engine (přidání namespace).

Před implementací této části bylo nutné podrobně prostudovat formát XML souborů, k tomu mi vypomohly ukázkové příklady na webu OMG [47]. Také jsem hojně využíval dvou konkurenčních nástrojů pro zjištění kompatibility: Camunda Modeler [33] a Bizagi Modeler [48], v obou nástrojích dochází ke správnému vykreslení diagramů exportovaných z platformy OpenPonk.

Během testování diagramu v těchto nástrojích jsem narazil na několik problémů a omezení, na které jsem při rešerši fungování XML formátu nenarazil. Například ID prvku musí začínat buď na „NazevPrvku_“ nebo sekvencí znaků „_6-“, před všechny ID generovaná OpenPonkem jsem proto musel dávat takovýto prefix. Další problém vznikl při vytvoření diagramu s bazény, název tagu pro *Pool* je totiž shodný jako pro tag označující celý proces, ve kterém se *Pool* nevyskytuje. Proto je nutné nejprve zkontrolovat, zda je v procesu obsažen bazén či nikoliv.

4.2.4.1 Pharo XML balíček

Pharo nabízí třídy pro základní práci s XML soubory, ve své bakalářské práci jsem využil balíček `XML-Writer-Core` se základní třídou `XMLWriter`. Při použití pouze této třídy by byl celý dokument na jednom řádku, proto balíček obsahuje i třídy pro formátování XML v čele s třídou `XMLWriterFormatter`.

V následující ukázce kódu 4.3 je vytvářeno jednoduché XML pro prvek aktivita. Jak lze z ukázky vidět, tak není třeba prvky vypisovat v XML formátu. Pro vytvoření jednoho tagu (značky) se používá následující syntaxe: `writer tag: attributes: with:`, kde `tag` specifikuje název značky. Atributy jsou sdruženy v poli `attributes` ve tvaru název → hodnota. Poslední `with:` definuje vnořené hodnoty nebo tagy, tak je umožněno vytváření stromové struktury dokumentu.

Výpis kódu 4.3 Ukázka vytvoření jednoduchého XML v jazyce Pharo

```
model := OPBPMNActivityModel new.
model name: 'Open ticket'.

(writer := XMLWriter new)
        xml;
        enablePrettyPrinting;
        tag: 'semantic:', model printType
        attributes: (Array with: 'name' -> model name
                             with: 'id' -> model uuid asString).

"Vystup:"
<?xml version="1.0"?>
<semantic:task name="Open ticket"
               id="5276ebcf-327d-0d00-8d78-fe6201f83167"/>
```

4.2.4.2 Vytvořené třídy

Generování XML jsem rozdělil do tří tříd. První z nich je `OPBPMNXMLPrettyWriter`, tato třída definuje vlastní formátování dokumentu, ale vychází z implementace `XMLPrettyPrintingWriterFormatter`, od které dědí. Tato třída se stará o zalomení řádky a odsazení vnořených tagů. Oproti Pharo reprezentaci je přidán atribut vlastního znaku konce řádku, díky tomu je umožněno použít libovolné kódování znaků. V implementaci používám pro konec řádku standardní Unicode znak „
“ a kódování UTF-8.

Druhou třídou je `OPBPMNXMLWriter`, tato třída je volána přímo z hlavní třídy `OPBPMNPlugin`. Jediným úkolem této třídy je vytvořit z dat, které zpracovává, plnohodnotný text, tedy `String`.

Poslední třídou je `OPBPMNXMLWriterVisitor`, která obsahuje instanci `XMLWriter` a dvě kolekce pro všechny hrany, jednu pro všechny *Sequence Flow* a druhou pro všechny *Message Flow*. Instanci `XMLWriter` je předána moje formátovací třída `OPBMNXMLPrettyWriter`, již zmíněné kódování a znak konce řádku.

Celé XML je rozděleno na dvě části: část sémantickou a část diagramovou, proto jsou tak rozdělené i metody v této třídě. Nejprve je však volána metoda `visit:`, která vytváří úvodní tag „definitions“ s atributy definujícími použité jmenné prostory. Vnořeně do tagu „definitions“ poté volá metody `visitSemantic:` a `visitDiagram:`.

Sémantická část překvapivě není uvozena žádným tagem, je tak nutné zjistit, zda diagram obsahuje bazény, či nikoliv. Pokud diagram neobsahuje bazén, je nutné jej celý obalit tagem „process“. Pokud bazén obsahuje nebo dokonce více, je nutné každý bazén uvést tagem „process“, a navíc je třeba na konci části v tagu „collaboration“ vyjmenovat všechny *Pool* a *Message Flow*.

Další volaná metoda `visitElement:` již vypisuje konkrétní prvky. Uvnitř prvku jsou vypsány odkazy na příchozí a odchozí hrany. Každá odchozí hrana je v této metodě přidána do kolekce všech hran, jejichž definice jsou vypsány až na konci každého tagu „process“. Každá takto vnořená část je reprezentována vlastní metodou pro větší přehlednost.

Diagramová část má za úkol vypsat velikosti a umístění prvků. Pro tuto část je definován jiný jmenný prostor „bpmndi“ a je možné ji mít ve zvláštním souboru, většinou se však přidává do stejného souboru za sémantickou část. Definice diagramu jsou uvozeny tagem „BPMNDiagram“, který má mimo názvu a ID ještě specifický atribut „resolution“, jeho hodnotu jsem po experimentování nastavil na 96,0, tak aby byly prvky v jiných nástrojích na první pohled přiměřeně velké. Tento úvodní tag vytváří metoda `visitDiagram:` spolu s tagem „BPMNPlane“, který představuje celý proces.

Konkrétní prvky, zde všechny contollery, jsou již zpracovávány vnořeně dalšími metodami. Obtížné bylo zjistit, jak jsou ukládány souřadnice prvků a jak se k nim dostat, protože se jedná o interní záležitost OpenPonku, kterou jsem jinde nemusel řešit. Souřadnice jsou ukládány v con-

trolleru v proměnné `figure`, která představuje *Roassal* tvar prvku, jeho velikost a umístění v prostoru. Tyto souřadnice však ukazují na střed prvku, ale v XSD formátu jsou chápány souřadnice jako levý horní roh prvku, proto bylo třeba je posunout o polovinu velikosti prvku. Práci s prvkem, včetně vypsání jeho souřadnic lze vidět v ukázce kódu 4.4.

Naopak souřadnice pro hrany jsem umístil do středu zdrojového a cílového prvku hrany, *Roassal* totiž nezaznamenává bod dotyku hrany s prvkem. Toto chování není podle BPMN specifikace, ale dle testování v jiných nástrojích se umí s takto definovanými souřadnicemi vypořádat a nezpůsobí ani žádné chyby při zpracování XML.

Výpis kódu 4.4 Ukázka vytvoření diagramové části, včetně přístupu k souřadnicím prvku

```
visitDiagramElement: anElement
writer
  tag: self diagramNamespace , 'BPMNShape'
  attributes:
    (Array
      with:
      'bpmnElement'
      -> (anElement model printIdType , anElement model uuid asString))
  with: [
    writer
      tag: 'dc:Bounds'
      attributes:
      (Array
        with: 'height' -> anElement figure height asString
        with: 'width' -> anElement figure width asString
        with:
          'x' -> (anElement figure position x
                - (anElement figure width / 2.0)) asString
        with:
          'y' -> (anElement figure position y
                - (anElement figure height / 2.0)) asString).
    writer tag: self diagramNamespace , 'BPMNLabel' ]

"Mozny Vystup:"
<bpmndi:BPMNShape bpmnElement="_6-0e5130ce-1e7d-0d00-a4c8-c2f20e3ea2c6">
   <dc:Bounds height="50.0" width="99.9972"
              x="-54.9981" y="-154.9870"/>
   <bpmndi:BPMNLabel/>
</bpmndi:BPMNShape>
```

4.2.4.3 Spustitelný formát

Pro potřeby spustitelnosti bylo třeba doplnit vlastnosti pro některé prvky a již zmíněný atribut `isExecutable`. Zaměřil jsem se zejména na časované události a *Service* a *Send* prvky. Všechny tyto vlastnosti je možné upravovat ve formuláři pro daný prvek (Pro zobrazení při změně typu je třeba kliknout na jiný prvek a až poté zpět pro odeslání a obnovení formuláře).

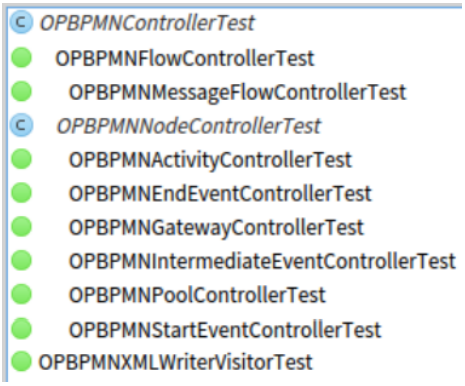
Implementačně jsem tyto prvky přenesl přímo do modelu jako instanční proměnné, pro zobrazení ve formuláři bylo třeba rozšířit metody `buildEditorForm:` upravených elementů.

Start Event a *Intermediate Event* prvky mohou být typu *Timer*, v praxi to znamená tři možné scénáře: čekat po nějakou dobu (time duration), čekat na určitý datum a čas (time date) nebo opakovat událost v cyklech (time cycle). Z těchto tří možností je možné si vybrat ve formuláři daného prvku a přiřadit definici hodnotu, příkladem pro vyplněnou hodnotu může být čekání po dobu pěti minut, zapsáno jako „PT5M“. V XML poté přibude vnořený tag „timerEventDefinition“, který událost definuje.

Activity, *Intermediate Event* a *End Event* mohou být typu *Send*, *Activity* navíc může být typu *Service*. Všechny tyto prvky obsahují ve spustitelné podobě atribut navíc, který označuje, co se má stát při vykonávání procesu v tomto prvku. Jedná se o atribut specifický pro process engine, proto jsou ve formuláři volná textová pole pro atribut a hodnotu (Executable atribute a Executable atribute value). Většinou se v závislosti na procesním stroji změní jmenný prostor pro atribut, například pro engine Camunda může mít prvek hodnotu „camunda:class“, pro engine Flowable se jedná o hodnotu „flowable:class“.

4.3 Testování

Aplikaci jsem testoval nejen vizuálně, či pomocí jiných již zmíněných nástrojů, ale také pomocí jednotkových testů. Pharo nabízí jednoduché rozhraní pro testování, které umožňuje snadné pouštění testů. Každá testovací třída musí dědit od základní třídy `TestCase`. Pro složitější testy, při kterých jsou třeba již existující instance jiných objektů, je možné využít metody `setUp`, která se spouští vždy ještě před vykonáváním testů. Všechny vytvořené testovací třídy včetně označení výsledků testů jsou k vidění na obrázku 4.6.



Obrázek 4.6 Vytvořené testovací třídy včetně výsledků testů – zelené úspěšné. (Obrázek autora)

Při testování controllerů jsem využil připravenou testovací třídu `OPControllerTest`, která se zaměřuje na základní vlastnosti controllerů. Třídu jsem rozšířil o další testy pomocí dalších tříd, které od ní dědí. U každého konkrétního controlleru se testuje: modelová třída, vytvoření modelové třídy, vytvoření controlleru, vytvoření tvaru, vytvoření popisku, návaznost na rodiče a aktualizace upraveného controlleru.

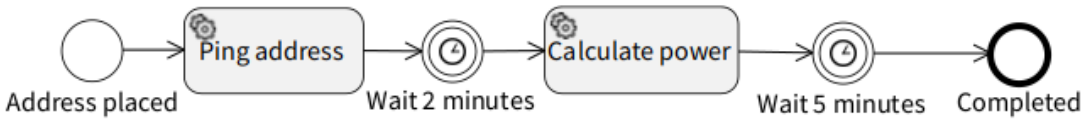
V části exportu diagramů jsem vytvořil testovací třídu pro `OPBPMNXMLWriterVisitor`, který obsahuje celou logiku vytváření exportu. Zde jsem testoval zejména sémantickou část včetně spustitelných atributů, jejich pravidla a správnost vypisování do XML. Užitečné bylo zejména testování správnosti ID referencí u hran, protože ID prvků jsou dlouhá a vizuální kontrola je náročná.

4.4 Modelová studie

Pro simulaci funkčnosti implementace jsem vytvořil modelový příklad, ve kterém je ukázána orchestrace dvou jednoduchých služeb. Využil jsem vytvořené šablony Maven projektu [49] v jazyce Java, jedná se o šablonu pro *Camunda Process Engine* [50]. Obdobně lze tvořit i pro jiné procesní stroje, pro ilustraci jBPM [35] a Flowable [36] nabízejí návody na vytvoření Maven projektu se stejnou strukturou, pouze za využití jejich procesního stroje.

4.4.1 Vytvoření modelové studie

Nejprve jsem si v OpenPonku vytvořil schéma procesu, viz obrázek 4.7. Jedná se o jednoduchý proces se dvěma aktivitami. Uživatel pomocí restového rozhraní pošle název webové adresy, ta je v první službě zpracována a je proveden ping na danou adresu. Poté proces čeká dvě minuty pro možnost kontroly funkčnosti. Doba čekání na odpověď dotazu ping je zpracována druhou službou, ve které je vytvořena mocnina z tohoto čísla. Než proces skončí, čeká ještě po dobu pěti minut, opět pouze pro možnost kontroly funkčnosti.



Obrázek 4.7 Diagram procesu modelové studie. (Obrázek autora)

Dalším krokem bylo doplnění spustitelných atributů do procesu, a to pomocí formulářů prvků. Obě aktivity spouštím pomocí atributu „camunda:class“ s hodnotou shodnou s názvem aktivity. Oba *Intermediate Event* jsem nastavil na typ *Timer* a změnil atributy na „timeDuration“ s hodnotou „PT2M“ a obdobně pro druhou na pět minut. Celý proces jsem nastavil jako spustitelný.

Ještě před exportováním procesu jsem v kódu do metody `visit:` připsal potřebné jmenné prostory pro *Camunda Process Engine*:

`xmlns:camunda="http://camunda.org/schema/1.0/bpmn"`,

`targetNamespace="http://bpmn.io/schema/bpmn"`.

Samozřejmě je možné tyto definice přidat až do výsledného XML, ale bylo by nutné tuhle změnu provádět při každém exportu zvlášť. Může se také stát, že pro složitější procesy bude nutné provést v kódu více změn, a právě zde se ukazuje výhoda platformy OpenPonk, kterou je její vysoká flexibilita.

Nyní je možné vytvořit export do XML (viz příloha A) a vložit jej přímo do připravené Maven šablony.

V Javě jsem připravil třídy pro jednotlivé služby. `PingAdapter` s několikařádkovou implementací pro ping na danou adresu a změření času. Pokud adresa není dostupná do dvou sekund, metoda ukládá do proměnné `currentTime` hodnotu 0. Druhá třída `PowAdapter` umocní hodnotu `currentTime` a uloží ji do proměnné `pow`.

Nakonec jsem vytvořil řídu pro REST rozhraní `PingRestController` s jedinou metodou *POST*, pro odeslání požadavku s proměnnou `address` na odkaze „/ping“. Tato třída zpracuje požadavek a spustí proces s hodnotou adresy. Ke spuštění procesu slouží metoda z Camunda balíčku `startProcessInstanceByKey`.

4.4.2 Ukázka modelové studie

Před spuštěním modelové studie je třeba mít nainstalovanou Javu a nástroj Maven. Všechny závislosti ukázkového Maven projektu se instalují samy příkazem z adresáře projektu:

```
mvn clean install
```

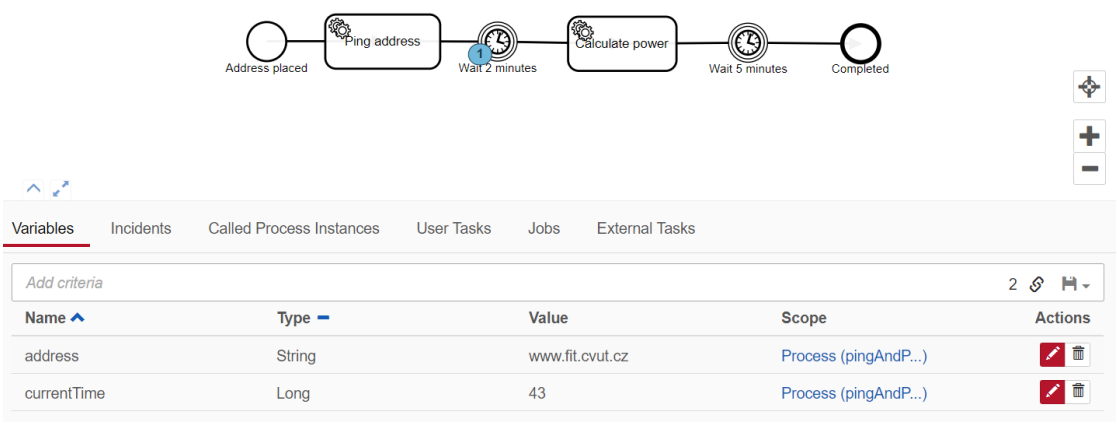
Ke spuštění slouží příkaz:

```
java -jar
target/camunda-spring-boot-amqp-microservice-cloud-example-0.0.1-SNAPSHOT.jar
```

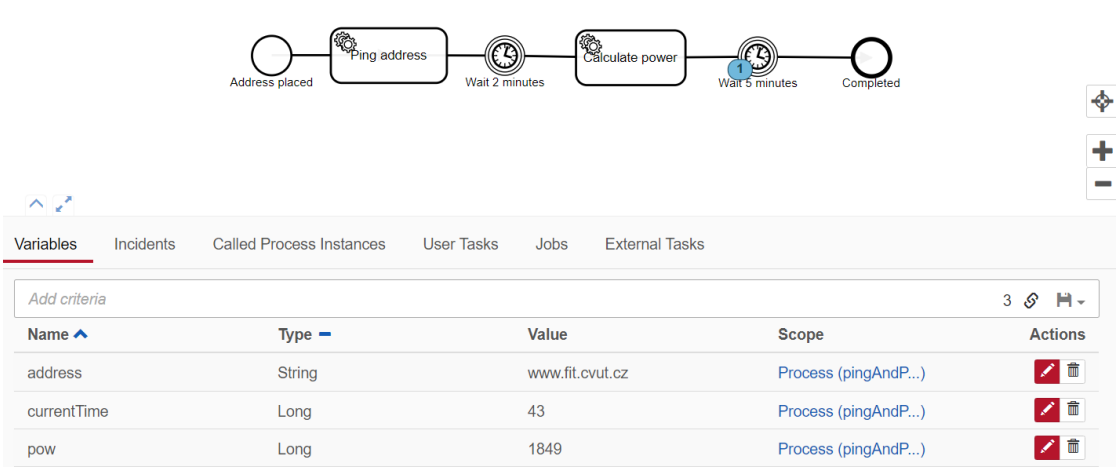
Již běží engine na adrese „localhost:8080“, počáteční přístupové údaje do webového rozhraní Camunda jsou „demo demo“. Zde je možné sledovat proces a jeho instance. Proces je spouštěn REST dotazem, který lze poslat například z příkazové řádky:

```
curl -d "address=www.fit.cvut.cz" -X POST http://localhost:8080/ping
```

Dotazem byla spuštěna instance procesu, které je přiřazeno ID. Celý průběh procesu pak lze sledovat ve webovém rozhraní, jak je znázorněno na obrázcích 4.8 a 4.9.



Obrázek 4.8 Průběh instance procesu modelové studie – stav po první službě. (Obrázek autora)



Obrázek 4.9 Průběh instance procesu modelové studie – stav po druhé službě. (Obrázek autora)

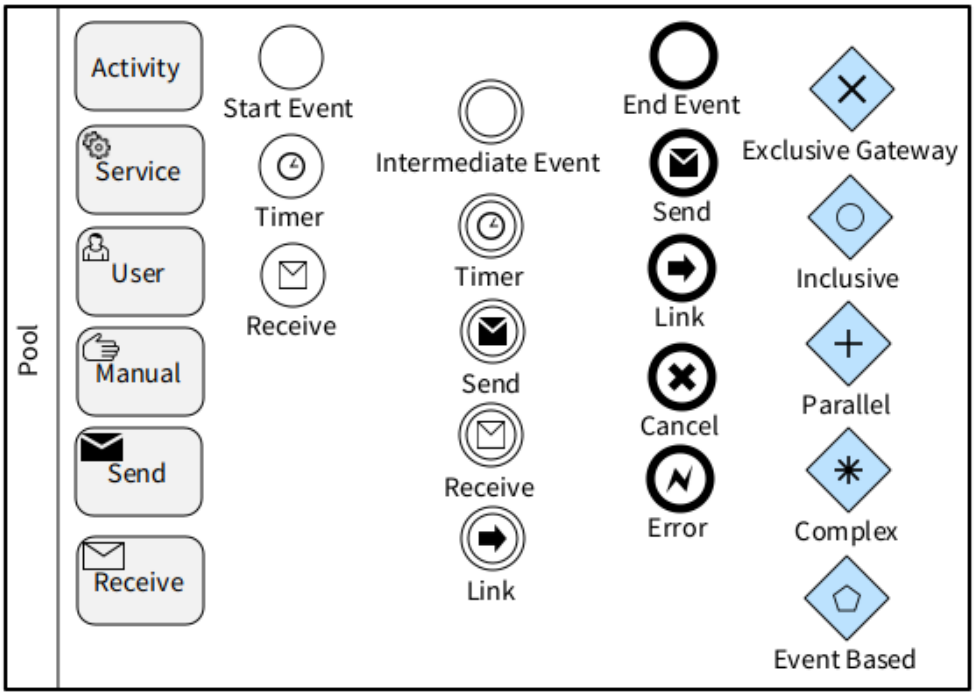
4.5 Shrnutí implementace a možnost budoucího vývoje

Bylo vytvořeno rozšíření pro platformu OpenPonk, které umožňuje vytváření základních BPMN diagramů. Tyto diagramy je možné použít k orchestraci webových služeb díky možnosti exportu diagramu do spustitelného XML formátu.

Z palety jazyka BPMN 2.0 se mi podařilo zachytit nejpoužívanější prvky včetně jejich typů (viz obrázek 4.10) a základní dva druhy hran *Sequence Flow* a *Message Flow*. Také se mi podařilo implementovat základní syntaktická pravidla pro vytváření BPMN diagramů, včetně omezení použití *Message Flow*.

Ze základních prvků jsem vynechal *Lane*, a to ze dvou důvodů. Tento prvek nemá v orchestraci webových služeb žádný význam, jedná se pouze o vizuální prvek. Druhým důvodem je technický aspekt, že se mi nepodařilo vytvořit uživatelsky přívětivou variantu vkládání *Lane* do diagramu. Tento prvek by mohl být dodělán v dalších verzích rozšíření, ideální by bylo využití tlačítek, které by umožňovaly přidat dráhu do stávajícího bazénu nad nebo pod ostatní dráhy.

Myslím, že by se dalo také navázat na vytvořený XML formát, který je sice funkční, ale nabízí mnoho možností rozvoje. Jedním z dalších kroků by mohlo být přidání spustitelných atributů také zbylým prvkům a přidání možnosti importu XML souboru. Pokud by uživatelé využívali možnosti upravit si aplikaci podle procesního stroje, který zrovna používají, líbilo by se mi vytvoření profilů aplikace pro nejpoužívanější procesní stroje. Uživatel by si jen před exportem vybral, pro jaký z nich chce XML vytvořit.



Obrázek 4.10 Paleta všech prvků v rozšíření platformy OpenPonk pro jazyk BPMN. (Obrázek autora)

Kapitola 5
Závěr

Cílem této práce bylo vytvořit nástroj pro modelování orchestrace webových služeb pomocí jazyka BPMN a demonstrovat jeho použití.

Výsledný modul pro platformu OpenPonk, napsaný v jazyce Pharo, umožňuje při modelování využít nejpoužívanějších prvků z notace BPMN. Při vytváření diagramu je vynuceno dodržování základních syntaktických pravidel. Je možné rozpracovaný diagram uložit, později znovu otevřít a pokračovat v práci.

Také je možné exportovat diagram do XML formátu XSD, který je nejrozšířenějším pro přenos BPMN diagramů (soubory .bpmn). XSD dokument lze také spustit na procesním stroji (process engine) a pomocí této BPMN reprezentace je možné orchestrovat webové služby.

V modelovém příkladu jsem v jazyce Java vytvořil dvě jednoduché služby a pomocí procesního stroje je spolu s exportovaným BPMN diagramem nahrál na server a otestoval funkčnost jejich orchestrace pomocí BPMN. Samotné instance procesu jsem spouštěl pomocí REST dotazů.

Přínosem, který tato práce a vzniklé rozšíření přináší, oproti jiným modelovacím nástrojům, je nezávislost na implementaci procesního stroje. OpenPonk díky možnosti přímé editace zdrojového kódu a jednoduché uživatelsky přívětivé implementaci umožňuje (techničtěji orientovaným) uživatelům upravovat si svoji verzi rozšíření. Tak je možné dosáhnout kompatibility BPMN diagramu napříč různými procesními stroji.

Modul pro jazyk BPMN je dalším krokem pro možnost provedení kompletní Business analýzy v nástroji OpenPonk, který již podporuje notace jako je UML, OntoUML či DEMO. Přestože se začíná OpenPonk stávat komplexním nástrojem, který je neustále rozvíjen, stále si zachovává svou jednoduchost a uživatelskou přívětivost.

Práce také přináší několik možností budoucího rozvoje. V dalších verzích je možné rozšířit modul o ostatní prvky notace BPMN a import diagramů z jiných nástrojů. Navíc OpenPonk nabízí rozhraní pro simulaci diagramů, o kterou by bylo možné BPMN modul rozšířit. Také bych viděl jako přínos vytvoření několika profilů pro export diagramů do nejrozšířenějších implementací procesních strojů.

Příloha A

Export diagramu modelové studie

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions targetNamespace="http://bpmn.io/schema/bpmn"
    xmlns:camunda="http://camunda.org/schema/1.0/bpmn"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
    xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
    xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL">
<bpmn:process isExecutable="true" id="pingAndPow">
    <bpmn:startEvent name="Address placed"
        id="_6-0e5130ce-1e7d-0d00-a4c6-bd810e3ea2c6">
        <bpmn:outgoing>_6-0e5130ce-1e7d-0d00-a4cb-427f0e3ea2c6
        </bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:serviceTask name="Ping address"
        id="_6-0e5130ce-1e7d-0d00-a4c7-8c720e3ea2c6"
        camunda:class="com.camunda.demo.springboot.adapter.PingAdapter">
        <bpmn:incoming>_6-0e5130ce-1e7d-0d00-a4cb-427f0e3ea2c6
        </bpmn:incoming>
        <bpmn:outgoing>_6-32718c3d-547d-0d00-897e-c5f50a761af9
        </bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:serviceTask name="Calculate power"
        id="_6-0e5130ce-1e7d-0d00-a4c8-c2f20e3ea2c6"
        camunda:class="com.camunda.demo.springboot.adapter.PowAdapter">
        <bpmn:incoming>_6-32718c3d-547d-0d00-897f-63590a761af9
        </bpmn:incoming>
        <bpmn:outgoing>_6-0e5130ce-1e7d-0d00-a4cd-8bba0e3ea2c6
        </bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:intermediateCatchEvent name="Wait 5 minutes"
        id="_6-0e5130ce-1e7d-0d00-a4c9-c9b00e3ea2c6">
        <bpmn:incoming>_6-0e5130ce-1e7d-0d00-a4cd-8bba0e3ea2c6
        </bpmn:incoming>
        <bpmn:outgoing>_6-0e5130ce-1e7d-0d00-a4ce-66fa0e3ea2c6
        </bpmn:outgoing>
        <bpmn:timerEventDefinition
        id="timerEventDefinition_0e5130ce-1e7d-0d00-a4c9-c9b00e3ea2c6">
```

```
        <bpmn:timeDuration xsi:type="semantic:tFormalExpression">
            PT5M</bpmn:timeDuration>
        </bpmn:timerEventDefinition>
    </bpmn:intermediateCatchEvent>
     <bpmn:endEvent name="Completed"
        id="_6-0e5130ce-1e7d-0d00-a4ca-50610e3ea2c6">
        <bpmn:incoming>_6-0e5130ce-1e7d-0d00-a4ce-66fa0e3ea2c6
        </bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:intermediateCatchEvent name="Wait 2 minutes"
        id="_6-32718c3d-547d-0d00-897d-b94a0a761af9">
        <bpmn:incoming>_6-32718c3d-547d-0d00-897e-c5f50a761af9
        </bpmn:incoming>
        <bpmn:outgoing>_6-32718c3d-547d-0d00-897f-63590a761af9
        </bpmn:outgoing>
        <bpmn:timerEventDefinition
        id="timerEventDefinition_32718c3d-547d-0d00-897d-b94a0a761af9">
            <bpmn:timeDuration xsi:type="semantic:tFormalExpression">
            PT2M</bpmn:timeDuration>
        </bpmn:timerEventDefinition>
    </bpmn:intermediateCatchEvent>
    <bpmn:sequenceFlow
    sourceRef="_6-0e5130ce-1e7d-0d00-a4c6-bd810e3ea2c6"
    targetRef="_6-0e5130ce-1e7d-0d00-a4c7-8c720e3ea2c6"
    name="" id="_6-0e5130ce-1e7d-0d00-a4cb-427f0e3ea2c6"/>
    <bpmn:sequenceFlow
    sourceRef="_6-0e5130ce-1e7d-0d00-a4c7-8c720e3ea2c6"
    targetRef="_6-32718c3d-547d-0d00-897d-b94a0a761af9"
    name="" id="_6-32718c3d-547d-0d00-897e-c5f50a761af9"/>
    <bpmn:sequenceFlow
    sourceRef="_6-0e5130ce-1e7d-0d00-a4c8-c2f20e3ea2c6"
    targetRef="_6-0e5130ce-1e7d-0d00-a4c9-c9b00e3ea2c6"
    name="" id="_6-0e5130ce-1e7d-0d00-a4cd-8bba0e3ea2c6"/>
    <bpmn:sequenceFlow
    sourceRef="_6-0e5130ce-1e7d-0d00-a4c9-c9b00e3ea2c6"
    targetRef="_6-0e5130ce-1e7d-0d00-a4ca-50610e3ea2c6"
    name="" id="_6-0e5130ce-1e7d-0d00-a4ce-66fa0e3ea2c6"/>
    <bpmn:sequenceFlow
    sourceRef="_6-32718c3d-547d-0d00-897d-b94a0a761af9"
    targetRef="_6-0e5130ce-1e7d-0d00-a4c8-c2f20e3ea2c6"
    name="" id="_6-32718c3d-547d-0d00-897f-63590a761af9"/>
</bpmn:process>
<bpmndi:BPMNDiagram
id="BPMNDiagram_0e5130ce-1e7d-0d00-a4c5-09a30e3ea2c6"
name="PingAndPow" resolution="96.0">
    <bpmndi:BPMNPlane
    bpmnElement="process_0e5130ce-1e7d-0d00-a4c5-09a30e3ea2c6">
        <bpmndi:BPMNEdge
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4cd-8bba0e3ea2c6">
            <di:waypoint x="79.99200079992009" y="-131.98680131986802"/>
            <di:waypoint x="195.980401959804" y="-132.98670132986697"/>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNShape
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4ca-50610e3ea2c6">
            <dc:Bounds height="34.0" width="34.0" x="282.99400059994"
            y="-149.01079892010796"/>
```

```
        <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNEdge
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4cb-427f0e3ea2c6">
            <di:waypoint x="-247.975202479752" y="-134.38656134386565"/>
            <di:waypoint x="-140.5859414058593"
            y="-134.38656134386565"/>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNShape
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4c6-bd810e3ea2c6">
            <dc:Bounds height="36.0" width="36.0" x="-265.975202479752"
            y="-152.38656134386565"/>
            <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNEdge
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4ce-66fa0e3ea2c6">
            <di:waypoint x="195.980401959804" y="-132.98670132986697"/>
            <di:waypoint x="299.99400059994" y="-132.01079892010796"/>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNShape
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4c7-8c720e3ea2c6">
            <dc:Bounds height="50.0" width="105.99660033996616"
            x="-193.58424157584238" y="-159.38656134386565"/>
            <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNShape
        bpmnElement="_6-32718c3d-547d-0d00-897d-b94a0a761af9">
            <dc:Bounds height="36.0" width="36.0" x="-52.996500349965"
            y="-151.98660133986604"/>
            <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNEdge
        bpmnElement="_6-32718c3d-547d-0d00-897e-c5f50a761af9">
            <di:waypoint x="-140.5859414058593"
            y="-134.38656134386565"/>
            <di:waypoint x="-34.996500349965" y="-133.98660133986604"/>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNShape
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4c8-c2f20e3ea2c6">
            <dc:Bounds height="50.0" width="99.99720027997208"
            x="29.993400659934053" y="-156.98680131986802"/>
            <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNShape
        bpmnElement="_6-0e5130ce-1e7d-0d00-a4c9-c9b00e3ea2c6">
            <dc:Bounds height="36.0" width="36.0" x="177.980401959804"
            y="-150.98670132986697"/>
            <bpmndi:BPMNLabel/>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNEdge
        bpmnElement="_6-32718c3d-547d-0d00-897f-63590a761af9">
            <di:waypoint x="-34.996500349965" y="-133.98660133986604"/>
            <di:waypoint x="79.99200079992009" y="-131.98680131986802"/>
        </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
</bpmn:definitions>
```

Bibliografie

1. ANISIMOV, Boris. *Podpora standardu BPMN na platformě OpenPonk* [online]. 2018 [cit. 2021-03-29]. Dostupné z: https://dspace.cvut.cz/handle/10467/63136/. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce ING. ROBERT PERGL, PH. D.
2. BOOTH, David; HAAS, Hugo et al. *Web Services Architecture* [online]. W3C, 2004 [cit. 2021-03-03]. Dostupné z: http://www.w3.org/TR/ws-arch/#whatis.
3. CHURÝ, Lukáš; LEHOCKÝ, Zdeněk. *Základy XML webových služeb* [online]. Programujte, 2005 [cit. 2021-03-05]. ISSN 1801-1586. Dostupné z: http://programujte.com/clanek/2005081704-zaklady-xml-webovych-sluzeb/.
4. QUIN, Liam. *XML Essentials* [online]. W3C, 2015 [cit. 2021-04-25]. Dostupné z: https://www.w3.org/standards/xml/core.
5. BOX, Don; EHNEBUSKE, David et al. *Simple Object Access Protocol* [online]. W3C, 2000 [cit. 2021-03-05]. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.
6. KOSEK, Jiří. *Inteligentní podpora navigace na WWW s využitím XML* [online]. 2010 [cit. 2021-03-05]. Dostupné z: https://www.kosek.cz/diplomka/html/websluzby.html.
7. CHINNICI, Roberto; MOREAU, Jean-Jacques et al. *Web Services Description Language* [online]. W3C, 2007 [cit. 2021-03-05]. Dostupné z: https://www.w3.org/TR/wsdl/.
8. WESKE, Mathias. *Business Process Management: Concepts, Languages, Architectures*. Postupim, Německo: Springer, 2007. ISBN 978-3-540-73521-2.
9. HE, Hao. *What Is Service-Oriented Architecture* [online]. XML.com, 2003 [cit. 2021-03-05]. Dostupné z: https://www.xml.com/pub/a/ws/2003/09/30/soa.html.
10. PANT, Kapil; MATJAZ, Juric. *Business Process Driven SOA using BPMN and BPEL*. Packt Publishing, 2008. ISBN 9781847191465.
11. ŠLAJCHRT, Zbyněk. *SOA vs. OOP – Jing a Jang v architektuře podnikových aplikací* [online]. blogger.com, 2010 [cit. 2021-03-08]. Dostupné z: http://spitballer.blogspot.com/2010/10/soa-vs-oop-jing-jang-v-architekture.html.
12. PRAGER, Martin; MARŠÍK, Vladimír. Využití orchestrace služeb pro řešení úloh v rámci ISKŘ. *GIS Ostrava* [online]. 2008 [cit. 2021-03-26]. ISBN 978-80-254-1340-1. ISSN 1213-239X. Dostupné z: http://gisak.vsb.cz/GIS_Ostrava/GIS_Ova_2008/sbornik/Lists/Papers/093.pdf.
13. MATKOVIC, J.; FERTALJ, K. Models for the development of Web service orchestrations. *MIPRO, IEEE* [online]. 2012 [cit. 2021-03-26]. Dostupné z: https://ieeexplore-ieee-org.ezproxy.techlib.cz/stamp/stamp.jsp?tp=&arnumber=6240674.

14. BARRETO, Charlton; BULLARD, Vaughn et al. *Web Services Business Process Execution Language Version 2.0* [online]. Oasis, 2007 [cit. 2021-03-11]. Dostupné z: https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm#_Toc166509686.
15. WS-BPEL (Web Services Business Process Execution Language). *ManagementMania.com* [online]. 2015 [cit. 2021-03-26]. ISSN 2327-3658. Dostupné z: https://managementmania.com/cs/ws-bpel-web-services-business-process-execution-language.
16. MARTIN, David; BURSTEIN, Mark et al. *OWL-S: Semantic Markup for Web Services* [online]. W3C, 2004 [cit. 2021-03-26]. Dostupné z: https://www.w3.org/Submission/OWL-S/.
17. ARKIN, Assaf; ASKARY, Sid et al. *Web Service Choreography Interface (WSCI) 1.0* [online]. W3C, 2002 [cit. 2021-03-26]. Dostupné z: https://www.w3.org/TR/2002/NOTE-wsci-20020808/.
18. *Oracle Business Process Management Suite* [online]. Oracle, 2014 [cit. 2021-03-26]. Dostupné z: https://www.oracle.com/technetwork/middleware/bpm/overview/bpm-suite-12c-ds-2264242.pdf.
19. KURZ, Matthias; MENGE, Falko et al. *Diagram Interchangeability in BPMN 2* [online]. Omg, 2021 [cit. 2021-03-26]. Dostupné z: https://www.omg.org/oceb-2/documents/BPMN_Interchange.pdf.
20. KUBÁŇ, Přemysl. *Modelování podnikových procesů* [online]. Altaxo, 2019 [cit. 2021-03-29]. Dostupné z: https://www.altaxo.cz/provoz-firmy/management/rizeni-podniku/modelovani-podnikovych-procesu.
21. ANTONUCCI, Y. L.; BARIFF, M. et al. *Guide to the Business Process Management Common Body of Knowledge BPM-CBOK*. Third. ABPMP, 2013. ISBN 978-1490516592.
22. MACURA, Tibor. *Business Process Mamagement System 2.0* [online]. 2009 [cit. 2021-03-29]. Dostupné z: https://theses.cz/id/2thm16/. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedoucí práce RNDR. JAN PAVLOVIČ, PH. D.
23. BASL, Josef; TUPA, Jiří et al. Technologie pro BPM, Business Process Management Suite. *Magazín BPM* [online]. 2007 [cit. 2021-03-29]. ISSN 1802-5676. Dostupné z: http://bpm-slovnik.blogspot.com/2008/04/bpms.html.
24. *Business Process Modeling Notation Specification* [online]. OMG, 2006 [cit. 2021-03-29]. Dostupné z: https://www.omg.org/bpmn/Documents/OMG_Final_Adopted_BPMN_1-0_Spec_06-02-01.pdf.
25. SHAPIRO, Robert; WHITE, Stephen et al. *BPMN 2.0 Handbook, Digital Edition* [online]. FutStrat, 2012 [cit. 2021-03-29]. Dostupné z: https://www.conradbock.org/white-bpmn2-process-bookmark-web.pdf.
26. *Business Process Model and Notation v2.0.2* [online]. OMG, 2014 [cit. 2021-03-29]. Dostupné z: https://www.omg.org/spec/BPMN/2.0.2/.
27. SILVER, Bruce. *BPMN method and style*. Second. USA: Cody-Cassidy Press, 2009. ISBN 9780982368114.
28. *BPMN 2.0 Symbol Reference* [online]. Camunda, 2021 [cit. 2021-03-29]. Dostupné z: https://camunda.com/bpmn/reference/.
29. *BPMN Quick Guide* [online]. Trisotech, 2021 [cit. 2021-03-29]. Dostupné z: https://www.bpmnquickguide.com/view-bpmn-quick-guide/.
30. *BPMN Notation Overview* [online]. Visual Paradigm, 2020 [cit. 2021-03-29]. Dostupné z: https://www.visual-paradigm.com/guide/bpmn/.

31. WHITE, Stephen A. *XPDL and BPMN* [online]. OMG, 2003 [cit. 2021-04-06]. Dostupné z: https://www.omg.org/bpmn/%7B%5Cnewline%7D/XPDL_BPMN.pdf.
32. *BPMN Model Interchange Working Group* [online]. OMG, 2021 [cit. 2021-04-06]. Dostupné z: https://www.omgwiki.org/bpmn-miwg/doku.php?id=start.
33. CAMUNDA. *Camunda* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://camunda.com.
34. *Automate Any Process Anywhere* [online]. Camunda, 2021 [cit. 2021-03-29]. Dostupné z: https://camunda.com/why-camunda/.
35. HAT, Red. *jBPM* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://www.jbpm.org.
36. FLOWABLE. *Flowable* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://flowable.com/.
37. ALFRESCO. *Activiti* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://www.activiti.org.
38. HEFLO. *Heflo* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://www.heflo.com.
39. UHNAK, Peter; BLIZNICENKO, Jan. *OpenPonk* [online]. 2020 [cit. 2021-03-29]. Dostupné z: https://openponk.org.
40. *OpenPonk* [online]. CCMI, 2016 [cit. 2021-03-29]. Dostupné z: https://ccmi.fit.cvut.cz/nastroje/openponk/.
41. BLIZNIČENKO, Jan. *Podpora simulace a vizualizace v nástroji DynaCASE* [online]. 2015 [cit. 2021-03-29]. Dostupné z: https://dspace.cvut.cz/handle/10467/63136/. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce ING. ROBERT PERGL, PH. D.
42. *About Pharo* [online]. Pharo, 2020 [cit. 2021-03-29]. Dostupné z: https://pharo.org/about.
43. BERGEL, Alexandre. *Agile Visualization*. LULU Press, 2016. ISBN 9781365314094. Dostupné také z: http://AgileVisualization.com.
44. NIKKU. *BPMN 2.0 icon font* [online]. GitHub, 2020 [cit. 2021-04-28]. Dostupné z: https://github.com/bpmn-io/bpmn-font#readme.
45. CAMUNDA. *BPMN.iO* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://bpmn.io.
46. HANKE, Dominik. *BASE64 Image* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://www.base64-image.de.
47. *Business Process Model and Notation* [online]. OMG, 2021 [cit. 2021-04-28]. Dostupné z: https://www.omg.org/spec/BPMN/.
48. BIZAGI. *Bizagi Modeler* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://www.bizagi.com/en/platform/modeler.
49. APACHE. *Maven* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://maven.apache.org/install.html.
50. BERNDRUECKER. *Camunda Spring Boot example* [online]. GitHub, 2020 [cit. 2021-04-28]. Dostupné z: https://github.com/berndruecker/camunda-spring-boot-amqp-microservice-cloud-example/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `Pharo64-win-OpenPonk-BPMN` ..... adresář se spustitelnou formou implementace
  - `OpenPonk-BPMN.exe` ..... soubor pro spuštění implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF