VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

POROVNÁNÍ NÁSTROJŮ PRO MODELOVÁNÍ A VÝVOJ PROCESNĚ ORIENTOVANÝCH APLIKACÍ

COMPARISON OF TOOLS FOR MODELING AND DEVELOPMENT OF PROCESS ORIENTED APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **SAMUEL SPIŠÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Spišák Samuel**
Program: Informační technologie
Název: **Porovnání nástrojů pro modelování a vývoj procesně orientovaných aplikací**
Comparison of Tools for Modeling and Development of Process Oriented Applications
Kategorie: Softwarové inženýrství

Zadání:

1. Seznamte se s dostupnými nástroji pro modelování a vývoj procesně orientovaných aplikací.
2. Vytipujte 3 nástroje, které mají srovnatelné vlastnosti a jsou volně dostupné pro akademické účely. Alespoň jeden z nich by měl pro modelování používat Petriho sítě.
3. Navrhněte vhodnou demonstrační aplikaci tak, aby možné porovnat způsob modelování, vývoje i cílového nasazení aplikace za pomoci vybraných nástrojů.
4. Navrženou aplikaci realizujte za pomoci zvolených nástrojů a výslednou aplikaci otestujte s ohledem na specifikované požadavky.
5. Stanovte kriteria pro porovnání použitých nástrojů a proveďte celkové vyhodnocení zkušeností z tvorby aplikace za pomoci těchto nástrojů.

Literatura:

- Martin Riesz, Martin Seckár, Gabriel Juhás:PetriFlow: A Petri Net Based Framework for Modelling and Control of Workflow Processes. ACSD/Petri Nets Workshops 2010: 191-205

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Práca sa zaoberá porovnaním nástrojou na modelovanie, vývoj a nasadenie procesne orientovaných aplikácií, ktoré ponúkajú spoločnosti Camunda, Bonitasoft a Netgrif. Hlavná časť práce patrí opisu spôsobu vývoja aplikácie navhrnutej na demoštračné účely, ktorá bola vyvinutá v každom nástroji zvlášť. Na základe získaných skúseností z návrhu a implementácie aplikácie v jednotlivých nástrojoch, sú v poslednej kapitole práce zosumarizované ich silné a slabé stránky. Po prečítaní práce by mal mať čitateľ vytvorenú predstavu o spôsobe vývoja v jednotlivých nástrojoch a v prípade záujmu o vytvorenie vlastnej procesne orientovanej aplikácie by mu mala práca pomôcť určiť z porovnávaných nástrojov ten, ktorý by bol na jej implementáciu najvhodnejší.

Abstract

Thesis deals with the comparison of tools for modeling, development and deployment of process-oriented applications. Three tools with a similar approach to the design and development of these applications were selected for comparison. The tools I chose are developed in companies Netgrif, Camunda and Bonitasoft. Main goal of this thesis was to separately develop an application with identical functionality and to describe the development process. Based on acquired experience, thesis summarizes strengths and weaknesses in the approach chosen compared tools. After reading this thesis, the reader should be able to get an idea of the development methods specific to each tool and to choose from compared tools, if interested in creating a process oriented application himself, the one most suitable for his needs.

Kľúčové slová

Petriho siete, Petriflow, BPMN, Camunda, Bonitasoft, Netgrif, Modelovanie podnikových procesov, Procesne orientovaný vývoj, Porovnanie

Keywords

Petri Nets, Petriflow, BPMN, Camunda, Bonitasoft, Netgrif, Bussines process modeling, Process driven development, Comparison

Citácia

SPIŠÁK, Samuel. *Porovnání nástrojů pro modelování a vývoj procesně orientovaných aplikací*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.

Porovnání nástrojů pro modelování a vývoj procesně orientovaných aplikací

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Doktora Ing. Vladimíra Janouška, Ph.D. Uviedol som všeky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Samuel Spišák
11. mája 2021

Poďakovanie

Za odbornú pomoc by som sa chcel poďakovať konzultantom z firmy Netgrif, menovite pánovi Profesorovi Gabrielovi Juhásovi zo Slovenskej Technickej Univerzity, pánovi Milanovi Šamajovi a jednotlivým zamestancom firmy. Za odborné vedenie patrí vďaka pánovi Doktorovi Vladimírovi Janouškovi a v neposlednom rade ďakujem mojej rodine za podporu pri vypracovávaní práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe, kedy sa veľká časť pracovných úkonov zamestnancov už presunula do digitálnej sféry, sa firmy zameriavajú na možnosti zefektívnenia pracovných postupov pomocou automatizácie a na možnosti ich rozšírovania bez obmedzenia iných častí systému, ktorý zastrešuje chod firmy. Nie vždy je to ale v starom systéme možné a v tom prípade je potrebné nájsť nové riešenie, ktoré bude schopné nahradiť funkcie starého systému s možnosťou ich jednoduchého rozšírenia.

V ideálnom prípade by spôsob fungovania celého systému mal pochopiteľný aj pre ľudí, ktorí požiadavky jeho funkcionalitu vytvárajú, nie len pre tých, ktorí ich implementujú. Takýmto spôsobom sa dá predísť nedorozumeniam, kedy si zadávateľ nemusí uvedomiť akým spôsobom bude systém fungovať a nevytvorí si tak realistickú predstavu o náročnosti implementácie jednotlivých požiadaviek. Tieto nedorozumenia následne vedú k predlžovaniu vývoja, neskorému nasadeniu systému do produkcie a zvyšovaniu jeho ceny.

Nástroje na vývoj a nasadenie procesne orientovaných aplikácii, ktoré takéto riešenie môžu predstavovať sa všeobecne zvyknú označovať ako platformy na modelovanie podnikových procesov. Vývoj procesnej aplikácie v nich prebieha iteratívnym spôsobom. Najskôr sa podnikový proces (alebo procesy) vizuálne namodeluje pomocou modelovacieho jazyka. Tento model je vizuálne reprezentovaný diagramom zloženým z viacerých elementov. Tie predstavujú jednotlivé úlohy, rozhodovania a udalosti, ktoré je počas určitého pracovného postupu potrebné vykonať. Keď je model resp. diagram procesu vytvorený, na jednotlivé miesta v diagrame je možné naviazať potrebnú rozhodovaciu a podniková logika, čím je vytvorený prvotný prototyp, pripravený na otestovanie a nájdenie nedostatkov. Celý tento cyklus sa opakuje až pokiaľ implementované procesy nezabezpečia správny chod a funkcionalitu pracovného postupu.

S takýmto nástrojom som sa mal možnosť zoznámiť počas práce v spoločnosti Netgrif[1]. Spoločnosť Netgrif je relatívne mladá firma a platforma, ktorú ponúka, je stále intenzíve vývíjaná. Nezostáva preto jej zamestnancom veľa času na prieskum trhu porovnaním ich riešenia s nástrojmi s obdobným prístupom k vývoju procesne orientovaných aplikácii. Keďže ma tento spôsob vývoja zaujal a zrovna som si hľadal tému na bakalársku prácu, zhodli sme sa vedením na téme mojej práce, ktorá by mi umožnila prehĺbiť znalosti o tomto spôsobe vývoja a pre Netgrif by slúžila ako užitočný prieskum trhu. Priame porovnania spôsobu vývoja vo viacerých nástrojoch tohoto typu nie sú všeobecne skoro vôbec dohľadateľné a tak by mohla mať práca v tomto smere aj spoločenský prínos. Ako dva ďalšie nástroje na porov-

[1] `https://netgrif.com/`

nanie som si vybral riešenie od spoločnosti Bonitasoft[2] a od spoločnosti Camunda[3], keďže obe spoločnosti ponúkajú bezplatné verzie ich platforiem s plnohodnotnou funkcionalitou potrebnou k vývoju a nasadeniu procesnej aplikácie.

V teoretickej časti práce najskôr čitateľa oboznamujem so základnou syntaxou a sémantikou modelovacých jazykou, ktoré nástroje používajú na vizuálnu definíciu procesov.

Táto časť zahŕňa aj predstavenie nástrojov, čo spočíva v opise jednotlivých komponentov, z ktorých sa platformy skladajú a aké funkcie tieto komponenty zabezpečujú.

V ďalšej časti špecifikujem požadovanú funkcionalitu procesnej aplikácie vyvíjanej v každom nástroji. Po nej nasleduje kapitola s opisom vývoja navrhnutej aplikácie v jednotlivých nástrojoch. Opis prebieha po celkoch, ktorých väčšina predstavuje nevyhnutnú sučasť každej procesne orientovanej aplikácie, bez ktorej by ju nebolo možné použiť.

V poslednej časti práce určujem na základe opisu vývoja a skúseností, ktoré som počas neho nadobudol silné a slabé stránky nástrojov, ich vhodné použitie a to, či bola v aplikáciach splnená požadovaná procesná funkcionalita.

[2] `https://www.bonitasoft.com/`
[3] `https://camunda.com/`

Kapitola 2

Teoretická časť

Na uvedenie čitateľa do kontextu problematiky spomeniem hlavné charakterisktiky a výhody využitia platforiem na vývoj procesných aplikácií (BPM platforiem) na implementáciu nového podnikového riešenia s inými možnosťami, aké si pri potrebe automatizovať alebo rozšíriť svoje pracovné postupy môže podnik zvoliť.

Následne v tejto kapitole popisujem modelovacie jazyky 2.1, ktoré mnou vybrané nástroje na modelovanie podnikových procesov používajú. V poslednej sekcii tejto kapitoly 2.2 u každého nástroja opisujem, z akých komponentov sa skladá a akú funkcionalitu tieto komponenty zabezpečujú.

Možnosti podniku pri výbere nového procesného riešenia

A. Zakúpenie existujúcej procesnej aplikácie

Jedna z možností pre ktorú sa môže podnik rozhodnúť je, že si zakúpi už existujúce riešenie s určitou procesnou funkcionalitou. Čas, kým sa systém dodá a zavedie do prevádzky je teda minimálny. Cena je dopredu daná a funkcionalita resp. procesy, ktoré riešenie implementuje tiež.

Nevýhody sú v obmedzenej funkcionalite bez možnosti jej rozšírenia, čo podniku veľakrát nestačí a zároveň vzniká závislosť na podpore od dodávateľa zakúpeného softwéru.

B. Vývoj internej procesnej aplikácie

Druhá možnosť je, že si podnik najme softvérovú firmu, ktorá mu vytvorí riešenie od základov tak, aby spĺňalo všetky špecifické žiadosti podniku.

Takýto prístup je vhodný najmä pri vývoji komplikovaných systémov s integráciou na takzvané "legacy" systémy so zastaralou resp. neštandardnou architektúrou, ktoré majú často obmedzené možnosti integrácie s inými systémami.

Výhody tohto prístupu spočívajú v potenciálne neobmedzenej funkčnosti ktorá bude uspôsobená na daný podnik a jeho potreby. Avšak takýto vývoj je drahý. Na implementáciu a následnú podporu treba zaplatiť vývojárov s vhodnou špecializáciou. Tým, že bude riešenie jedinečné, bude mať aj jedinečný zdrojový kód a prípadné rozširovanie systému môže byť komplikované. Komunikácia medzi vývojármi a zadávateľmi, resp. medzi IT sférou a biznis sférou nie je často transparentná pre obe strany, čím môže dôjsť k nepochopeniu pri nedostatočnej špecifikácii požiadaviek. Vývoj býva dlhý a existuje riziko, že sa za tú dobu potreby podniku zmenia, čo znamená ďalšie predlžovanie a predražovanie vývoja.

C. Vývoj v BPM platforme

Tieto platformy v podstate kombinujú oba vyššie uvedené prístupy. Funkčnosť systému, resp. pracovné postupy sa dopredu namodelujú pomocou modelovacieho jazyka. Vo vytvorených modeloch procesov je jasne vidno, akým spôsobom bude práca postupovať a aké úlohy a udalosti sa budú vykonávať. Jednoduchšie procesy dokážu zaškolení zamestanci podniku vytvoriť aj sami a v prípade potreby je možnosť konzultácie alebo spolupráce s dodávateľom. Vďaka transparentnosti modelov jenotlivých procesov je uľahčená komunikácia medzi IT a biznis sférou, čo urýchľuje vývoj a čas nasadenia do produkcie. Podľa komplexnosti požiadaviek sa určuje aj cena vývoja, no vo väčšine prípadov bude cena nižšia ako pri vytváraní riešenia od základov. Rozširovanie funkcionality je jednoduché a výsledná riešenie je celkovo flexibilné. Pokiaľ však v podniku často nastávajú situácie, ktoré sú nepredvídateľné a nedajú sa dopredu namodelovať, môže byť vhodnejší prístup B.

2.1 Procesné modelovacie jazyky

Každý proces v procesne orientovanej aplikácii je vytváraný na základe modelu, ktorý je reprezentovaný diagramom zloženom z viacerých elementov. Diagramy sú zapísané v procesnom modelovacom jazyku.

Táto práca nemá za úlohu dopdrobna popísať všetky symboly jazykov a každú funkciu ktorú môžu nadobúdať. Avšak z dôvodu pochopenia možností jazykov pri modelovaní rôznych situácii, ktoré v podnikoch bežne nastávajú a sú prítomné aj v mnou navrhnutej aplikácii, je potrebné uviesť ich základné elementy a princíp, akým tieto elementy ovplyvňujú riadenie procesu.

Ako prvý je opísaný zápis *BPMN 2.0* (Notácia pre modelovanie podnikových procesov verzie 2.0), ktorú na modelovanie procesov používajú nástroje od spoločností Camunda a Bonitasoft. Jej kompletnú špecifikáciu, z ktorej pri popise jednotlivých častí jazyka vychádzam, je možné nájsť na zdroji [15]

Spoločnosť Netgrif používa na modelovanie a popis procesov jazyk *Petriflow* založený na Petriho sieťach, ktorý je opísaný v sekcii 2.1.3 a jeho špecifikácia je elektronicky dostupná na zdroji [5].

2.1.1 BPMN 2.0

Prvá verzia tejto notácie z roku 2005 slúžila na modelovanie procesne orientovaných diagramov, ktoré budú pochopiteľne zobrazovať pracovné postupy v podnikoch. S nástupom nástrojov umožňujúcich automatizáciu procesov bola konzorciom OMG (Object Management Group) notácia štandardizovaná, aby mohla byť využitá nielen na zobrazovanie procesov, ale aj na ich namapovanie na technické riešenia a koordináciu [6].

Základné pojmy

Na začiatok uvádzam a vysvetľujem výrazy, ktoré sa budú v práci vyskytovať a súvisia s procesnými princípmi tejto notácie.

Definícia procesu je súbor vo formáte XML, ktorý obsahuje zápis modelu reprezentujúci podnikový proces. Definície procesov, ktoré majú byť spustiteľné sa nasadzujú do aplikácie. Analógiou k definícii procesu môže trieda v objektovo orientovaných programovacích jazykoch.

Inštancia procesu je aktuálne vykonávaný proces, ktorý bol v procesnej aplikácii vytvorený na základe jeho definície a dá sa prirovnať k inicializovanému objektu triedy objektovo orientovaného programovania.

Stav procesu je konkrétna konfigurácia procesu z množiny jeho možných konfigurácii. To zahŕňa aktuálne vykonateľné úlohy a hodnoty jeho dát.

Token Podľa špecifikácie je token „teoretický koncept, ktorý sa používa na uľahčenie vyjadrenia správania sa procesu, ktorý je práve vykonávaný“ [15]. Reprezentuje momentálny stav procesu konkrétnej procesnej inštancie. Počas vykonávania procesu sa token presúva jeho jednotlivými časťami. Keď je token na úlohe, úloha sa môže vykonať.

Pracovný tok Predstavuje vetvu v diagrame procesu, v ktorej sa inštancia procesu práve nachádza. Dá sa povedať, že jeden token predstavuje jeden aktívny pracovný tok. Zo začiatku je v novej inštancii jeden, no môže sa na určítých miestach v diagrame rozdeliť, alebo spojiť s iným tokom do jedného. Viacero tokov reprezentuje viacero úloh, ktoré je možné v procese vykonávať nezávisle od seba v jednom momente.

Popis elementov a symbolov

Základné časti, z ktorých sa model zapísaný pomocou BPMN 2.0 skladá autori príručky k notácii BPMN spoločnosti Bonita [8] zaraďujú do troch kategórii:

- Elementy pracovného toku
- Organizačné elementy
- Anotačné elementy

Elementy pracovného toku

Tieto grafické elementy tvoria kostru celého modelu procesu a majú spojitý, sekvenčný charakter. Týmito elementami prechádza pracovný tok a predstavujú tak vykonateľné časti procesu.

Delia sa na:

- Aktivity/Úlohy
- Spojovacie elementy
- Brány
- Udalosti

Aktivity (alebo úlohy, z angl. *tasks*) reprezentujú vykonanie určitej práce. Tieto elementy bývajú navzájom prepojené spojovacími elementami, ktoré riadia tok práce alebo len naznačujú miesto, kde prebiehajú určité operácie, alebo komunikácia v procese. Hlavný tok sa môže deliť na viacero súbežne vykonávaných tokov, alebo sa len presúvať do vybranej vetvy procesu. Delenie, spájanie a presmerovanie toku sa modeluje pomocou brán, ktoré znázorňujú rozhodovanie v procese. Pokiaľ je potrebné reagovať na určitú aktivitu v procese a koordinovať komunikáciu v procese, použijú sa elementy udalostí.

Aktivity

Označujú sa zaobleným obdĺžnikom a podľa atomickosti vieme aktivity rozdeliť na:

- Úlohy (Atomická aktivita)
- Podprocesy a volania aktivít (Zložená aktivita)

Úloha

Úloha reprezentuje samostatnú jednotku práce ktorá sa bude vykonávať. Vykonávateľ danej úlohy sa vyznačuje v ľavom hornom rohu elementu.

Ak je úloha manuálna a teda vykonávaná určitou poverenou osobou, nazýva sa úloha užívateľa (označenie User task).

V druhom prípade môže byť úloha vykonávaná automaticky, buď vzdialenou službou, ktorú procesná aplikácia zavolá ale neimplementuje (označenie Service task)[1], alebo reprezentuje vykonanie určitej funcionality implementovanej priamo v procesnej aplikácii (označenie scripTask)[2]

Podproces

V diagrame reprezentuje synovský proces, ktorý zdieľa s otcovským procesom dáta. Vyznačuje sa symbolom „+“ v spodnej časti tohto elementu.

Aby sa mohol token posunúť v otcovskom procese ďalej z aktivity podprocesu, musí sa najskôr celý synovský proces vykonať a teda token musí najskôr prejsť celým procesom,

[1] napríklad volanie webovej služby (web service)

[2] funkcionalita môže byť implementovaná Java triedou v procesnej aplikácii, alebo skriptom, ktorý je zapísaný priamo v jej definícii

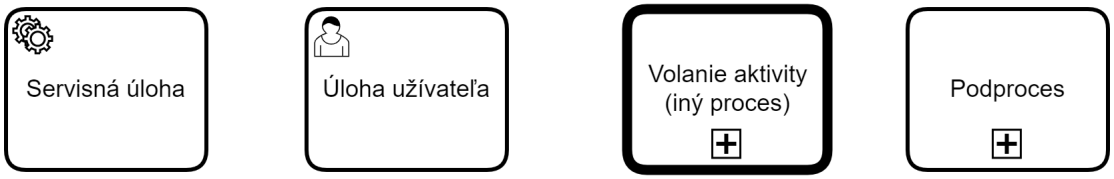
ktorý podproces v grafe reprezentuje. Má za úlohu vizuálne zjednodušiť graf otcovského procesu.

V prípade prerušovaného okraju je aktivácia podprocesu podmienená nejakou udalosťou, ktorá môže nastať v priebehu vykonávania otcovského procesu. Od bežného podprocesu sa líši tým, že z neho, ani do neho žiadne spojovacie elementy nevedú a neovplyvňuje tak priamo tok práce otcovského procesu.

Volanie aktivity
Vyznačuje sa rovnako ako podproces, s tým rozdielom, že okraj elementu je zhrubnutý.

Zo sémantického hľadiska sa od podprocesu líši tým, že sa volá vzdialený proces, ktorý nie je súčasťou definície otcovského procesu. Tieto procesy majú samostatný kontext a pokiaľ je potreba nejaké dáta z otcovského procesu použiť aj vo volanom procese, musí sa tak urobiť explicitne.

Výhoda použitia volania aktivity spočíva v jej znovupoužiteľnosti a modularite procesu, ktorý ju obsahuje. Pokiaľ v podniku existujú procesy, ktoré sú súčasťou viacerých pracovných postupov, oplatí sa tieto procesy nasadiť samostatne a v pracovných postupoch ich vzdialene volať pomocou volania aktivity. Týmto spôsobom sa pri zmene logiky vo volanom procese nemusia meniť všetky modely pracovných postupov, stačí upraviť a nasadiť novú verziu vzdialene volaného procesu.



Obr. 2.1: Príklad atomických a zložených aktivít

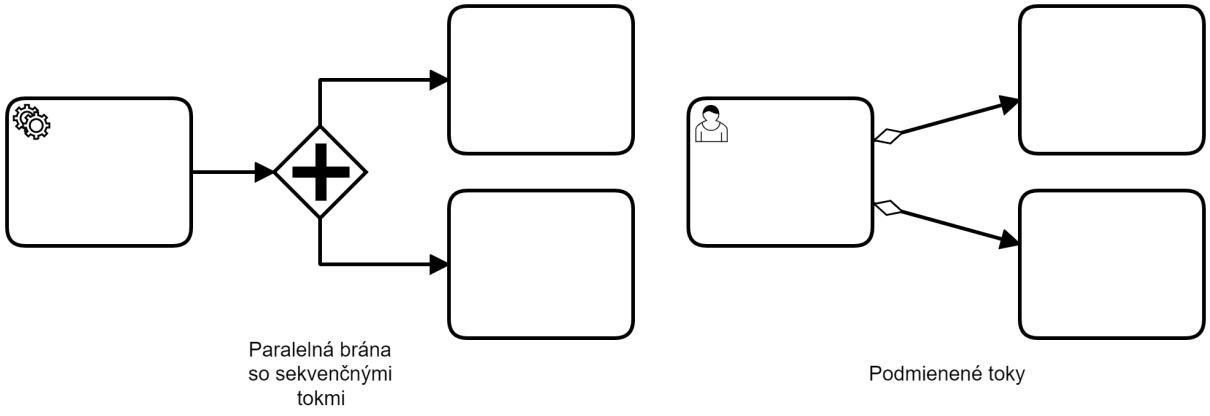
Spojovacie elementy
Konkrétne nazývané ako *sekvenčné alebo asociačné toky*. Tieto elementy prepájajú navzájom všetky aktivity, úlohy a brány. Taktiež môžu mať len vizuálnu funkciu, napríklad na vyznačenie dátových operácii a komunikácie v procese.

Sekvenčné toky Smerujú a riada pracovný tok. Značia sa šipkami s plnou čiarou. Podľa toho, koľko sekvenčných elementov z aktivity vychádza sa rozošle odpovedajúci počet tokenov ďalej do procesu a vykonávanie ďalších úloh v procese sa deje súbežne.

Môžu mať na sebe definovanú podmienku, ktorá musí byť splnená na to, aby sa token v procese posunul ďalej po danom sekvenčnom toku.[3] Kombinujú sa s bránami.

Asociačné toky sú v modeli sú znázornené prerušovanou čiarou. Používajú sa na znázornenie komunikácie ktorá medzi dvoma elementami, alebo na znázornenie vstupno-výstupných a datových operácii, no neovplyňujú priamo pracovný tok a poradie vykonávania aktivít.

[3]Pokiaĺ z elementu vychádza viac sekvečných tokov a viacero spĺňa definované podmienky prechodu, odošle sa na každý tok jeden token.



Obr. 2.2: Príklad tokov

Brány
Existujú tieto typy brán:

- Exkluzívne (XOR) brány
- Inkluzívne (OR) brány
- Paralelné brány
- Udalosťami podmienené brány

Exkluzívna brána (XOR gateway)
Slúži na znázornenie rozhodovania v procese. Pri prechode exkluzívnou bránou sa pomyselný token resp. tok procesu pohybuje ďalej len po tom sekvenčnom elemente, ktorého podmienka sa vyhodnotí ako splnená.[4] V prípade. že by mohla nastať situácia, kedy sa ani jedna z podmienok nevyhodnotí ako splnená, musí z brány vychádzať aj takzvaný *predvolený* tok, cez ktorý následne prejde token. Prichádzajúce tokeny na bráne nečakajú a sú vždy púšťané ďalej.

Paralelná brána (Parralel gateway)
Slúžia na paralelizáciu vykonávania úloh v procese. Ich funkcia spočíva v rozdeľovaní a spájaní pracovného toku, resp. vo vytváraní nových tokenov v odchádzajúcom smere a zľučovaní viacerých tokenov do jedného v prichádzajúcom smere. Na pripojené toky k bráne sa nedefinujú podmienky.

Pokiaľ sa do brány nedostanú tokeny zo všetkých prichádzajúcich tokov, brána sa neaktivuje.

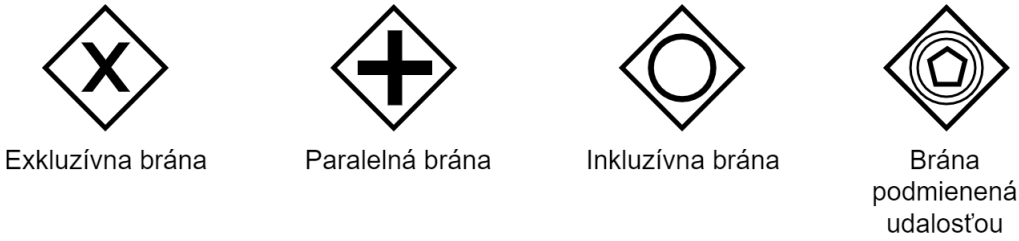
Inkluzívna brána (Inclusive gateway)
Môžeme o tejto bráne uvažovať ako o kombinácii paralelnej a exkluzívnej brány. Ku nej pripojené toky môžu mať definované podmienky prechodu. Na rozdiel od exkluzívnej brány sa podľa počtu splnených podmienok odošle odpovedajúci počet tokenov, po každom toku

[4]Pri viacerých možnostiach prechodu sa token pošle po tom sekvenčnom elemente, ktorého splnenie podmienky sa vyhodnotí ako prvé.

so splnenou podmienkou jeden. Ak sa ani jedna podmienka nesplnení, mal by byť k bráne pripojený predvolený tok, tak ako v prípade exkluzivnej brány.

Aktivácia brány závisí na tom, koľko tokov do nej vstupujúcich je aktívnych. Pokiaľ tokeny zo všetkých aktívnych tokov prišli, aktivuje sa. Z implementačného hľadiska to znamená, že nástroj s možnosťou použiť tento typ brány by mal podporovať určitú synchronizáciu tokenov.

Udalosťami podmienené brány (Event gateway)
Na rozdiel od ostatných typov brán, podmienka na aktiváciu odchádzajúceho toku nie je určená hodnotou nejakej premennej, ku ktorej sa dá v momente príchodu pracovného toku do brány pristúpiť. Prechod je v tomto prípade podmienený udalosťou, ktorá v priebehu procesu môže nastať. O udalostiach píšem v sekcii 4.10. Tokeny, ktoré do brány vstupujú sú púšťané bez rozdielu.



Obr. 2.3: Typy brán

Udalosti
Existuje mnoho typov udalostí a spôsobov ich využitia, preto nebudem zachádzať do príliš veľkých detailov.

Z pohľadu umiestnenia udalostí v procese sa delia na:

- Počiatočné udalosti (Start events) - vytvárajú tok práce
- Ukončujúce udalosti (End events) - rušia tok práce[5]
- Priebežné udalosti - riadia tok práce

Priebežné udalosti môžu mať prerušovaný okraj. Vtedy sú neprerušujúce (non-interrupting events), čo znamená, že pracovný tok, ktorý ich aktivoval nie je nimi ovplyvňovaný. Podľa toho, či udalosť akciu vytvára alebo na ňu reaguje sa dajú udalosti rozdeliť na:

- Zachytávajúce udalosti (Catching events)
- Vyvolávacie udalosti (Throwing events)

Zachytávajúce udalosti
Aktivácia Zachytávajúce udalostí závisí buď od splnenia podmienky v bode procesu v ktorom sa volaná udalosť nachádza, alebo od vyvolávacej udalosti, ktorá je s ňou spárovaná[6].

Typy volaných udalostí:

[5]Treba poznamenať, že obyčajná ukončujúca udalosť ruší len jeden pracovný tok. Pokiaľ je v procese viacero aktívnych tokov, celý proces sa neukončí. Na ukončenie všetkých aktívnych pracovných tokov existuje typ ukončujúcej udalosti označovanej ako *Terminate end event*

[6]Môže byť aj počiatočná, ktorá začína nový proces alebo udalosťou podmienený podproces

- Správa[7] (Message)
- Signál (Signal event)
- Podmienená udalosť (Conditional event)
- Časovač (Timer event)
- Výnimka (Error event)

Volaná udalosť správy čaká na prijatie pre ňu určenej správy od vyvolávacej udalosti. Na rozdiel od správy, pri signáli môže byť volaných udalostí čakajúcich na ten istý signál viac. Signál sa teda využíva pri potrebe spúšťania viacerých tokov naraz.
Podmienená udalosť a časovač nečakajú na vyvolávaciu udalosť, ale na príchod tokenu na ich miesto v procese.[8]

Podmienená udalosť aktivuje k nej pripojený sekvenčný tok, pokiaľ je jej podmienka v danej chvíli splnená.

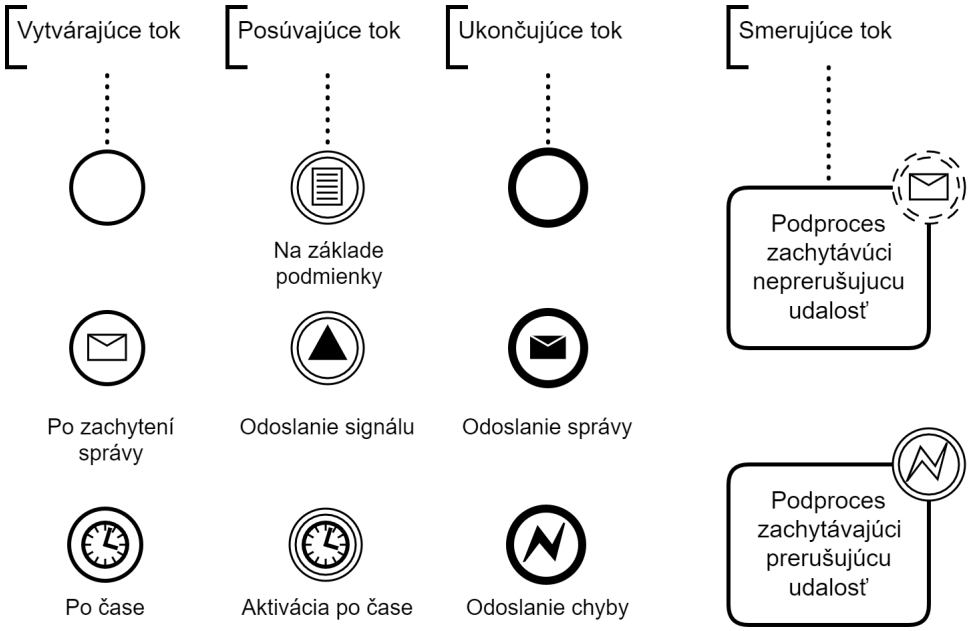
Časovač po príchode tokenu spustí odpočítavanie a pokiaľ sa po uplynutí času token na mieste časovača stále nachádza, aktivuje sa k časovaču pripojený odchádzajúci sekvenčný tok.

Výnimka obsluhuje chyby resp. výnimočné situácie, ktoré môžu nastať v priebehu procesu. Pokiaľ sa pomocou udalosti výnimka vyvolá, aktivuje sa tok vychádzajúci z udalosti zachytávajúcej túto výnimku.

Vyvolávacie udalosti (Throwing events)
Základné vyvolávacie udalosti su *správa*, *signál* a *výnimka*.
Pri vstupe tokenu do udalosti vyvolávacej správu je konkrétnemu prijímateľovi resp. miesto v procese odoslaná správa, pričom token pokračuje dalej po odchádzajúcom toku.
Podobne funguje aj signál, akurát nie je adresovaný konkrétnemu prijímateľovi. Všetky vyvolávané udalostí v aplikácii, ktoré na daný signál čakájú sa aktivujú.
Po príchode tokenu do udalosti vyvolávacej výnimku sa aktivuje volaná udalosť obsluhujúca výnimku. Udalosť vyvolávajúca výnimku sa modeluje ako konečná udalosť, čiže token po príchode do nej zmizne a daný pracovný tok v nej končí.

[7] Správy odosielajú informácie o určitej udalosti medzi dvoma miestami v procese, alebo medzi dvoma procesmi. Správy v tomto zmysle nereprezentujú mailové správy

[8] V prípade, že sú zároveň počiatočnými udalosťami, vytvárajú po splnení určitej podmienky resp. po uplynutí určenej doby novú inštanciu procesu.



Obr. 2.4: Typy udalostí. Zľava počiatočné, priebežné, ukončujúce a priradené úlohám

Organizačné elementy

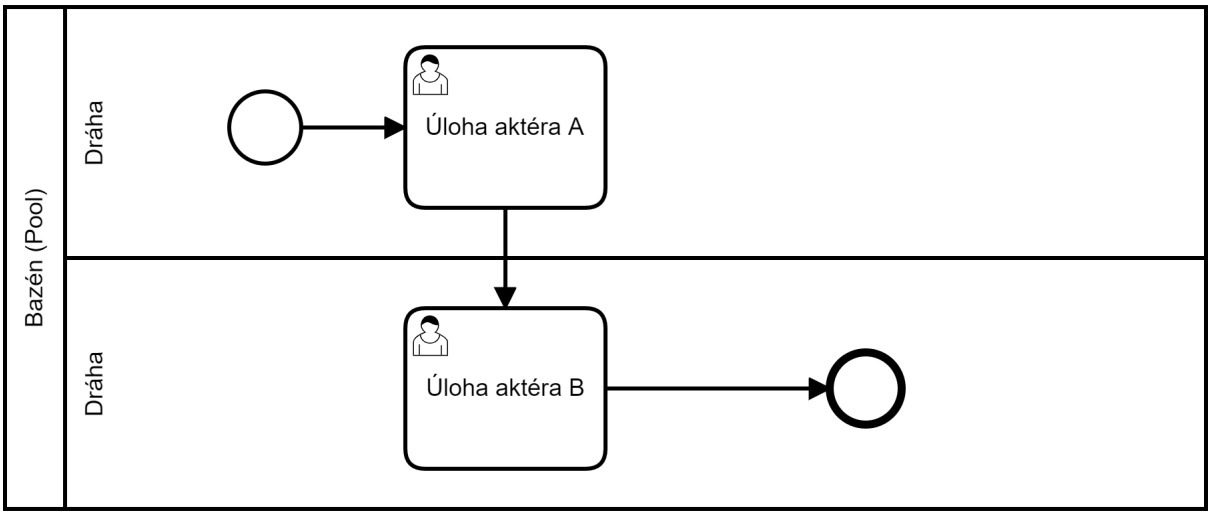
Pre potrebu čitateľnosti modelu a vyznačenia aktérov je možné použiť organizačné elementy *bazén (Pool)* a *dráha (lane)*

Bazény obsahujú jednotlivé procesy resp. vyznačujú všetky aktivity potrebné k vykonaniu k určitej práce v podniku. Komunikácia medzi bazénmi prebieha spravidla pomocou udalostí, spojitý tok práce nesmie presahovať hranice bazénu.

Môžu obsahovať viacero dráh, ktoré vyznačujú jednotlivých aktérov alebo ich skupiny a im prislúchajúce aktivity v procese.

Dráha vyznačuje jedného z aktérov ich skupiny, ktorí sa procesu zúčastňujú. Všetky aktivity v dráhe by mal vykonávať ten istý aktér. Dráha môže byť použitá aj na vyznačenie automaticky aplikáciou vykonávaných aktivít, kedy je aktérom systém.

Tok práce môže na rozdieľ od bazénov presahovať hranice dráhy, ako vidno na obrázku2.5.



Obr. 2.5: Bazén s dvoma dráhami

Anotačné elementy

V diagrame zobrazené prerušovanou čiarou, tieto elementy nemajú za úlohu riadiť pracovný tok, len ho vizualizovať. Zvyknú sa nimi spájať volacie a im príslušné zachytávacie udalosti, aby bolo jasné, akým spôsobom bude pokračovať vykonávanie procesu po vyvolaní danej udalosti. Taktiež môžu naznačovať typy operácii (napríklad dôležité databázové operácie), ktoré na mieste vyznačenia prebiehajú, prípadne doplniť slovný popis k elementom.

2.1.2 Rozhodovací model a Notácia

Z porovnávaných nástrojov tento štandard podporuje riešenie od spoločnosti Camunda. V angličtine známe pod názvom „Decision Model and Notation“ (DMN), je konzorciom OMG štandardizovaný modelovací jazyk a notácia na vytváranie rozhodnutí na základe podnikových pravidiel. Je navrhnutý tak, aby bol kompatibilný s modelmi BPMN 2.0.

Rozhodovacie tabuľky

Slúžia na opis rozhodovacieho procesu. Stĺpce v tabuľke reprezentujú vstupné a výstupné premenné. Riadky reprezentujú rozhodovacie pravidlá. Každé pravidlo má v okienku prislúchajúcemu vstupnej premennej interval akceptovaných hodnôt. To sú podmienky rozhodovania. Okienko riadku prislúchajúce výstupnej premennej určuje hodnotu, ktorú táto premenná nadobudne pri splnení všetkých podmienok v danom riadku.

V oficiálnej špecifikácii štandardu DMN [16] sa môže čitateľ oboznámiť s implementačnými detailami jeho rozhodovacej logiky.

2.1.3 Petriflow

Jazyk Petriflow je "high-level" programovací jazyk na vývoj procesných aplikácii. Vykonávanie v ňom definovaných procesov je založené na princípe fungovania Petriho sietí. Petriho siete sú na rozdiel od notácie BPMN matematicky formalizované.

Matematický formalizmus má z hľadiska procesných modelov svoje výhody, ako je napríklad jednoducho overiteľná dostupnosť jednotlivých častí siete. Bohužiaĺ, vizuálna reprezentácia pracovného toku podnikového procesu a všetkých priebežných udalostí, ktoré počas jeho vykonávania nastávajú, je z dôvodu potreby zachovať matematický formalizmus Petriho sietí neprehľadná. Táto neprehľadnosť následne zvyšuje riziko vytvorenia chybného modelu. O tom, ako nedostatky Petriho sietí pri modelovaní podnikových procesov modelovací jazyk Petriflow rieši, opisujem v sekcii 2.1.3.

Petriho siete

Aby mohla byť pochopená sémantika jazyka Petriflow, uvádzam základné definície a syntaktickosémantickú logiku Petriho sietí.

Definícia Petriho sietí
Petriho siete navrhol v roku 1962 Adam Petri na popis chemických procesov [19]. Definící Petriho siete existuje viacero. Uvediem tú, ktorú uvádza Juhás a Desel vo svojej knihe [13] a ktorú jazyk Petriflow priamo rozširuje.

Petriho sieť je pätica $P = (S, T, F, W, M_0)$, pričom:

- S je konečná množina miest
- T je konečná množina prechodov
- F je konečná množina hrán, kde $F \subseteq (S \times T) \cup (T \times S)$
- $W : F \to \mathbb{N}^+$ je množina váh, ktoré každej hrane $f \in F$ priraďujú číslo $n \in \mathbb{N}^+$ označujúce počet tokenov, ktorý bude odobraný z miesta, z ktorého hrana f vychádza, resp. počet tokenov, ktorý bude pridaný na miesto do ktorého hrana vchádza.
- $M_0 : S \to \mathbb{N}$ je počiatočné značenie, kde v každom mieste $s \in S$ je $n \in \mathbb{N}$ značiek, ktoré sa nazývajú *tokeny*[9].

Graf Petriho siete
Je to bipartitný orientovaný graf. Skladá sa z miest a prechodov, čo sú uzly grafu. Taktiež obsahuje orientované hrany, ktorí tieto uzly spájajú. Tento základný typ Petriho siete sa zvykne nazývať *PT net*.

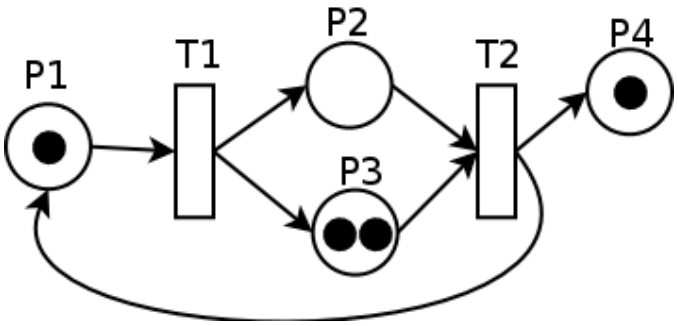
Miesta sa označujú kruhom. Môže do nich aj z nich viesť 0 až N hrán. Slúžia na spájanie prechodov. Hrana nemôže viesť z jedného miesta do druhého. Z pohľadu konkrétneho prechodu môžu byť miesta *vstupné* alebo *výstupné*. Vstupné miesto je pre daný prechod vtedy, keď hrana z miesta vychádza a v tomto prechode končí. Výstupným miestom je miesto naopak vtedy, keď v ňom hrana končí a z daného prechodu vychádza.

[9]Tokeny sú reálnou súčasťou Petriho sietí, ktoré ovplyňujú jej vykonácanie, na rozdiel od tokenov v notácii BPMN, kde sú teoretickým konceptom. ktorý reprezentuje stav procesu

Prechody sa zvyknú značiť štvorcom, alebo obdĺžnikom. Spájané môžu byť len s miestami a na rozdiel od miest sú aktívnym prvkom siete. To znamená, že sa môžu po splnení podmienky prechodu[10] aktivovať a presunúť počet tokenov určený váhami hrán medzi miestami, s ktorými sú spojené. Pri nesplnení podmienok na aktiváciu sú v *neaktívnom* stave.

Hrany sú značené šípkami. Spájajú miesta a prechody. V prípade začiatku hrany v mieste a jej konci v prechode, určuje váha hrany počet tokenov, ktoré sa z daného miesta z ktorého vychádza odoberú. Pokiaľ hrana vychádza z prechodu a končí v mieste, do daného miesta sa po aktivácii prechodu počet tokenov odpovedajúci váhe hrany pridá.

Simulácia Petriho siete
Pred spustením je sieť v počiatočnom stave M_0. Následne sa určí, či existujú prechody do ktorých smerujú hrany z miest, ktoré majú počet tokenov odpovedajúci, alebo vyšší ako je váha danej hrany. Pokiaľ áno, *podmienka prechodu* pre dané prechody je splnená, prechody sa môžu aktivovať a presunúť počet tokenov odpovedajúci váham hrán medzi miestami. Tento proces sa môže opakovať, až pokým nezostávajú prechody, ktoré by sa mohli aktivovať.[11]



Obr. 2.6: Príklad Petriho siete. Prebraté z [4]

Popis jazyka Petriflow

Pre potreby procesného modelovania jayzk Petriflow rozširuje *PT sieť* o ďalšie komponenty, ako sú ďalšie typy hrán [17], ktoré môžu smerovať len z miest do prechodov a nie naopak.

- Read hrany
- Reset hrany
- Inhibítorové hrany

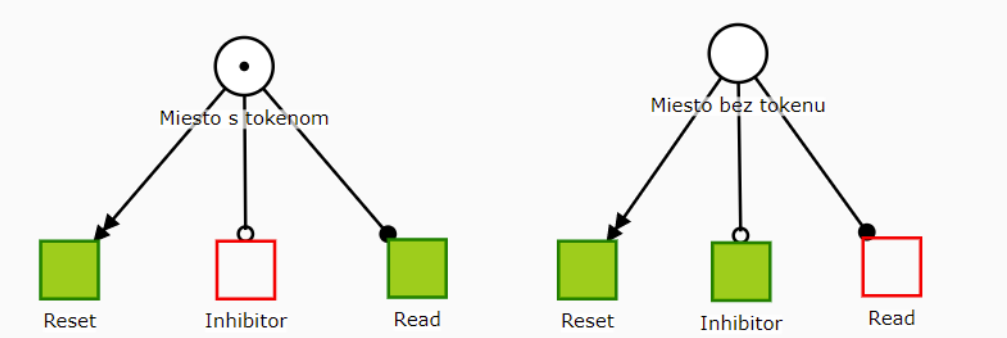
Read hrana slúži na neobmedzený počet aktivácii prechodu pokiaľ je jeho podmienka prechodu splnená. Po jeho aktivácii vo vstupnom mieste ostáva počet tokenov rovnaký. Najčastejšie sa používa na úlohy zobrazenia dát v rôznych fázach procesu.

[10] *Podmienku prechodu bližšie špecifikujem nižšie v časti „Simulácia Petriho siete“*

[11] Toto je len náčrt, akým spôsobom prebieha simulácia Petriho sietí. Spôsob vykonávania simulácie sa od jednotlivých simulačných nástrojov môže líšiť.

Reset hrana umožňuje zo vstupného miesta odobrať všetky tokeny, pričom jej váha má hodnotu 0 a tak vie aktivovať prechod v každom stave procesnej inštancie. Využíva sa pri rušení procesu.

Inhibítorová hrana aktivuje prechod do ktorého vchádza len vtedy, pokiaľ miesto z ktorého vychádza neobsahuje tokeny. Vďaka tomu sa dajú blokovať určité prechody.



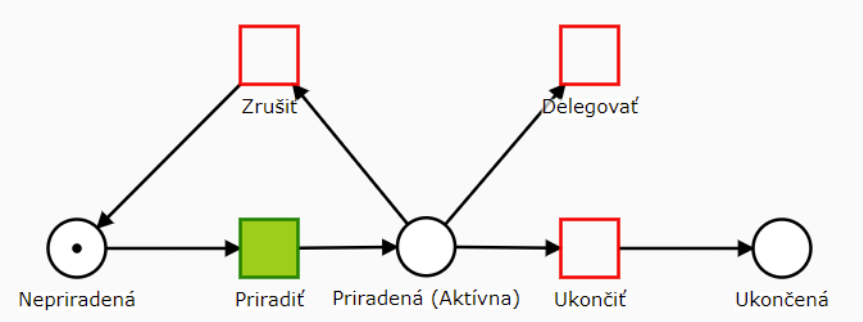
Obr. 2.7: Správanie špeciálnych typov hrán

Reprezentácia úlohy v podnikovom procese
Podnikové úlohy sú vo väčšine nástrojou na modelovanie podnikových procesov reprezentované ako jeden atomický prechod. Na nižšej úrovni pohľadu sa však skladajú z viacerých fáz, resp. podúloh, ktoré musia byť vykonané, aby sa úloha vykonala [20]. Zvyčajne sú to fázy:

- Priradenia úlohy
- Začiatku vykonávania
- Zrušenia vykonávania
- Ukončenia vykonávania

Na umožnenie vykonávania spomenutých podúloh je každý prechod v sieti vytvorenej v modelovacom jazyku Petriflow, ktorý reprezentuje úlohu v procese, sám o sebe Petriho sieťou, ako je možno vidieť na obrázku 2.8. Každá úloha je teda podproces. Tento spôsob implementácie zvyšuje mieru vizuálnej abstrakcie siete, čo zvyšuje jej čitateľnosť a je umožňuje na jednotlivé fázy vykonávania úlohy priradiť akcie, ktoré vykonávajú automatizáciu. Fázy prechodu môžu byť nasledovné [17]:

- Priradenie - konzumácia tokenu na vstupnom mieste a jeho vloženie do miesta „Priradená“, ktorom token zotrváva počas vykonávania práce užívatelom
- Zrušenie - zruší vykonávanie úlohy a vracia token do začiatočného miesta úlohy
- Delegácia - preradenie úlohy na iného užívateľa
- Ukončenie - ukončuje úlohu a presúva token z miesta „Priradená“ na miesto „Ukončená“



Obr. 2.8: Jednotlivé prechody úlohy

Procesy sú v jazyku Petriflow reprezentované XML štruktúrou, ktorá obsahuje jednotlivé elementy diagramu Petriho siete. Kvôli potrebám procesného modelovania podnikových procesou je táto štruktúra obohatená o ďalšie tri vrstvy:

- Role
- Dáta
- Akcie

Role predstavujú skupiny užívateľov resp. aktérou v procese. Sú definované na začiatku súboru a následne sa môžu priradiť jednotlivým úlohám pomocou referencie do XML štruktúry úlohy. Takto sa určí, kto môže danú vykonávať.

Procesné dáta sú taktiež definované na začiatku súboru. Referencia dát, ktoré majú byť v konktrétnej úlohe viditeľné, sa vložia do XML štruktúry úlohy, podobne ako pri rolách.

Akcie sa od oboch spomenutých vrstiev líšia tým, že sa nezapisujú vo formáte XML, ale v špecifickom jazyku vytvorenom na potreby pridania automatizácie do procesu. Tento jazyk vychádza z objektovo orientovaného programovacieho jazyka Groovy[12]. Aj keď tieto akcie nie sú definované štruktúrov XML, sú súčasťou súboru XML definujúceho proces. Špeciálne tagy určujúce fázu, kedy sa má akcia vykonať, sa pridajú do štruktúry k určitej úlohe, ktorú proces obsahuje.

[12] `https://groovy-lang.org/`

Príklad štruktúry prechodu (úlohy) v procese

```
<transition>
    <id>14</id>
    <x>480</x>
    <y>350</y>
    <layout>
        <offset>0</offset>
    </layout>
    <label>Reject / Loan officer</label>
    <icon>close</icon>
    <priority>2</priority>
    <assignPolicy>auto</assignPolicy>
    <roleRef>
        <id>loan_officer</id>
        <logic>
            <perform>true</perform>
        </logic>
    </roleRef>
    <event type="finish">
        <id>14_finish</id>
        <actions phase="pre">
            <action>
            <!-- @formatter:off -->
            decision: f.status;
            change decision value {"Mortgage Rejected"}
            <!-- @formatter:on -->
            </action>
        </actions>
    </event>
</transition>
```

2.2 Komponenty vybraných nástrojov

Všetky mnou vybrané BPM platformy sa skladajú z viacerých súčastí. Cieľom tejto práce nie je dopodrobna vysvetliť ich fungovanie, ale je potreba spomenúť informácie o tom, akú majú funkciu a využitie.

2.2.1 Camunda

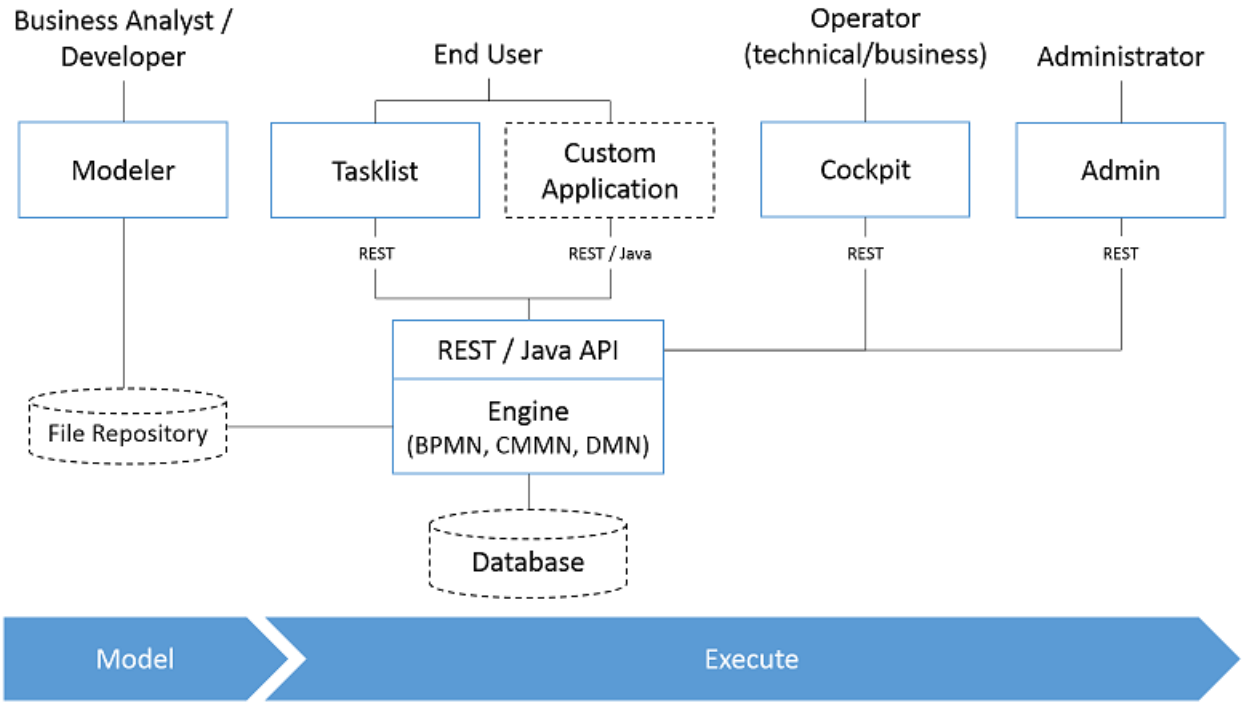
BPM platforma od spoločnosti Camunda sa dá opísať ako open-source, kompozitné zoskupenie BPM nástrojov, určených pre vývoj procesne orientovaných aplikácii v programovacom jazyku Java [14, 329].

Jej základom je procesný (workflow) engine, ktorý spúšťa a riadi nasadené procesy. Platforma obsahuje aj engine na spúšťanie rozhodovacích tabuliek štandardu DMN (Decision model and Notation).

Camunda taktiež ponúka nástroje plne integrovateľné s procesným a DMN enginom. Všetky súčasti platformy a dodatočné nástroje na vývoj a monitorovanie aplikácie sú zrhnuté v tomto zozname:

- Modelovanie:
  - Modeler
  - Cawemo (plná funkcionalita vo verzii Enterprise)
- Beh aplikácie:
  - Procesný engine
  - DMN engine
- Vykonávanie užívateľských úloh a správa aplikácie:
  - Tasklist
  - Cockpit (plná funkcionalita vo verzii Enterprise)
  - Admin
  - Optimize (len v Enterprise verzii)

Komunitná verzia obsahuje všetky potrebné súčasti a aplikácie na vývoj a nasadenie procesnej aplikácie. Enterprise verzia pridáva resp. rozširuje možnosti jednak kolaboratívneho modelovania, ale najmä analýzy a optimalizácie procesov.



Obr. 2.9: Komponenty BPM platformy od spoločnosti Camunda. Prevzaté z [3]

Modeler

Modeler je dostupný na stiahnutie zadarmo ako desktopová aplikácia[13]. Umožňuje modelovanie procesov v notácii BPMN 2.0 a vytváranie rozhodovacích tabuliek podľa štandardu DMN napojiteľných na úlohu rozhodovania v diagrame procesu[14]. Je v ňom dostupný pohľad aj na XML štruktúru modelovaného procesu.

Implementačné parametre sa v modeleri dajú nastaviť priamo jednotlivým elementom v diagrame. Uľahčuje je to elementy napojiť na podnikovú logiku aplikácie a umožňuje aj priamo definovať automatizáciu, ktorá sa má v danom stave procesu vykonať. Medzi najdôležitejšie parametre patrí:

- Napojenie na triedu implementujúcu prácu automatizovanej úlohy
- Určenie akcii v jednotlivých logických fázach úlohy
- Definícia skriptu[15], ktorý úloha spúšťa
- Definícia vstupných a výstupných parametrov úlohy
- Použitie výrazového jazyka (napr. v podmienkach prechodu)

[13] `https://camunda.com/download/modeler/`

[14] Tatiež podporuje vytvárať model správy a správy prípadov (CMMN), ale tento spôsob modelovania Camunda z dôvodu malej využiteľnosti už neaktualizuje [12].

[15] Natívne podporovaný skriptovací jazyk *Groovy*

Modeler je možné rozšíriť doplnkami vytváranými komunitou, ktoré sú dostupné k stiahnutiu na distribuovanom systéme riadenia revízii Github.[16] Príkadom doplnku je základná simulácia tokenov v modeli procesu, čo je možné použiť na zlepšenie predstavy o fungovaní procesu a jeho testovanie priamo v modeleri.

Cawemo

Hlavnou funkciou tejto webovej aplikáci je kolaboratívne modelovanie diagramu procesu. Rozlišuje tri úrovne práv zásahov do kolaboratívne vytváraného modelu:

- Zobrazenie diagramu
- Komentovanie diagramu
- Úprava diagramu a pozývanie účastníkov ku kolaborácii

V enterprise verzii obsahuje ďalšie funkcie ako vytváranie a porovnávanie cieľov, vytváranie zdieľateľného katalógu s modelmi procesov, vytváranie predlôh vo formáte JSON na rozšírenie funkcionality modeleru potrebnej pre špecifickú doménu s možnosťou synchronizácie s lokálnym modelerom a spravovanie kľúčov potrebných k integráciam (vrátane integrácii s modelerom a procesným enginom).

Podrobnejšie sa o funkciách dá dočítať v oficiálnej dokumentácii[17].

Procesný engine

V angličtine *Workflow engine*, je jadrom celej BPM platformy. Má za úlohu spúšťanie procesov a ich riadenie. Rozhoduje, ktoré úlohy a volania vzdialených služieb sa v procese majú vykonať a za akých podmienok sa majú vykonať. Na základe ich výstupu určuje následné kroky v procese. Je kombináciou riadenia pracovného toku a aplikačnej integrácie. Zároveň slúži na monitorovanie procesov, čo sa dá využiť pri ich optimalizácii [14].

Do aplikácie sa môže pridať samostatne ako knižnica v Maven[18] konfigurácii. Na prístup k jeho funkcionalite sa používa API (rozhranie pre programovanie aplikácie), ktoré je definované v oficiálnej dokumentácii[19].

Podporuje pripájanie cez REST API, čo je vhodné pri záujme mať aplikáciu vzdialene prístupnú s oddeleným používateľským rozhraním od aplikáčnej logiky.

Na rozdiel od Enterprise verzie, ktorá dostáva priebežné aktualizácie, v komunitnej verzii je engine aktualizovaný len raz za 6 mesiacov.

Engine môže byť z pohľadu architektúry vyvíjaného riešenia implementovaný viacero spôsobmi [11]:

Vstavaný v aplikácii čím bude súčasťou aplikácie ako Java knižnica, čo je vhodné pri vývoji samostatnej procesnej aplikácie, ktorá bude nasadená s enginom v jednom kroku.

[16] `https://github.com/camunda/camunda-modeler-plugins#camunda-modeler-plugins-electric_plug`

[17] `https://docs.camunda.org/cawemo/latest/user-guide/`

[18] `https://maven.apache.org/`

[19] `https://docs.camunda.org/manual/7.14/user-guide/process-engine/process-engine-api/`

Ako služba v rámci aplikačného serveru na ktorú bude pristupovať viacero procesných aplikácii nasadených na danom aplikačnom serveri. Tento prístup je vhodný, pokiaľ chce mať podnik viacero samostatných procesných aplikácii na rôzne typy podnikových procesov s centrálnou zdieľanou riadiacou jednotkou resp. centrálnym procesným enginom. Týmto spôsobom sa oddelí vývoj aplikácii od konfigurácie procesného enginu.

Samostatne nasadený procesný engine na serveri, na ktorý sa bude pristupovať vzdialene (najjednoduchšie cez vstavané REST API). Správa enginu tak prebieha úplne oddelene od procesných aplikácii, ktoré ho využívajú, čo môže byť potrebné z bezpečnostných dôvodov.

Enginy pracujúce v zhluku so spoločnou databázov pre podniky poskytuje zvýšenú škálovateľnosť a zníženie rizika nedostupnosti služby.

DMN engine

Spúšťa rozhodovacie tabuľky v procese. Je na neho možné pristupovať cez REST API[20]. Môže byť pridaný do aplikácie ako knižnica v konfigurácii Maven a jeho funkcionalita tak môže byť použitá v prípade potreby samostatne. Na prácu s ním je definované Java API dostupné v oficiálnej dokumentácii[21].

Tasklist

Webová aplikácia slúžiaca na spracovanie užívateľských úloh a ako predvolené užívateľské rozhranie ktoré je súčasťou BPM platformy. Všeobecne slúži na vykonávanie a spravovanie úloh, ktoré potrebujú zásah užívateľa.

Medzi hlavnú funkcionalitu patrí:

- Manuálne spúšťanie procesov
- Preberanie úloh alebo ich delegácia na iného užívateľa
- Vykonávanie úloh a vypĺňanie ich formulárov
- Zobrazenie úloh podľa filtrov[22]
- Nastavovanie termínov dokončenia pre úlohy

Cockpit

Slúži na monitorovanie nasadených procesov. Vie zobraziť prebiehajúce aj ukončené inštancie procesov a detaily o nich. Zobrazenie diagramu procesu ukazuje počet a miesta tokenov v inštancii, ktoré reprezentujú momentálne vykonateľné úlohy v procese. Umožňuje zobraziť chyby, ktoré sa v inštancí procesu vyskytli a tie dokáže v niektorých prípadoch aj priamo opraviť.

Riešenie problémov v procesoch pomocou nástroja Cockpit by sa dalo nazvať „vizuálnym debugovaním“. Riešnie problémov prebieha približne nasledovne [1]:

[20] `https://docs.camunda.org/manual/7.14/reference/rest/decision-definition/post-evaluate/`

[21] `https://docs.camunda.org/manual/7.14/user-guide/dmn-engine/evaluate-decisions/`

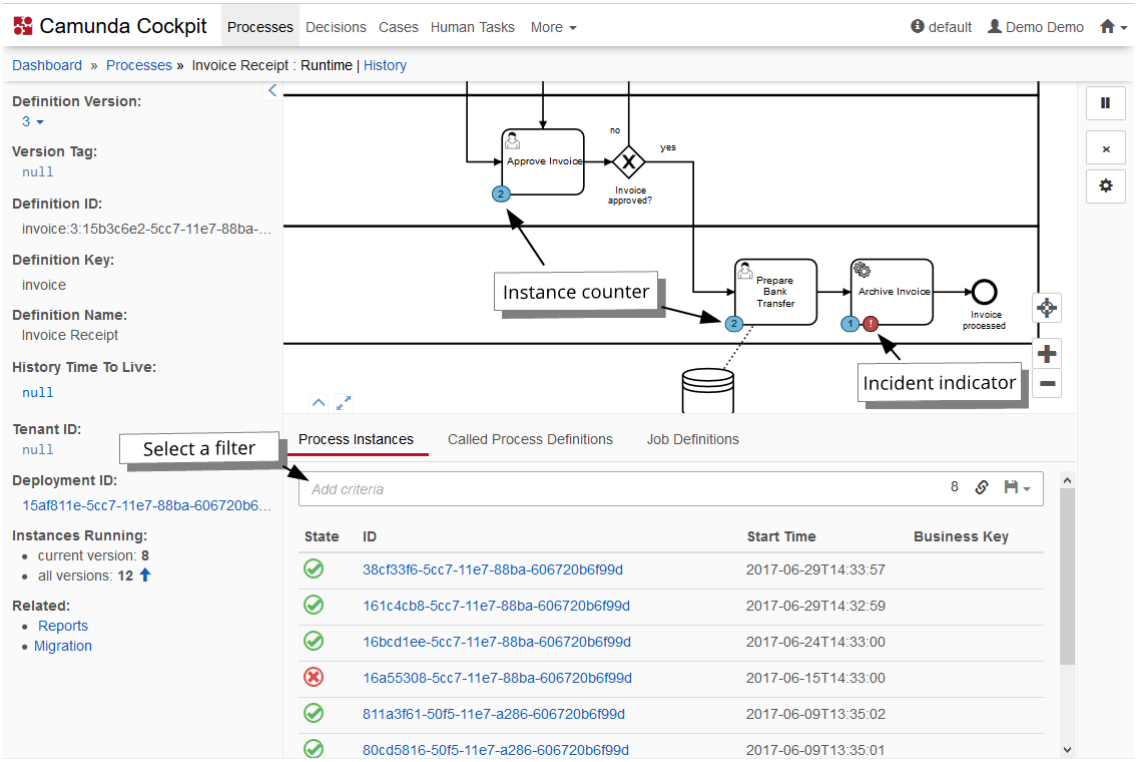
[22] Vyhľadávanie môže byť na základe ID hodnoty procesu, úlohy, prideleného vykonávateľa a iných kritérii (viď. dokumentácia `https://docs.camunda.org/manual/7.14/webapps/tasklist/filters/`)

Nájdenie problému je možné pomocou prehľadu bežiacich procesných inštancii. Inštancie, v ktorých bola zaznamenaná chyba (vyvolaná udalosť výnimky v procese, neodpovedajúca alebo chybne volaná vzdialená služba, chýbajúce dáta, výnimka v kóde Java triedy atď.) sú v prehľade zvýraznené.

Analýza problému prebieha pomocou detailného zobrazenia problémovej inštancie. Užívateľ vidí, aký konkrétny problém nastal a v ktorom bode procesu nastal. Vie si zobraziť podľa akých pravidiel sa rozhodovalo v tabuľkách DMN. V prípade, že problém nastal kvôli hodnote určitej premennej, vie si zobraziť ako sa hodnota danej premennej menila počas prechodu procesom a kedy nadobudla neočakávanú hodnotu.

Vyriešenie problému je možné priamo v aplikácii. Vďaka možnosti určitý proces pozastaviť, následne upraviť jeho model na novú verziu a existujúce inštancie na túto verziu migrovať, alebo možnosti priamej zmeny stavu inštancie vie byť riešenie problémov dynamické a časovo efektívne.

Treba spomenúť, že funkcionalita nástroja je v komunitnej verzii značne obmedzená. Zoznam funkcii dostupných v komunitnej verzii je uverejnený na stránkach spoločnosti Camunda[23] a detaily v oficiálne dokumentácii[24].



Obr. 2.10: Camunda Cockpit. Prebrané z [1]

[23] https://camunda.com/products/camunda-platform/cockpit/
[24] https://docs.camunda.org/manual/7.14/webapps/cockpit/bpmn/process-definition-view/

Admin

Webová aplikácia, ktorá spravuje užívateľov pomocou API abstrakcie *Identity Service*, ktorá je súčasťou procesného enginu. Spravuje oprávnenia užívateľov a skupín. V enterprise verzii je pomocou neho možná integrácia s existujúcim systémom na správu užívateľov (LDAP).

Optimize

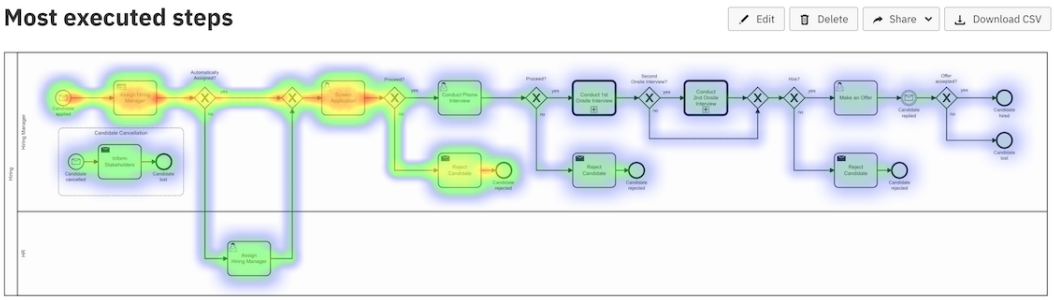
Nie je priamou súčasťou distribúcie BPM platformy. Pokiaľ užívateľ používa enterprise verziu distribúcie, dá sa nainštalovať a integrovať s procesným enginom cez REST API. Údaje v ktorých vyhľadáva si ukladá vo vlastnej NoSQL databáze Elasticsearch[25], určenej na spracovanie veľkého množstva dát.

Poskytuje štatistický prehľad nasadených procesov a umožňuje o nich vytvárať reporty. Na základe zbieraných štatistík ponúka vytvorenie upozornení pri prekročení určitých hodnôt (po prekročení daného počtu bežiacich procesov, preročenie počtu procesov čakajúcich v určitom stave dlhšie ako niekoľko hodín atď.). [2]

Umožňuje zobraziť heatmapu nasadeného procesu, ktorá ukazuje najčastejšie volené cesty procesom a najpoužívanejšie aktivity. Zároveň umožňuje zobraziť časovú heatmapu, ktorá zobrazuje časti procesu, ktorých uskutočňovanie trvá najdlhšie. Užívateľ si tak môže vytvoriť predstavu, ktoré časti procesu treba zrýchliť, alebo zmeniť.

Pre väčší prehľad v monitorovaných štatistikách je možné vytvoriť *Dashboard*, ktorý bude prispôsobený potrebám užívateľa a bude zobrazovať všetky dôležité údaje na jednom mieste.

O možnostiach nástroja a práci s ním sa dá informovať na stránkach spoločnosti[26].



Obr. 2.11: Camunda Optimize. Prebraté z [2]

[25] https://www.elastic.co/
[26] https://camunda.com/products/camunda-platform/optimize/

2.2.2 Bonita

Platforma spoločnosti Bonitasoft ponúka na vývoj a nasadenie procesných aplikácii vlastné integrované vývojové prostredie (IDE), umožňujúce počas vývoja komunikovať s ďalšími komponentami platfomy. Nástroj sa nazýva Bonita Studio. Vytvorená aplikácia sa môže následne nasadiť do behového prostredia samostatne ponúkaného spoločnosťou.

- **Bonita Studio** - Vývojové prostredie
  - Application Designer
  - Bonita UI Designer
  - Bonita Runtime v konfigurácii Tomcat server + H2 Databáza
- **Bonita Runtime (Bonita Server)** - Prostredie na beh aplikácii
  - Bonita Portal
  - Bonita Engine

Bonita Studio

Je prostredím na návrh a nasadenie procesnej aplikácie do testovacieho prostredia. Vďaka integrácii modulov zodpovedných za jednotlivé fázy vývoja a dizajnu procesnej aplikácie umožňuje rýchle vytvorenie funkčného prototypu aplikácie. Pri záujme vytvoriť a otestovať novú procesnú aplikáciu zahŕňa Bonita Studio všetko, čo je k tomu potrebné. Ako vidno na obrázku 2.12, štúdio obsahuje vstavané behové prostredie (Bonita Portal, procesný engine) a a dizajnér užívateľského rozhrania (Bonita UI Designer).

Application Designer

Základný vývojový komponent prostredia Bonita Studio. Je to integrovavné vývojové prostredie (IDE) uspôsobené na potreby vývoja procesnej aplikácie. Obsahuje *vývojové menu* a *editor procesných diagramov*.

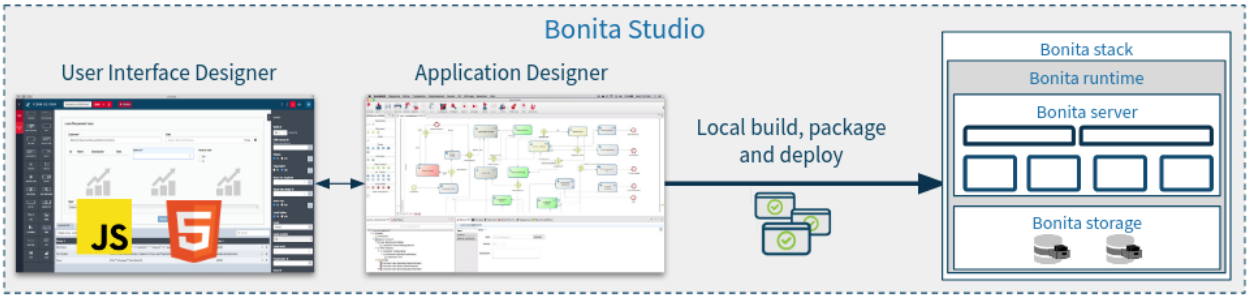
Vývojové menu pomáha jednoducho vytvárať objekty, ktoré na svoj chod procesná aplikácia vyžaduje, ako napríklad objekt s podnikovými dátami, súbor organizačnej štruktúry podniku, alebo rôzne connectory na exteré systémy a databázy. Pri spustení Bonita Štúdia sa spúšťa aj procesný engine. Do enginu je tak možné priamo počas vývoja tieto súbory nasadiť. Aplikačný dizajnér ich zaregistruje a umožní s nimi počas vývoja ďalej pracovať.

Editor diagramov umožňuje vytvárať diagrami v notácii BPMN 2.0 priamo vo vývojovom prostredí. Parametre diagramu, ako sú dáta a aktéri na jednotlivých úlohách sa dajú nastavovať na základe definovaných dátových a organizačných objektov. Z elementu úlohy užívatela sa dá priamo pristúpiť na dizajnér UI, ktorý umožňuje vytvárať podobu formuláru prideleného k danej úlohe.

Dizajnér užívateľského rozhrania (Bonita UI Designer)

Obsahuje viac než 100 vizuálnych prvkov (widgetov), ktoré dizajnér umožňuje štýlom „Drag and drop“ vkladať do formulárov a vytvárať tak responzívné formuláre k úlohám, ktoré sú vykonávané užívateľom. Dájú sa pomocou neho vytvárať pohľady, ktoré bude aplikácia

obsahovať a definovať navigácia medzi týmito pohľadmi. Vývojárovi frontendu procesnej aplikácie stačí poznať definície dát z objektu dátového modelu aplikácie a môže definovať celkový dizajn procesnej aplikácie rovno s napojením na dáta.



Obr. 2.12: Bonita Studio. Prevzaté z [10]

.

Bonita Portal

Bonita Portal predstavuje predvolené používateľské rozhranie k procesnej aplikácii a poskytuje prístup ku:

- Pohľadom užívateľa
- Pohľadom administrátora
- Pohľadom technického správcu

Pohľady užívateľa slúžia k vykonávaniu úloh a vytváraniu nových inštancii procesov. Pohľad administrátora umožňuje správu nasadených procesov, organizácie a reportov. Technický správca má ako jediný práva vypínať a zapínať aplikačné služby pri ich údržbe alebo aktualízácii [7].

Bonita Engine

Dodáva sa spolu s Bonita Portálom v behovej (runtime) konfigurácii na priame nasadenie do produkcie. Taktiež je súčasťou vývojového balíka Bonita Studio. Obsluhuje volania dát v procesoch a dátových objektoch, nasadzuje, spúšťa a vykonáva procesy. Obsahuje internú databázu na ukladanie údajov o procesoch a ich histórie spúšťania. Podnikové dáta ukladá do samostatnej databázy, ktorú je potreba pred nasadením do produkcie nakonfigurovať. Pre potreby testovania je dodávaná s databázou H2 na ukladanie dátových objektov, ktorá sa po reštarte enginu premazáva.

2.2.3 Netgrif

Platforma od spoločnosti Netgrif sa skladá z dvoch nástrojov:

- Netgrif Application Builder (NAB)
- Netgrif Application Engine (NAE)

Netgrif Application Builder

Je verejná webová aplikácia[27], kde si používatelia môžu rýchlo navrhnúť vlastné procesné modely (spolu s dátami, dátovými formulármi a rolami), ktoré môžu byť následne exportované vo formáte XML a vložené do aplikačného enginu. NAB obsahuje tieto nástroje:

- Editor vytvárania a úprav akcii
- Editor vytvárania a úprav dátových premenných
- Editor vytvárania a úprav formulárov pre jednotlivé úlohy
- Nástroj na vytváranie a úpravu procesov a ich metadát
- Nástroj na modelovanie procesných modelov vo formáte Petriho sietí
- Editor na vytváranie a úpravu rolí prislúchajúcich procesu alebo úlohám

Netgrif Application Engine

Slúži ako nástroj na nasadenie a vykonávanie daných podnikových procesov. Je navrhnutý ako Java aplikácia s trojvrstvovou architektúrou. Frontend v aplikácii pokrýva technológia Angular[28]. Backend aplikácie je postavený na technológii Spring Boot[29]. Tri databázy, tvoriace dátovú vrstvu sú MongoDB[30], Elasticsearch[31] a Redis[32]. Jednotlivé vrstvy medzi sebou komunikujú za pomoci služieb REST API. To je použité aj smerom z aplikácie.

NAE obsahuje responzívny webový portál. Webový portál ľahko generuje dynamické používateľské rozhranie pomocou formulárov na základe importovaných procesov z NAB. Skladá sa z viacerých častí. Základom je server procesného enginu, ktorý umožňuje:

- Nahrávanie, spustenie a odstraňovanie procesov
- Vytváranie a odstraňovanie inštancí procesov
- Priradenie, dokončenie a zrušenie úloh inštancii procesu
- Prácu s dátovými premennými
- Vykonávanie akcí a ich spúšťanie na základe udalostí

[27] Dostupná na odkaze `https://builder.netgrif.com/modeler`
[28] `https://angular.io/`
[29] `https://spring.io/projects/spring-boot`
[30] `https://www.mongodb.com/`
[31] `https://www.elastic.co/elasticsearch/`
[32] `https://redis.io/`

Kapitola 3

Návrh aplikácie

V tejto kapitole z procesného pohľadu opisujem aplikáciu, ktorá bude v jednotlivých nástrojoch vyvíjaná. Na začiatok uvádzam podnikový účel aplikácie, resp. akú prácu aplikácia reprezentuje a umožňuje vykonávať. Ďalej opisujem spôsob, akým táto práca bude prebiehať a bližšie špecifikujem procesy, z ktorých sa aplikácia skladá. Následne predstavím aktérov v aplikácii a ich náplň práce pomocou diagramu prípadov použitia (Use Case Diagram).

Opis demonštračnej aplikácie

Alikácia reprezentuje pracovný postup firmy, ktorá ponúka poistenie motorových vozidiel. Náplň práce spočíva vo vytváraní, schvaľovaní a ukladaní žiadostí o poistenie.

Obchodný zástupca poisťovacej firmy začína proces vytvorenia žiadosti na poistenie a vypĺňa potrebné údaje, ktoré mu zadá záujemca o poistenie telefonicky, alebo osobne.

Žiadosť následne prechádza procesom schvaľovania. Ten začína automatickým určením úrovňe rizika na základe zadaných údajov a kontroly, či už rovnaká žiadosť nebola v minulosti evidovaná. Pokiaľ je riziko vyhodnotené ako vysoké, žiadosť bude schvalovaná manuálne schvalovateľom a v prípade jeho nedostupnosti po definovanom časovom limite vedúcim firmy. V prípade nízkeho a stredného rizika sa sa žiadosť schváli automaticky.

Po schválení žiadosti sa automaticky vypočíta cena poistného ako násobok úrovne rizika a výkonu vozidla, ktorá bude oznámená prostredníctvom obchodného zástupcu zákazníkovi. Pokiaľ zákazník cenu akceptuje, obchodný zástupca to potvrdí, žiadosť sa uloží do databázy a zákazníkovy sa odošle mail s číslom zmluvy novovytvoreného poistnenia a jej cenou.

V prípade nájdenia duplicity, zamietnutia žiadosti schvalovateľom, alebo neakceptácie ceny zo strany zákazníka sa mail neodosiela a podĺa dôvodu sa nastaví stavová premenná príslušnej žiadosti pre prípad potreby spätného dohľadania dôvodu zamietnutia.

3.0.1 Požadovaná procesná funkcionalita

Tu opisujem procesy a ich jednotlivé aktivity, ktoré by mala vyvíjaná aplikácia implementovať.

Proces vytvárania žiadosti

Začiatok životného cyklu žiadosti o poistnenie je vždy rovnaký:

1. Vytvorenie novej inštancie procesu žiadosti obchodným zástupcom
2. Vyplnenie údajov o osobe a vozidle
3. Automatické vytvorenie schvaľovacieho procesu

V tomto bode sa výkon presúva do novej inštancie schvaľovacieho procesu. Po jeho ukončení sa podľa jeho priebehu určí nasledujúci postup v procese vytvárania žiadosti:

- Po schválení žiadosti:
    1. Prijatá udalosť schválenia žiadosti
    2. Oznámenie ceny zákazníkovy a zaznamenianie jeho rozhodnutia
    3. Odoslanie mailu s údajmi o zmluve [1]
    4. Ukončenie procesu
- Po zamietnutí žiadosti alebo nájdení duplicity:
    1. Prijatá udalosť zamietnutia žiadosti
    2. Oznámenie dôvodu zamietnutia zákazníkovya následné ukončenie procesu

Proces schvaľovania žiadosti

1. Prijatá udalosť vytvorenia procesu
2. Kontrola duplicity na základe evidenčného čísla vozidla
3. Výpočet rizika na základe poskytnutých údajov[2]

Pokiaľ sa našla duplicita, proces končí a je o tom informovaný volajúci proces.
Pri neduplicitnej žiadosti a vysokom riziku žiadosti pokračuje proces nasledovne [3]:

1. Manuálne schválenie
    - Schválená - proces pokračuje
    - Neschválená - proces sa ukončuje, žiadosť je zamietnutá

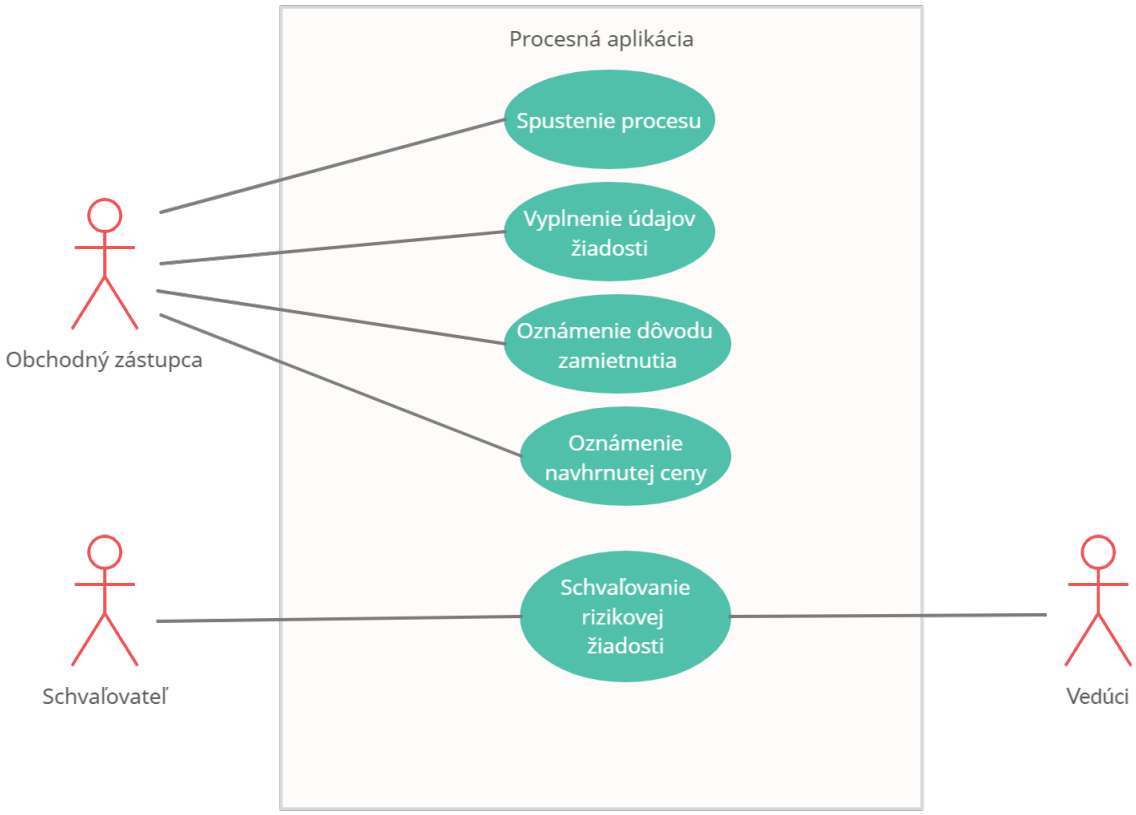
Po automatickom alebo manuálnom schválení nasleduje:

1. Automatický výpočet ceny

[1] len v prípade akceptácie ceny
[2] Výpočet bude prebiehať na základe veku vodiča a vozidla, výkonu vozidla a značky výrobcu
[3] Pri nižšom riziku sa táto fáza preskočí z dôvodu automatického schválenia.

2. Oznámenie ceny zákazníkovy
3. Akceptovania ceny alebo zamietnutie ceny [4]
4. Ukončenie procesu schvaľovania

3.0.2 Use case diagram aplikácie



Obr. 3.1: Use case diagram demonštračnej aplikácie

.

[4] Ak zákazník cenu neakceptuje, oba procesy sa v tom momente končia.

Vymedzenie porovnávania aplikácie

Navrhnutá aplikácia má slúžiť ako samostatný, interný systém pre novú fiktívnu poisťovaciu firmu "POISTISA". Nebudem preto porovnávať a testovať spôsoby integrácie existujúce systémy a dodatočné dodatočné spôsoby zabezpečenia prístupu k aplikácii.

Porovnávané je modelovanie procesov, spôsoby implementácie špecifikovanej procesnej funkcionality a práca s dátami. Spomenuté sú aj možnosti vytvorenia jednotlivých užívateľov (zamestnancov firmy) a nastavenia práv, ktoré špecifikujú, aké úlohy môže aktér vykonávať. Na záver porovnám spôsob nasadenia vytvorenej aplikácie do prevádzky. Zhrnutie silných a slabých stránok na základe nadobudnutých skúseností z vývoja sa nachádza v ďalšej kapitole.

Kapitola 4

Porovnanie vývoja aplikácie

V tejto kapitole je postupne v troch nástrojoch porovnaný vývoj aplikácie z pohľadu modelovania, implementácie funkcionality a nasadenia procesov. Každá časť je opísaná pre jednotlivé nástroje samostatne. Pri vývoji som sa v každej oblasti zameriaval na opis natívne podporovaných spôsobov riešenia problematiky danej oblasti. V ďalšej kapitole zhrniem moje skúsenosti s vývojom v jednotlivých nástrojoch a vzhľadom k jednotlivým častiam vývoja opisovaných v tejto kapitole spomeniem ich silné a slabé stránky.

4.1 Verzie nástrojou použité na vývoj aplikácie

Camunda

- Camunda Modeler verzie 4.6.0
- Komunitná edícia platformy vo verzii 7.15.0
- Aplikačný framework Spring Boot verzie 3.4.3

Bonita

- Komunitná edícia vývojového prostredia Bonita Studio verzie 2021.1

Netgrif

- Netgrif Application Engine verzie 5.1.2

4.2 Vytváranie diagramu

V úvodnej sekcii porovnania vývoja opisujem vytváranie diagramu procesu v modelovacích nástrojoch ponúkaných platformami. Zameriavam sa na vizuálne spracovanie, intuitívnosť, prehľadnosť diagramu a jednoduchosť pri úprave a nastavení parametrov vytváraného modelu. Vytvorené diagramy sú pre lepšiu čitateľnosť zobrazené samostatne v prílohe.

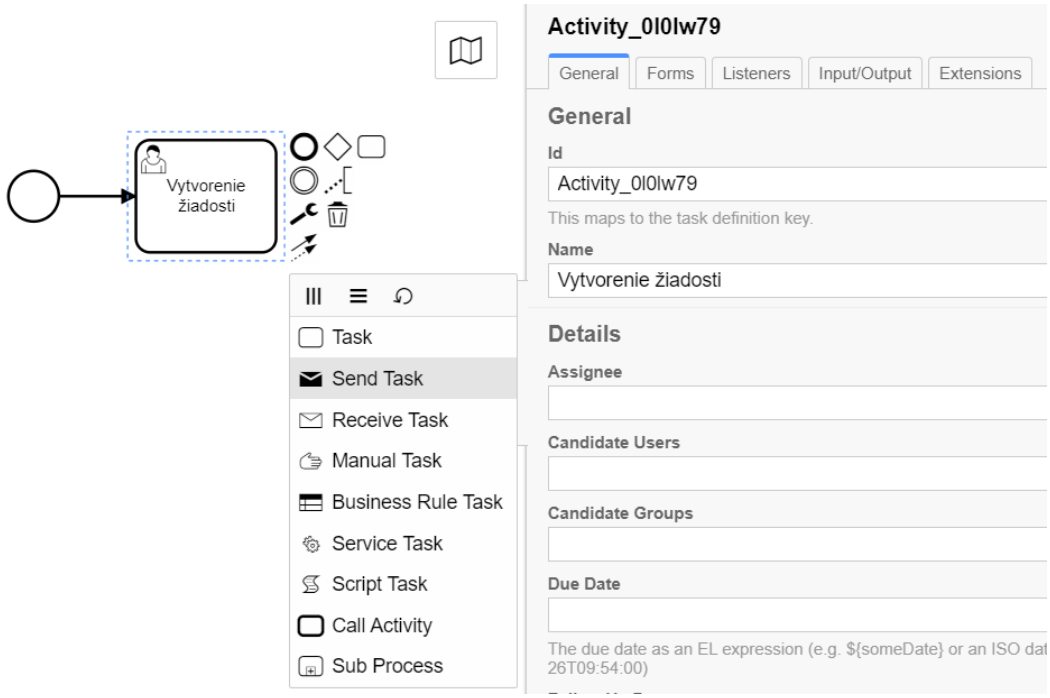
Camunda

V desktopovej aplikácii Modeler je na ľavej strane výber z kategórii symbolov a režimy ovládania, na pravej strane panel na úpravu parametrov práve vybraného prvku diagramu. Je možné mať otvorených viacero diagramov naraz a prepínať medzi nimi. Naspodu sa dá prepínať medzi diagramovým zobrazením procesu a jeho popisom vo formáte XML, ktorý sa dá taktiež priamo upravovať.

Diagram BPMN

Pri elemente, ktorý je práve vybraný, sú zobrazené ďalšie elementy, ktoré je možné vytvoriť a na vybraný element napojiť jedným kliknutím. Ďalej je pri ňom zobrazená ikona francúzskeho kĺúča, ktorá otvára menu s možnosťou zmeniť jeho typ, čo určuje jeho procesnú funkcionalitu (obr. 4.1). Podľa typu sa mení panel s parametrami relevantnými k danému typu. Okrem toho je možnosť aktívny prvok anotovať vizuálnym komentárom alebo zmazať. Tým, že sa tieto možnosti zobrazujú hneď pri vybranom prvku, je modelovanie veľmi intuitívne a užívateľ tak nestráca čas a prehľad v diagrame hľadaním ďalšieho symbolu v zozname na kraji obrazovky. Modeler neodlišuje prvky farebne a všetky majú uniformnú veľkosť, čo mu z môjho pohľadu dodáva čistý, nerušivý vzhľad. Aplikácia je responzívna celkový dojem z modelovania procesu bol veľmi dobrý.

Jediné čo by som uvítal, by boli nápovedy k jednotlivým parametrom. Novým užívateľom by to mohlo pomôcť pochopiť, ktoré parametre použiť a v akom prípade ich je vhodné použiť. Možností kedy to bude fnugovať je často viac, ale nie všetky sú na určité situácie ideálne.



Obr. 4.1: Možnosti úprav aktívneho prvku v modeleri a panel s parametrami

Tabuľky DMN

Rozhodovacie tabuľky sú súčasťou diagramu BPMN ako element typu *Bussiness rule task*. Tomu sa v Modeleri vyplní parameter `Decision Ref`, ktorý odkazuje na jednu z tabuliek DMN, ktorá bude nasadená v procesnej aplikácii a do parametru `Result Variable` sa uvedie názov premennej, ktorá uchová výsledok rozhodovania. Vytváranie tabuliek a prácu s nimi opisujem bližšie v sekcii 4.8

Bonita

Diagram BPMN

Vytváranie diagramu prebieha v integrovanom vývojom prostredí Bonita Studio. Tak isto ako v prípade predchádzajúceho nástroja, aj tu je panel s elementami vľavo a po kliknutí na element v diagrame sa pri ňom zobrazuje menu na úpravu typu elementu, alebo jeho napojenie na jeden z ponúkaných elementov. Ikonky sú však veľmi malé a chvíľu trvá, kým sa nový užívateľ zorientuje, aké elementy sú ikonkami zobrazené.

Počas práce v tomto nástroji je zapnutý pocesný engine, ktorý neustále komunikuje s vytváraným diagramom. Oproti Modeleru od spoločnosti Camunda je tak vytváranie diagramu oveľa pomalšie. Jednak je to slabou responzivitou pri presúvaní a napájaní jednotlivých elementov. Problém je ale aj so samotným dizajnom elementov. Sekvenčné toky spájajúce elementy sú veľmi tenké, je ťažké sa na ne trafiť. Zarovnávanie elementov tiež nie je ideálne a nástroj sa veľmi nesnaží užívateľovi v tomto smere pomôcť. Elementy nie sú v základnej veľkosti dostatočne veľké aby obsiahli dlhší text ako 2 slová. Pokiaľ má byť text vidno, treba ich zväčšiť, pričom nie je jednoduché trafiť uniformnú veľkosť s ostatými elementami aktivít.

Ako negatívum hodnotím zobrazovanie názvu pri elementoch rozdeľujúcich pracovný tok, ako sú brány, bez možnosti ho skryť. Často názov zakrýva tok alebo iný element a diagram sa tak stáva neprehľadnejším. Importoval som však procesy vytvorené v nástroji Camunda Modeler a vtedy názvy brán nebolo vidno, aj ked sa názvy importovali tiež. Diagram je vďaka tomuto trochu prehľadnejší. Pri modelovacom nástroji od spoločnosti Camunda som spomenul ako pozitívum, že elementy nie sú farebne odlíšené. V tomto nástroji sú, ale neberiem to vyslovene ako negatívum, pretože by bol diagram bez farebného odlíšenia ešte neprehľadnejší.

Netgrif

Petrifow model

Na vytvorenie modelu procesu v jazyku Petriflow sa používa verejne dostupná aplikácia Netgrif Builder[1]. Tá obsahuje vrchnú lištu s výberom elementov siete, režimov modelu a všetky možnosti úprav modelu. Pre nového užívateľa môže byť lišta neprehľadná a bolo by vhodné mať jednotlivé skupiny podľa funkčnosti viac oddelené. Čo mi chýbalo, bola možnosť mať viac modelov otvorených naraz v jednom okne prehliadača a možnosť vrátenia posledne vykonanej zmeny. Samotné vytváranie siete prebieha vybraním uzlu, ktorý chce užívateľ do siete vložiť (prvok miesta a prechodu) a ich „naklikaním“ do pracovnej plochy. Tieto prvky sa následne spájajú pomocou rôznych typov hrán. Taktiež je možné miestam priradiť určitý počet tokenov a vytvoriť tak počiatočnú konfiguráciu siete. Vytvorenú sieť je možné exportovať ako súbor vo formáte XML.

Ako možno vidieť v prílohe práce, z vizuálneho hľadiska nie je vytvorený diagram taký atraktívny ako v prípadoch diagramov v notácii BPMN, ktoré sú na modelovanie podnikových procesov uspôsobené množstvom typov symbolov, znázorňujúcich ich charakter práce. Počet elementov v diagrame môže byť až podstate dvojnásobný, keďže okrem úloh sa musia definovať aj stavy procesu v podobe miest. Takisto nie je vidno, aký aktér bude jednotlivé úlohy vykonávať, na čo v zápise BPMN slúžia dráhy.

Dá sa ale na tento spôsob procesného modelovania zvyknúť a výsledok môže byť pri vhodne navhrnutej sieti pochopiteľný aj pre laika. Zároveň je presne vidno, akým spôsobom sa môže pracovný tok v procese pohybovať a možnosti paralelizácie úloh resp. deľby práce na podúlohy sú v podstate neobmedzené a efektívnejšie ako pri použití brán v notácii BPMN.

Režim úpravy dát a formulárov

Vďaka režimu úpravy dát je možné nadefinovať dáta procesu, ich dátové typy, identifikáciu a validáciu. Pokiaľ je dát väčšie množstvo a presahujú výšku obrazovky tak občas nastane problém so zobrazením a posledne vytvorené dáta nie je vidno. Nástroj sa stále vyvíja a vo viacerých prípadoch je to vidno.

Formuláre je možné vytvárať na jednotlivých prechodoch (úlohách), pričom je možné do formuláru vložiť už definované dáta, alebo vytvoriť nové. Pri vytváraní modelu je tak možné definovať tento vizuálny aspekt procesnej aplikácie. Vytváranie a prácu s formulármi bližšie opisujem v sekcii 4.4.

[1] `https://builder.netgrif.com/modeler`

Úprava rolí v procese

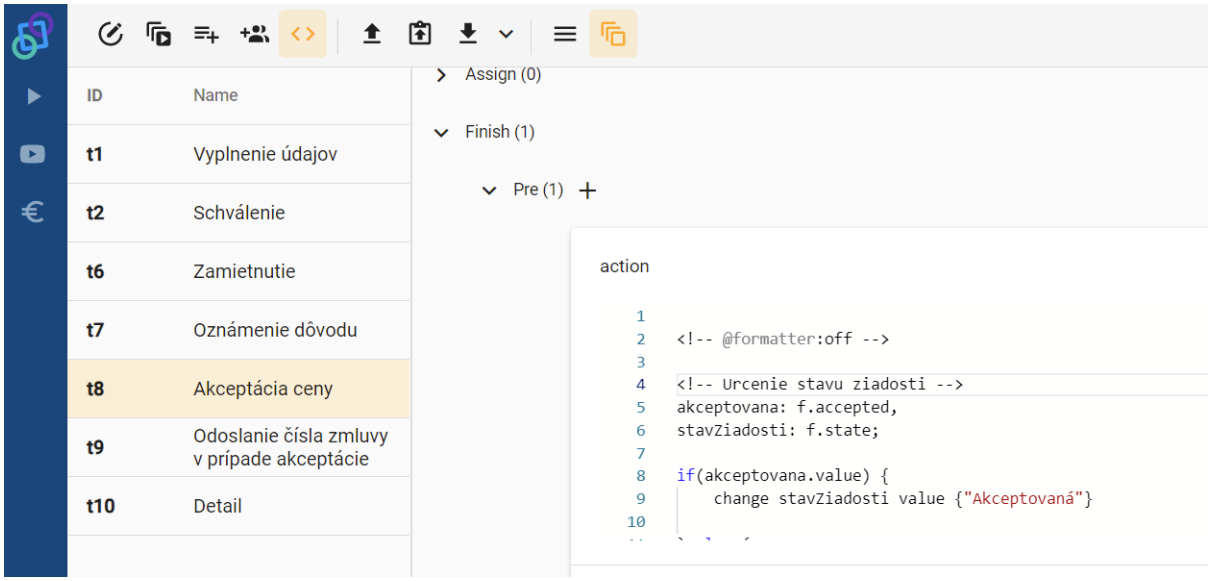
Režim rolí umožňuje definovať jednotlivé role, ktorým sa môžu definovať práva na celý proces (vytváranie, zobrazenie a rušenie inštancií), alebo práva vykonávať prechody jednotlivých úloh[2]

Vytváranie akcií

Na jednotlivé fázy prechodov ale aj pri zmene hodnoty určitej premennej sa dajú priamo v nástroji Builder definovať akcie, ktoré pridávajú do procesu dynamiku a automatizáciu. Akcie sa píšu v jazyku založenom na programovacom jazyku Groovy. Ich štruktúru a možnosti bližšie popisujem v sekcii automatizácie 4.7.

Režim simulácie siete

Na rozdieľ od zvyšných dvoch opisovaných modelovacích nástrojov, Netgrif Builder natívne podporuje simuláciu prechodu procesom, čím si môže užívateľ overiť správne vykonávanie procesu. Akcie resp. automatizácia sa samozrejme simulovať nedá, čiže to nemusí vždy zodpovedať reálnemu prechodu procesom, ale na overenie dostupnosti v sieti je to užitočný režim, ktorý je použiteľný aj na rýchle pochopenie fungovania Petriho sietí.



Obr. 4.2: Režim vytvárania akcii v nástroji Netgrif Applicaton Builer

[2]Priradenie, delegácia, zrušenie, dokončenie. Viď. reprezentáciu úlohy v jazyku petriflow v sekcii 2.1.3

4.3 Dáta v procese

V tejto časti je porovnaná práca s procesnými dátami. Zameriavam sa na vytváranie dát, aké typy dát sú natívne podporované a ako sa k nim v procese pristupuje.

Camunda

Camunda definuje dva rozsahy premenných:

- Procesný rozsah
- Lokálny rozsah

K premenným z procesného rozsahu sa dá pristupovať počas priebehu procesnej inštancie. Po skončení procesu sa ukladajú do databázovej schémy s historickými dátami a teda sú perzistentné. Premenné z lokálneho rozsahu sú viditeľné len pre danú úlohu, alebo prechod. Neskôr zanikajú a nedá sa k nim už pristúpiť. Vďaka rozslišovaniu rozsahov sa dá šetriť miestom v databáze a zefektívniť chod aplikácie, ale na druhej strane to vyžaduje správne rozlišovanie rozsahov a dôslednejšiu implementáciu.

Vstupné a výstupné parametre

Úlohe v procese sa v Modeleri v záložke *Input/Output* v paneli s parametrami dajú namapovať vstupné a výstupné dáta. Nimi sa pomocou skriptu, alebo použitím EL dá napríklad určiť hodnota na základe premennej z aktuálneho dátového rozsahu, ktorá sa uloží do premennej lokálneho rozsahu v danej úlohe, s ktorou bude v úlohe prebiehať výpočet. Alebo sa môže jednoducho určiť parameter pre potreby automatizácie, napríklad meno správy, ktorá sa v úlohe odosiela a na inom mieste podľa daného mena zachytáva.

Výstupné parametre sa môžu použiť na priradenie hodnoty z lokálnych premenných do procesných premenných. Celkový prínos môže byť v znížení pamäťovej náročnosti aplikácie bez náročnejšej implementácie navyše.

Dátové typy

Natívne podporované dátové typy sú:

- string
- boolean
- integer
- long
- double
- date (vo formáte "yyyy-MM-dd'T'HH:mm:ss")

Všetky hodnoty sa dajú natívne transformovať z typu `java.lang.String`. Tieto typy vo väčšine prípadov postačujú. Pokiaľ ale chce užívateľ nahrať súbor, alebo len dátum bez času (natívne podporovaný je len vyššie spomenutý formát), je potreba to ošetriť v kóde aplikácie [3], čo je v porovnaní so zvyšnými dvoma nástrojmi práca navyše.

[3] V prípade súboru je potrebné v aplikácii vytvoriť súbor vo formáte HTML, ktorý bude napojený pomocou názvu súboru v modeleri

Frekvencia komunikácie s databázov

Procesný engine od spoločnosti Camunda je pasívny kus kódu v jazyku Java, ktorý pokiaľ nevykonáva prácu, čaká v stave *Wait*. V tomto stave je potrebné čakať na externý podnet, ktorý oznamuje, že sa procesná inštancia môže ďalej vykonávať, až pokiaĺ sa nenarazí na ďalšiu úlohu vyžadujúcu externý podnet (úloha užívateľa, zachytávajúce udalosti).

Jeden celok práce enginu medzi dvoma miestami vyžadujúcimi externý podnet sa nazýva transakcia. Transakcia ukladá zmeny do databázy až po jej úspešnom vykonaní. Pokiaľ teda niekedy počas vykonávania transakcie nastane chyba a engine v práci nemôže pokračovať, vykonaná práca do daného momentu od začiatku transakcie sa zahodí a proces zostáva na mieste posledného čakcieho stavu.

Transakcie prebiehajú synchónne v jednom vlákne, ale pokiaľ je treba, môžu sa pred a po prejdení pracovného toku elementom vykonať asynchrónne. Tak je možné definovať, o koľko sa vykonávanie procesu presunie dozadu pokiaľ dojde k chybe. Asynchrónne spracovanie je možné definovať v Modeleri v paneli parametrov prvku diagramu (zvolenie parametrov *asyncBefore* a *asyncAfter*). Je to vhodné najmä v prípade viacerých automatizovaných úlohách za sebou, ktoré predvolene neobsahujú hraničné miesto transakcie.

Bonita

V procesných aplikáciach sú dáta definované dvoma spôsobmi.

- Model podnikových dát
- Procesné premenné

Model podnikových dát

Dáta, ktoré sa budú ukladať do databázy sa musia definovať pomocou *podnikového modelu dát (Buiness Data Model)* vo vývojovom prostredí Bonita Studio. Procesná aplikácia obsahuje len jeden takýto model, ktorý je v aplikácii prítomný ako súbor vo formáte XML a predstavuje hierarchiu databázy podnikových dát. Ten je tvorený hierarchickou štruktúrou balíkov obsahujúcich dátové objekty, ktoré reprezentujú tabuľky databázy. Dátové objekty obsahujú typicky všetky dôležité atribúty určitej podnikovej položky a slúžia ako predlohy na vytváranie dát danej položky v procesnej inštancí. Jednotlivým modelom procesov sa tieto predlohy môžu pridať a počas procesu ich treba inicializovať, čím sa vytvoria dáta konkrétnej položky. Dátové objekty môžu obsahovať referencie na iné objekty z dátového modelu a vedia sa tak vytvárať kompozičné a alebo agregačné závislosti medzi dátovými objektami resp. tabuľkami dátovej databázy.

Výhodou tohto prístupu je, že sa konkrétny dátový objekt, ktorý je uložený v databáze podnikových dát môže jednoducho vyhľadať a použiť vo viacerých procesoch. Môže byť tak zdielaný viacerými aplikáciami nasadenými v jednom behovom prostredí. Používanie dátových objektov umožňuje automatizovať inicializáciu dát procesnej inštancie.

Nevýhody tohto prístupu sa ukázali pri potrebe zmeniť, alebo pridať čo i len jednu premennú do určitého dátového objektu, keďže to znamená zmenu databázovej štruktúry. Vtedy sa musí celý systém pozastaviť, dátový model znova nasadiť a aplikácia opäť spustiť.

V mojom prípade som vytvoril jeden dátovy objekt obsahujúci dáta ohľadom žiadosti o poistenie. Tieto objekty sa vytvárajú pomocou editoru podnikových dátových modelov.

Po definícii stačilo dátový model nasadiť do enginu procesnej aplikácie jedným kliknutím a následne som mohol dátový objekt žiadosti priradiť k procesu a mať tak preddefinované základné dáta aplikácie, ktoré som mohol dynamicky priradiť úlohám.

Procesné premenné

Tak isto ako v nástroji od spoločnosti Camunda, môžu byť *lokálneho* rozsahu, kedy sú definované a dostupné len v jednej úlohé v procese, alebo *procesného* rozsahu, kedy sú definované v bazéne procesu 2.1.1. Môže sa s nimi manipulovať v celom procese a po ukončení inštancie sa ukladajú do databázy procesného enginu spolu s informáciami o danej inštancii.

Všeobecne sú skôr vhodné na ukladanie informácii, ktoré nebude po ukončení inštancie procesu potreba dohľadávať. Ukladajú sa síce do databázy procesného enginu v rámci archivovanýc inštancií, no prístup k dátovým objektom je priamočiarejší.

Procesné premennné som využil na ukladanie výsledkov pri určovaní vekových kategórii vodiča a vozidla, ktoré boli neskôr použité na rozhodovanie o rizikovosti žiadosti.

Dátové typy

Dátové objekty môžu obsahovať všetky primitívne dátové typy jazyku Java. Navyše je podporovaných aj viacero formátov dátového typu *LocalDate* (DATETIME-TIMEZONE, DATETIME-NO TIMEZONE a DATE ONLY). Negatívom je, že súbory nie sú podporované ako atribúty dátových objektov a nie je ich tak možné ukladať do dátovej databázy ako súčasť jedného objektu podnikovej položky. Súbor je však možné nahrať, len bude uložený v oddelenej databáze procesného enginu, kde sa ukladajú jednotlivé inštancie procesov spolu s ich procesnými premennými. Ak sa chce užívateľ dostať k súboru patriacemu k určitej podnikovej položke, tak musí poznať Id konkrétnej inštancie procesu, ktorá podnikovú položku inicializovala.
Premenné procesu majú širšie možnosti, čo sa týka ich typov dát. Môžu predstavovať hocijaký typ objektu podporovaného jazykom Java a tak sa pomocou nich dajú spracovávať aj špeciálne typy dát.

Prístup k dátovým objektom

Pristupovať k dátovým objektom sa dá pomocou API aplikačného enginu cez prístupového objektové rozhranie *DAO (Data access object)* [4], pomocou preddefinovaných queries na základe parametrov dátového objektu, čo urýchľuje ich filtráciu. Taktiež sa dá pristupovať pomocou REST API [9]. Aktualizovať hodnoty objektu je možné len v bežiacom procese pomocou *operácii*.

Operácie

Operácie slúžia na zmenu hodnôt dát alebo ich inicializáciu v procese. Vytvárajú sa v editore operácii v nástroji Bonita Studio. Nastavujú sa jednotlivým úlohám v diagrame a po skončení danej úlohy sa operácie vykonajú. Operácia má dva parametre: premennú, ktorá sa má aktualizovať a novú hodnotu.

Nová hodnota môže byť výsledok skriptu a môže tak predstavovať aj prácu automatizovanej úlohy, alebo len konkrétna premenná. Operácie sa v prípade úloh užívateľa vytvoria

[4] `https://en.wikipedia.org/wiki/Data_access_object`

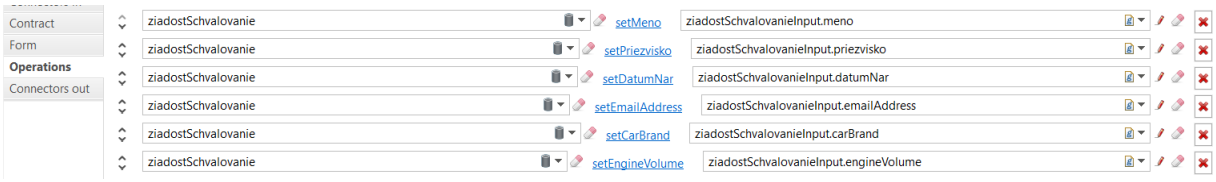
automaticky na základe kontraktu, pričom nové hodnoty určuje vstup od užívateľa zadaný vo formulári.

Kontrakty

Kontrakty predstavujú dáta, ktoré musia byť v procese dostupné po ukončení úlohy, toto je kontrakt úlohy. Na ich základe sa automaticky vytvoria operácie, ktoré mapujú vstup od užívateľa z formulára na premenné dátového objektu, alebo procesné premenné. Keď je na úlohe užívateľa definovaný kontrakt, formulár vytváraný v nástroji *UI designer* bude mať automaticky vytvorené polia na vstup na základe daného kontraktu, čím sa vytváranie formulárov uľahčuje.

Existujú aj procesné kontrakty, slúžia na definíciu a inicializáciu dát, ktoré majú prítomné v procese počas jeho životného cyklu. Týmto spôsobom procesný engine vie, aké dáta budú v úlohách procesu dostupné a môže tak počas vytvárania diagramu dynamicky kontrolovať prácu s dátami v procese ponúkaním relevantných dát úlohám.

Kontrakty je vhodné používať inicializáciu a zmenu hodnôt dátových objektov. Jednotlivé premenné dátového objektu, ktorý je v procese definovaný ako podniková premenná, sa dajú automaticky pridať do kontraktu, pričom je možné určiť, či sa po vykonaní úlohy resp. vytvorení procesu majú inicializovať, alebo aktualizovať. V oboch prípadoch sa automaticky vytvorí skript, ktorý sa po ukončení úlohy spustí a inicializuje dátový objekt a jeho položky, alebo ich len aktualizuje podľa vstupu z formuláru.



Obr. 4.3: Operáci vytvorené na základe kontraktu

Frekvencia komunikácie s databázov

Ukladanie stavu procesu prebieha asynchrónne a čo najčastejšie. Pokiaľ je na úlohe použitý konektor pristupujúci ku vzialenému zdroju a odosielajúci do, resp. z úlohy dáta, vytvárajú sa ďalšie transakčné úlohy. Konektory preto vedia byť signifikantnou záťažou pri vykonávaní procesu. Transakčné hranice nie je možné upravovať.

Netgrif

Ako hlavná databáza je použitá MongoDB. Radí sa medzi NoSQL databázy a je dôležitým stavebným blokom pre „real-time“ webové aplikácie ako je NAE (Netgrif Application Engine). Do tejto databázy sa ukladajú všetky dáta z Petriflow procesov, ale aj informácie o používateľoch, skupinách, roliach a iné. Dáta sú definované na začiatku súboru procesného modelu a v úlohách, v ktorých majú byť viditeľné a upraviteľné ako súčasť formuláru, sú odkazované pomocou ich identifikačnej hodnoty[5].

Pokiaľ prirovnáme spôsob ukladania dát v tomto nástroji s relačnými databázami, procesy v jazyku Petriflow korešpondujú s tabuľkami, zatiaľ čo inštancie procesov predstavujú jednotlivé riadky (záznamy) v tabuľke. Podobne ako v prípade cudzieho kľúča v tabuľke

[5]Tá sa líši od názvu premennej, ktorá je vo formulári zobrazovaná

relačnej databázy, procesné dátové premenné môžu ukladať referencie na inštancie procesov alebo na úlohy konkrétnej inštancie [18].

Všetky dáta, ktoré sú v modeli procesu definované, sú tak viditeľné počas celého jeho životného cyklu a všetky úlohy k nim môžu pristupovať, čiže neexistujú rozsahy premenných v rámci procesu. Je to odlišný prístup od zvyšných porovnávaných nástrojov. Na jednej strane to uľahčuje vývoj, ale pridáva to záťaž na databázu.

Dáta taktiež netreba manuálne inicializovať ako v ostatných dvoch nástrojoch, čo znova uľahčuje vývoj. Je však treba zvážiť, aké dáta sú v procese skutočne potrebné kedže k nim musí byť stále možný prístup.

Typy dát

Aplikačný engine definuje množstvo typov objektov, ktoré slúžia na uchovávanie rôznych typov procesných dát. Dátová premenná resp. dátový objekt je primárne definovaný svojim typom, ktorý určuje aké hodnoty môže obsahovať, unikátnym ID, pomocou ktorého je v procese dátový objekt odkazovaný a názvom, ktorý je užívateľovi vo formulári zobrazený. [17]

Tieto objekty tak môžu podľa svojho typu predstavovať atomické dáta v procese vrátane dátumu, alebo zoznamy s textovými hodnotami (*Multichoice, Enumeration*) a iné, špeciálne typy, akými sú napríklad súbor alebo referenčný typ odkazujúci na dáta inej úlohy.

Prístup k dátam

Ako som spomenul, dáta ktoré majú byť súčasťou formuláru úlohy, musia byť v jej XML štruktúre odkazované pomocou ich identifikačnej hodnoty. To je možné docieliť už v nástroji Builder pri vytváraní formuláru danej úlohy. Pokiaľ majú byť dáta procesu použité v akcii, musia sa na jej začiatku deklarovať zápisom, ktorý bližšie popisujem v sekcii 4.7.

Referenčné premenné

Je to špeciálny typ typ premennej, ktorá predstavuje všetky dáta konkrétnej úlohy. Pokiaľ je v procese viac úloh s rovnakými formulármi, stačí, aby úloha obsahovala referenciu na úlohu s rovnakými dátami a tie sa po inicializácii referenčnej premennej zobrazia tak, ako v odkazovanej úlohe. Pomáha to pri znovupožiteľnosti dát a urýchľuje to vývoj. Zároveň to môže znižovať záťaž na databázu, pokiaľ je potreba zobraziť alebo upraviť dáta z iného procesu a netreba dáta v danom procese používať neskôr[6].

Prepočet stavu procesu

Pokiaľ sa v sieti udeje zmena v podobe vykonania prechodu resp. niektorej z jeho prechodových fáz[7], prepočítava sa nová konfigurácia (stav) siete a ten sa ukladá do databázy. Tak sa určí, ktoré prechody sú po prepočte zmien spustiteľné. Zmeny hodnôt dát sa ukladajú priebežne tiež, čo predstavuje možnost mať rozpracovaný formulár a hocikedy sa k nemu vrátiť. Dátové zmeny ale prepočet konfigurácie siete neaktivujú, čo trochu znižuje dynamiku automatizácie. Ako príklad uvediem situáciu, kedy proces vytvárania žiadosti čaká na schválenie žiadosti vo vzdialenom procese. Z miesta v ktorom čaká sú dva prechody spustiteľné potom, ako splnia váhové podmienky hrán určených procesnými premennými. Tieto

[6] V aplikácií som referenčnú premennú použil na zobrazenie dát pri manuálnom schvaľovaní žiadosti

[7] viď. princíp úloh v jazyku Petriflow 2.1.3

premenné som vzdialene po rozhodnutí o schválení nastavil, no prechody sa neaktivovali, keďže k prepočtu aktívnych prechodov nedošlo. Aby bola konfigurácia siete prepočítaná, musel som ešte následne vzdialene aktivovať prechodovú fázu úlohy zobrazovania detailov v procese vytvárania žiadosti, čo nie je ideálne riešenie. Dobré by bolo mať možnosť nastaviť prepočet po dokončení vzdialene vytváraného procesu, čo by tak fungovalo podobne ako element *volania aktivity* v notácii BPMN 2.0.

4.4 Formuláre

Po namodelovaní procesu a definícii dát, ktoré bude obsahovať, je potreba navrhnúť formuláre, ktoré budú priradené jednotlivým užívateľským úlohám a budú poskytovať potrebný vstup, ktorý vypĺňa užívateľ. V tejto časti opíšem natívne podporované funkcie na vytváranie formulárov, ich prepojenie s aplikačnými dátami a možnosti ich vizuálnych úprav.

Camunda

Treba na začiatok poznamenať, že riešenie od Camundy v sebe nezahŕňa veľa možností na vizuálnu úpravu procesnej aplikácie a skôr sa počíta s napojením aplikácie na samostatne vyvíjaný frontend. Poskytuje však predvolené užívateľské prostredie *Tasklist* na vykonávanie užívateľských úloh, ktoré na demonštračné účely postačuje a ktoré v aplikácii používam. Tu uvádzam jeho možnosti zobrazenia formulárov.

Vytváranie formulárov

- Generické formuláre
- Generované formuláre
- Embedované formuláre
- Externé formuláre

Generické formuláre

Sú vhodné pri testovaní aplikácie. Sú predvolené, pokiaľ nie sú v Modeleri definované žiadne dáta fomuláru úlohy. Užívateľ pridáva manuálne nové premenné, ktorým nastaví hodnotu. Po dokončení úlohy sa dané premenné uložia do inštancie procesu.

Generované formuláre

Vytvoria sa automaticky z definície procesu, ktorá je vo formáte XML nasadená v aplikácii, pokiaľ má úloha definované dáta formuláru. Tieto dáta sa dajú nastaviť v paneli parametrov pre užívateľskú úlohu.

Uvítal by som, kebyže sa dáta použité vo formulári automaticky incializujú. Pokiaľ sa tak totiž pred úlohou, ktorá obsahuje formulár s dátami obsahuje neurobí, vykonávanie skončí chybou.

Embedované formuláre

Modeler umožňuje napojiť v paneli parametrov cez parameter `Form Key` formulár, ktorého súbor vo formáte HTML je prítomný v aplikácii. Tak poskytujú aspoň určitú formu personalizácie. Pokiaľ by chcel užívateľ napríklad nahrať do Tasklistu súbor, musí tento súbor vo formáte HTML vytvoriť a vložiť do aplikácie, kedže nahrávanie súborov nie je natívne podporované.

Externé formuláre

Poskytujú možnosť otvoriť na novej stránke v prehliadači formulár, ktorý obsahuje iná aplikácia.

Parametre dát vo formulároch

V modeleri je možné jednotlivým dátam nastaviť základnú validáciu, pomocou klúčových slov. Napríklad *upraviteľnosť, viditeľnosť, povinné pole*. Stretol som sa ale s chybou, kedy som chcel zobraziť Boolean premennú, ktorá mala byť len viditeľná a nemalo byť možné ju vo formulári zmeniť. Napriek tomu to bolo možné a kvôli neznámej chybe som úlohu nevedel dokončiť. Pokročilejšie validácie je možné napojeniť na validátor, ktorý môže byť vo formáte JavaScript súboru prítomný v aplikácii.

Bonita

Riešenie od spoločnosti Bonita ponúka bohaté možnosti úprav responzívneho užívateľského rozhrania vďaka nástroju UI Designer, ktorý som v aplikácii používal.

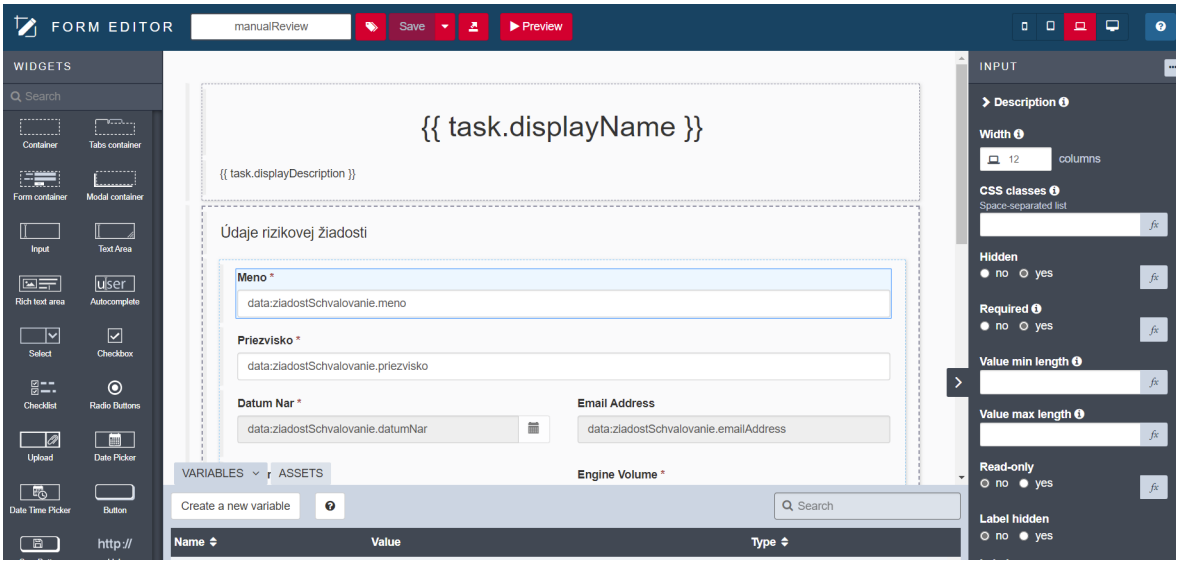
Vytváranie formulárov

V aplikácii sú implicitne vytvorené aj generické formuláre na účely testovania, ale samotné vytváranie formulárov je jednoduché a generické formuláre som nevyužil. UI Designer umožňuje spôsobom „drag and drop“ definovať, aké widgety budú použité na vstup od užívateľa pre jednotlivé premenné v procese a akým spôsobom budú widgety rozložené vo formulári. Ako príklad môžem uviesť widget výberu dátumu, selekcie z ponúkaných hodnôt, vloženia súboru, obrázku, grafu, tabuľky alebo odkazu URL. Výber je naozaj bohatý a je výhodné tento nástroj použiť na vytvorenie cieľového užívateľského rozhrania aplikácie.

Parametre dát vo formulároch

Jednotlivým widgetom je možné nastavovať parametre, napríklad či majú byť skryté, povinné, alebo akým počtom znakov sú limitované. Frontend developer môže v nástroji UI Designer vytvoriť triedy CSS, ktorý sa môžu následne použiť na rozšírenú štylizáciu widgetov.

UI Designer sa môže otvoriť priamo z nastavenia parametrov užívateľskej úlohy, čím sa na ňu napojí. Po uložení formuláru je automaticky uložený v aplikácii a pripravený na použitie. Po uložení je formulár možné použiť aj v iných úlohách, čo je praktické a formuláre sú tak recyklovateľé. V kombinácii s kontraktmi a operáciami úloh je práca s dátami a ich vizualizáciou vo formulároch priamočiara.



Obr. 4.4: UI designer. Prevzaté z [10]

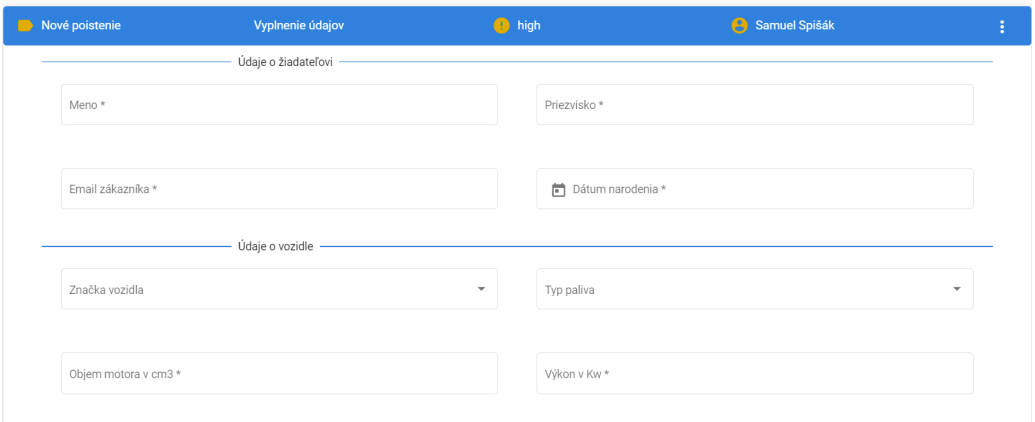
.

Netgrif

Generované formuláre

Vďaka napojeniu aplikačného enginu cez REST API na frontend, ktorý pokrýva technológia Angular, je umožnené automatické generovanie responzívnych formulárov z dát referencovaných v úlohách procesu. Preto som vytváranie formulárov v nástroji NAB (Netgrif Application Builder) nevyužil a vizuálny výsledok je stále dobrý, ako možno vidieť na obrázku 4.5.

Takisto je možné vďaka tomuto prepojeniu vytvárať dynamické formuláre so skrývaním a odkrývaním jednotlivých polí. To sa robí požitím API aplikačného enginu v ackiách v sieti, ktoré môžu byť vyvolané zmenou hodnoty procesnej premennej. Toto som využil pri vypĺňaní údajov žiadosti, kedy sa odkryje časť polí s informáciami o vozidle po vyplnení osobných údajov.



Obr. 4.5: Automaticky generovaný vizuál formulárov

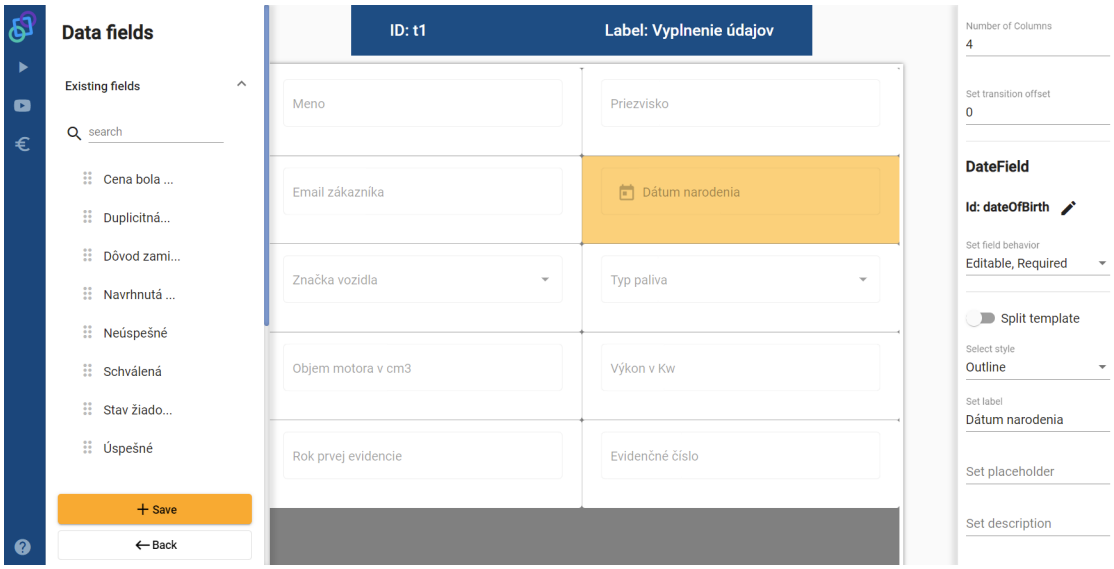
.

Vytváranie formulárov v nástroji Builder

Vytváranie formulárov prebieha v nástroji presúvaním procesných premenných do priestoru rozdeleného mriežkami, pričom sa podľa typu premennej vytvorí odpovedajúci typ formulárového elementu. Vizuálne možnosti a úpravy nie sú tak bohaté a komplexné ako v prípade nástroja *UI Designer* od spoločnosti Bonitasoft, ale postačujú.

Parametre dát vo formulári

Hlavnou výhodou vytvárania formulárov v tomto nástroji je automatické vloženie referencí na dáta definované v procese do úlohy. Taktiež je možné nastaviť jednotlivým dátam určité parametre a popisy. Dáta môžu byť viditeľné, povinné alebo skryté. Tieto parametre by sa mohli nazvať stavom premennej na danej úlohe, pričom tento stav je možné jednoducho meniť v akciách reagujúcich na zmenu hodnoty určitej položky vo formulári. Tak sa dajú jednoducho vytvárať dynamické formuláre, reagujúce na vstupy od užívateľa.



Obr. 4.6: Form builder

.

4.5 Užívatelia v procese

Keď sú v namodelovanom procese vytvorené dáta a úlohám priradené formuláre na vyplnenie potrebných údajov, je potreba vytvoriť užívateľov. Každý nástroj ponúka možnosti integrácie na existujúci systém na správu užívateľov (LDAP), na tie sa však ako som spomínal zameriavať nebudem.

Zameriavam sa na natívne podporované možnosti vytvorenia užívateľov a priraďovanie im potrebnej autorizácie. Tento krok je pred integráciou na existujúci systém na správu užívateľov súčasťou vývoja a testovania každej podnikovej procesnej aplikácie.

Camunda

Po spustení procesného enginu sa môže užívateľ pomocou administrátorského mena a hesla (uvedené v konfiguračnom súbore `application.yaml`) prihlásiť do *Camunda Webapps*, ktoré predstavujú predvolené užívateľské rozhranie na prácu s procesmi[8].

Vytváranie užívateľov

Vytváranie užívateľov prebieha v module *Admin*. Na vytvorenie nového užívateľa sa vyplní a formulár s osobnými údajmi. Takým istým spôsobom sa vytvárajú *Skupiny* užívateľov, ktorým sa môžu nastavia autorizácie a do skupín sa pridelia jednotliví užívatelia[9].

Autorizácia

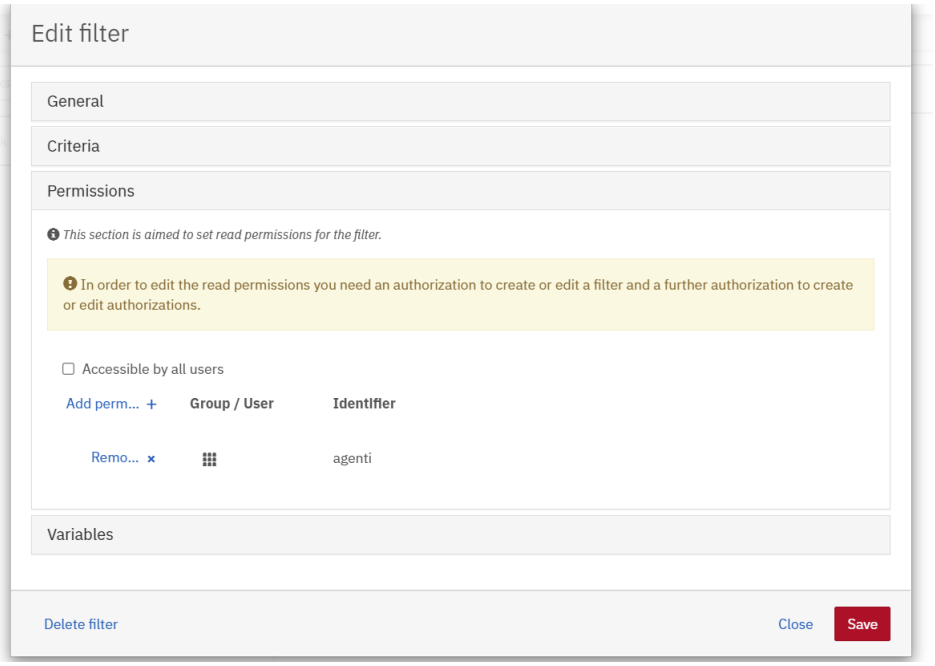
Nastavovanie autorizácie má široké možnosti, od oprávnení na vykonávanie taskov až po prístup k jednotlivým *Webapps* aplikáciam.

Bez explicitného povolenia autorizácie v konfigurácii enginu sa nastavované pravidlá autorizácie neaplikujú. Nemyslím si, že to je najlepšie riešenie. Minimálne by o tom mal byť Administrátor upozornený, aby nehľadal chybu v autorizačných pravidlách.

Absentuje možnosť priamo zakázať prístup ku konkrétnej úlohe pre skupinu užívateľov, alebo užívateľa. Maximálne sa dá nastaviť oprávnenie pre celý proces. Tým pádom môže hocikto prevziať všetky úlohy procesu, ktoré vidí v Tasklist aplikácii. Nie je to ale problém vyriešiť, pretože nástroj ponúka široké možnosti nastavenia filtrov, ktoré môžu byť nastavené tak, aby zobarzovali len tie úlohy, ktorých kandidátne skupiny alebo užívatelia sa nastavia v parametroch elementu úlohy v nástroji Modeler.

[8] Pokiaľ administrátorské údaje nie sú nakonfigurované, zobrazí sa formulár na vytvorenie administrátora

[9] Autorizácie sa môžu prideliť aj jednotlivým užívateľom



Obr. 4.7: Definícia filtru

.

Bonita

Vytváranie užívateľov

V nástroji Bonita Studio sa dá pomocou editoru organizačnej štruktúry jednoducho nadefinovať podnik s jeho zamestnancami vytváraním *súboru podnikovej štruktúry*. Táto štruktúra je prítomná v aplikácii ako súbor vo formáte XML. Definujú sa prihlasovacie a osobné údaje jednotlivých užívateľov aplikácie a zároveň skupiny a role, ktoré im budú následne priradené.

Autorizácia

Počas vytvárania diagramu procesu sa musia definovať aktéry, ktorý budú v procese prítomný a ktorý sa pred nasadením procesu namapujú na role vytvorené v súbore podnikovej štruktúry. Pokiaľ má byť proces spúšťaný manuálne, treba jedného z aktérou nastaviť ako iniciátora procesu. Aktéry sa definujú na dráhach, pričom všetky užívateľské úlohy budú pridelené automaticky aktérovi v dráhe, čo ušetrí čas. V prípadoch, že je v jednej dráhe viacero aktérov, môže sa vybrať pre jednotlivé úlohy manuálne.

Po namapovaní aktérov na role alebo skupiny cez výber *Server* → *Configuration* z vrchnej lišty nástroja Bonita Studio, bude automaticky nakonfigurovaná autorizácia jednotlivých úloh podľa aktérov. Autorizáciu teda nie je potreba dodatočne riešiť, napríklad cez filtre ako v prípade riešenia od spoločnosti Camunda. Upravovať štruktúru užívateľov je možné aj za behu aplikácie.

Profily

Profily slúžia na definíciu prístupu k jednotlivým pohľadom modulov procesnej aplikácie, ako sú užívateľský a administrátorský pohľad vo webovej aplikácii Bonita Portal. V komunitnej open-source edícii nie je táto možnosť a tak vo výslednej aplikácii je každému užívateľovi prístupný aj administrátorský pohľad, v ktorom môže pridávať a odoberať nasadené procesy a upravovať organizačnú štruktúru. Absencia tejto funkcie robí pre podniky, ktoré chcú využívať predvolené užívateľské rozhranie komunitnú verziu nástroja prakticky nepoužiteľnou.

Netgrif

Vytváranie užívateľov

V tomto smere riešenie od spoločnosti Netgrif zatiaľ zaostáva a na rozdiel od zvyšných porovnávaných riešení, toto neobsahuje nástroj na jednoduché vytváranie užívateľov alebo definíciu organizačnej štruktúry s jednotlivými zamestnancami a ich údajmi. Avšak je to pochopiteľné, vzhľadom na to, že nástroj zatiaľ nie je ponúkaný ako samostatný vývojový nástroj pre firmy, ktorých zamestnanci by si vytvárali procesnú aplikáciu sami. Procesné aplikácie sú vyvíjané zamestnancami firmy Netgrif a tie sú napájané s existujúcim systémom na správu užívateľov.

Existuje ale možnosť odoslať mail s odkazom na registráciu do aplikácie. Túto možnosť som využil a vytvoril tak troch užívateľov odpovedajúcich aktérom v mojom procese.

Autorizácia

Jednotlivých užívateľov môže autorizovať admin priradením rolí v jednotlivých procesoch užívateľovi na základe jeho pracovnej pozície. Role sa vytvoria v nástroji NAB. Roliam sa v ňom následne priraďujú možnosti manipulácie s procesom alebo jednotlivými úloham. Tak sa dá jednoducho zamedziť, aby niekto nemohol vykonať úlohu, na ktorú nemá kompetenciu.

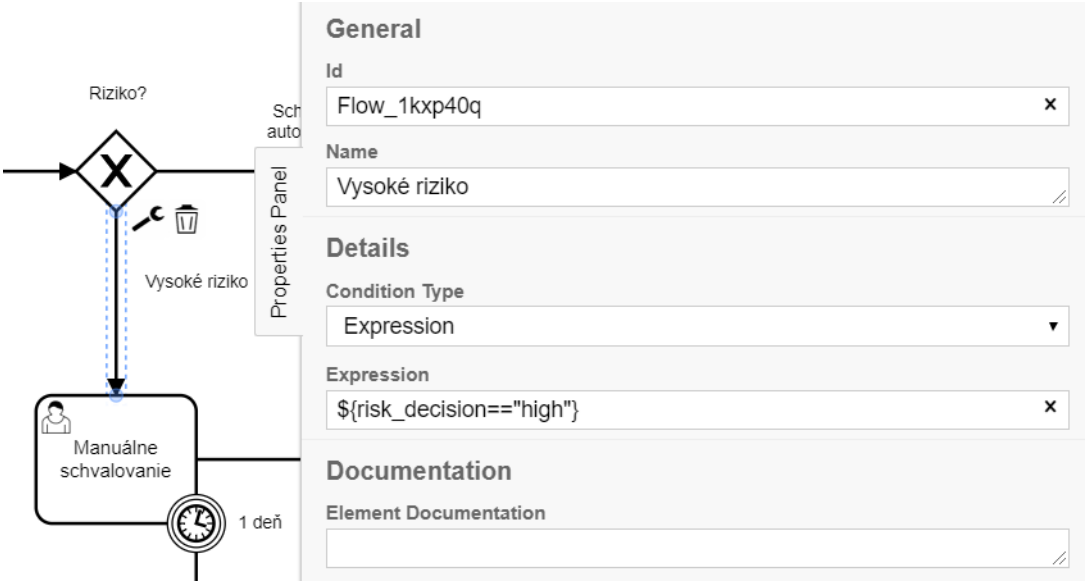
4.6 Výber smeru pracovného toku v procese

Táto sekcia porovnáva spôsoby, ktorými nástroje podmieňujú smer pracovného toku.

Camunda

Podmienky prechodu som v modeleri definoval na sekvenčných tokoch pomocou výrazového jazyka (ďalej zápis skratkou EL, z angl. Unified Expression Language). Je podporované aj spájane podmienok pomocou kľúčových slov *and/or*.

Ako príklad rozhodovania je uvedená situácia po vyhodnotení rizikovosti žiadosti o poistnenie . Sekvenčné toky vychádzajú z brány typu XOR, ktorá symbolizuje rozhodovanie v procese. Panel s parametrami sekvenčného toku je tejto skutočnosti prispôsobený možnosťou definovať podmienku prechodu. Pokiaľ má premenná `risk_decision` hodnotu „high“, proces pokračuje manuálnym schvaľovaním.



Obr. 4.8: Použitie EL na definovanie podmienky prechodu

Bonita

Podmieneným prechodom sa môže nastaviť podmienka prechodu pomocou skriptu, ktorý vracia hodnotu typu Boolean, alebo rozhodovacou tabuľkou. V prípade rozhodovacej tabuľky sa postupne validujú pravidlá. Prvé pravidlo, ktoré má splnené podmienky určí, či sa má daný prechod spustiť alebo nie.

Takisto je možné podmienky definovať výrazom EL (expression language). Výrazy sú však limitované na jednu porovnávanú hodnotu a popravde vo väčšine prípadov sa podmienky ani nevyhodnotili, preto som používal radšej jednoduché skripty.

Netgrif

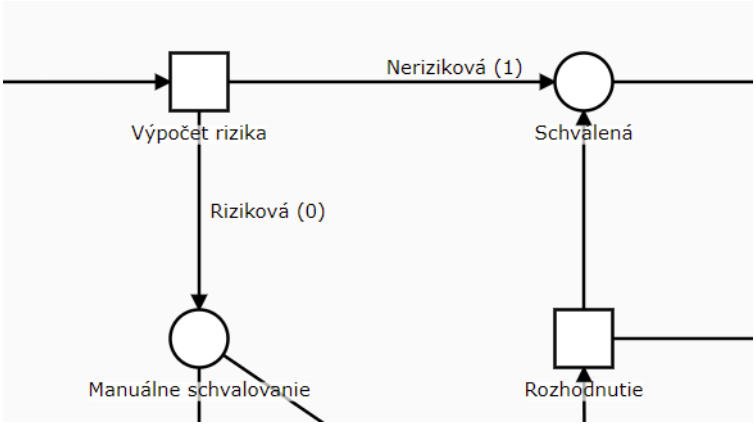
V prípade Petriho sietí, ktoré neobsahujú element rozhodovania resp. delenia pracovného toku, akým sú brány v prípade notácie BPMN, je rozhodvanie náročnejšie na implementáciu ako by sa mohlo zdať. Nie je napríklad na jednotlivých hranách možné definovať zloženú

podmienku, ktorá musí byť splnená aby sa prechod vykonal. Automatizáciu rozhodovania je možné docieliť pomocou kombinácie akcií a variabilných hrán.

Variabilné hrany

Sú špeciálnym typom hrán v jazyku Petriflow, ktorých váha sa určuje na základe premennej číselného typu. V prípade rozhodovania na základe hodnoty určitej premennej je tak potrebné v akcii skontrolovať hodnotu danej premennej a podľa toho zmeniť hodnoty číselných premenných, ktoré určujú váhy hrán vychádzajúcich z úlohy (prechodu), ktorý je práve vykonávaný[10].

Takýto prípad rozhodovania je možné vidieť na obrázku 4.9, kedy sa po určení rizikovsti žiadosti rozhoduje, či má byť žiadosť manuálne schvaľovaná.



Obr. 4.9: Použitie variabilných hrán na určenie vetvy, ktorou má proces pokračovať

[10]alebo z miesta, v ktorom sa práve proces nachádza, čím sa môže aktivovať jeden z prechodov, ktoré za miestom nasledujú a tak sa vyberie úloha, ktorá má byť vykonaná

4.7 Automatizované úlohy

Úlohy, ktoré umožňujú automatizovať alebo nahradiť prácu užívateľom v procese, sú prítomné vo väčšine podnikových procesov. Implementácia týchto akcii by mala byť čo najjednoduchšia. Aj keď tu sa už programovaniu vyhnúť nedá, mať k dispozícii jednoduché API na prístup k dátam a systémovým službám vie tento proces značne zjednodušiť.

Na začiatok opíšem všeobecné možnosti automatizácie v konkrétnom nástroji a potom porovnám implementáciu *úlohy na kontrolu duplicít* a *úlohy odosielania mailovej správy*.

Počas vykonávania úlohy na kontrolu duplicít sa zisťuje, či už nebola evidovaná žiadosť obsahujúca rovnaké evidenčné číslo, ako je momentálne prítomné v inštancii procesu. Úloha odosielania mailu po schválení žiadosti na zadanú mailovú adresu zákazníka odosiela informácie s číslom zmluvy a cenou poistného.

Camunda

Automatizácia začína v modeleri. Na jednotlivé fázy vykonávania úloh a prechodov v procese sa môžu nastaviť *listenery*.Pre automatizáciu, ktorá je dôležitou súčasťou procesu a má byť v diagrame viditeľne reprezentovaná sa vytvárajú samostatné úlohy v diagrame (Servisná alebo Skriptová úloha).

Listenery

Počas rôznych fáz prechodu tokenu určitým symbolom v diagrame sa môže aktivovať listener na danom mieste. Ten následne spúšťa akciu. Tá môže byť zapísaná priamo v modeleri výrazom EL (akcie na jeden riadok, ako priradenie hodnoty premennej, volanie metódy beanu Java z aplikačného kontextu), skriptom v jazyku Groovy (napr. výpočet odvodený z procesných premenných), alebo môže byť listener namapovaný na triedu Java nasadenú v procesnej aplikácii (ľubovoľná akcia).

Táto trieda musí implementovať rozhranie `JavaDelegate` s jedinou metódou `execute`, ktorá pristupuje k aplikačnému exekučnému kontextu cez API procesného enginu a ovláda tak nasadené procesy v aplikácii. Treba spomenúť, že pri rozdelovaní pracovného toku sa vytvoria osobitné synovské exekučné kontexty, ktoré nemajú priamy prístup k otcovskému kontextu celého procesu vytvorenom zároveň s procesnou inštanciou. Nemajú tak priamu možnosť ovplyvňovať automatizáciu v inom pracovnom toku. Takto je vytváraný hierarchický model exekučného kontextu. Dá sa síce pristupovať aj z nižších vrstiev na vyššie, ale je to nejasný koncept, ktorý nie je veľmi zdokumentovaný a vie brzdiť vývoj, pokiaľ sa s tým nepočíta [11].

Listener som použil napríklad na začiatku procesu na inicializáciu procesných premenných. Priradil som ho na fázu ukončenia počiatočnej udalosti. Implementovaný bol pomocou skriptu v jazyku Groovy, zapísaným priamo v Modeleri.

Úloha služby (Service task)

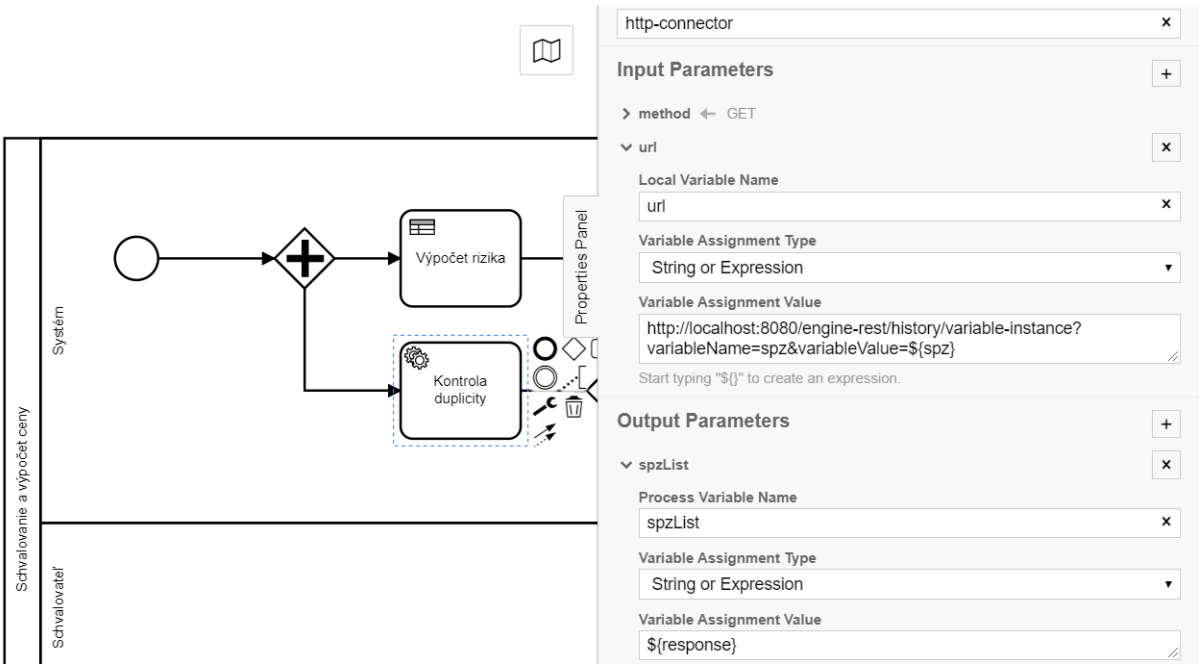
Okrem rovnakých možností automatizácie ako pri listeneroch, je pri tomto type úlohy možné v modeleri definovať *http connector* so vstupnými parametrami, ktorý bude posielať požia-

[11] `https://docs.camunda.org/manual/7.14/user-guide/process-engine/process-engine-concepts/#executions`

davku protokolom HTTP na špecifikovaný endpoint URL [12]. Je to vhodné pri využívaní webslužieb, alebo prístupe cez REST API k procesnému enginu, bez potreby vytvárania prístupu k nemu v kóde aplikácie. Dáta z odpovede sa dajú uložiť do samostatnej procesnej premennej ako jedna hodnota, alebo kolekcia viacerých hodnôt. Tento výsledok je možné parsovať priamo v modeleri pomocou skriptu v jazyku Groovy.

Implementácia služby na kontrolu duplicít

Kontrola duplicít bola implementovaná pomocou *http connectoru* definovaného v Modeleri, ktorý odosiela požiadavku typu GET cez REST API na procesný engine. Táto požiadavka[13] vyhľadá v ukončených inštanciách procesov tie, ktoré obsahujú názov premennej `spz` (hodnota evidenčného čísla vozidla). Telo odpovede sa uloží do premennej definovanj v Modeleri. Listener v podobe skriptu na ukončujúcej fáze úlohy premennú kontroluje. Pokiaľ je premenná resp. telo odpovede prázdne, znamená to, že neexistuje duplicitná historická žiadosť a logickej premennej `duplicity` je nastavená záporná hodnota.



Obr. 4.10: Konektor na kontrolu duplicity definovaný v Modeleri

.

Použitie REST API je relatívne dobre zdokumentované a umožňuje jednoducho manipulovať s procesmi v aplikácii, bez potreby priameho prístupu k API procesného enginu. Na umožnenie volaní požiadaviek v SOAP a REST formáte je potrebné dodatočne vložiť do aplikácie knižnicu *Camunda Connect*[14], ktorá základné connectory definuje.

[12] Podporuje základné definície pre formáty REST a SOAP

[13] `https://docs.camunda.org/manual/latest/reference/rest/history/variable-instance/get-variable-instance/`

[14] `https://docs.camunda.org/manual/7.14/reference/connect/`

Implementácia odoslania mailu s číslom zmluvy

Úlohu som implementoval pomocou HTTP connectora na prácu mailovými správami, ktorý je dostupný ako komunitné rozšírenie platformy[15]. Pridá sa do aplikácie ako knižnica a následne treba id connectora špecifikovať v Modeleri v parametri úlohy. Taktiež je treba do aplikácie vložiť konfiguráciu `mail-config.properties`, ktorá je dostupná na stránke rozšírenia.

Bonita

Definovanie automatizácie počas vytvárania modelu je v nástroji Bonita Studio uľahčené tým, že je zapnutý aplikačný engine. Dostupné premenné a kontext aplikácie sú registrované enginom a ponúkané pri vytváraní skriptov. Je tak jasne vidno, k čomu bude mať proces po jeho spustení prístup na určitom mieste.

Vytváranie skriptov

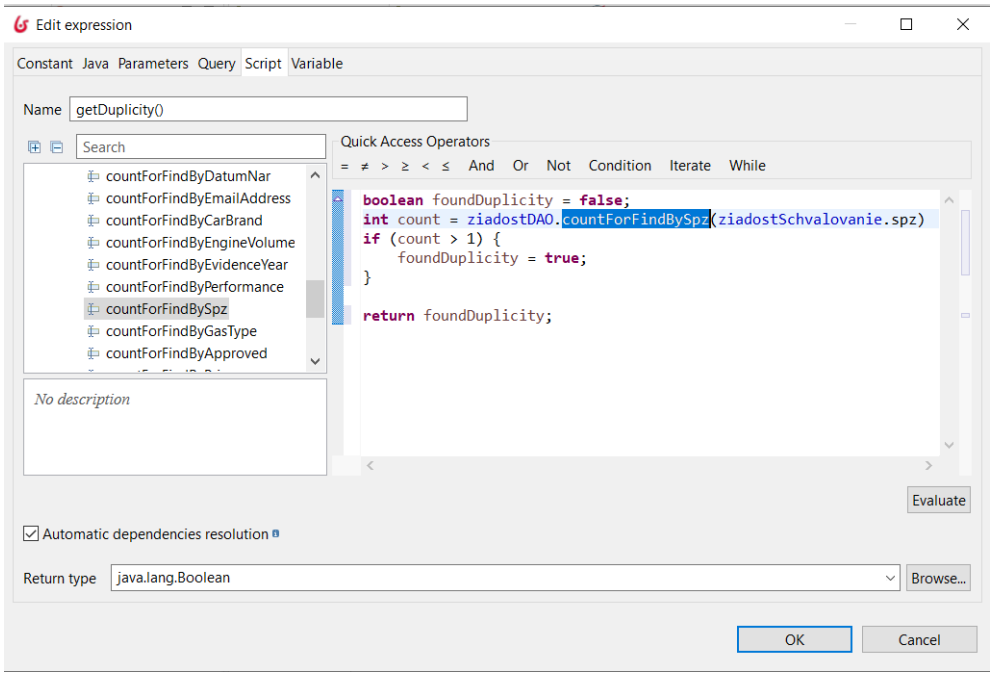
Automatizácia prebieha najčastejšie pomocou skriptov. Editor skriptov obsahuje strom prístupových objektov na engine a dátá, ktoré sú dostupné v danom kontexte úlohy na ktorej je skript definovaný. Po kliknutí na element sa do skriptu vloží príkaz alebo premenná, ktorá ho reprezentuje. Pri potrebe hľadaní v databáze je možnosť vybrať query podľa konkrétnej premennej dátového objektu procesu, ktorá následne prehľadáva dátové objekty podľa jej hodnoty. Celkovo prácu so skriptami hodnotím pozitívne a editor skriptov mi automatizáciu uľahčil.

Implementácia služby na kontrolu duplicít

Na skriptovej úlohe som vytvoril operáciu, ktorá mapuje logickú hodnotu, ktorá je výstupom skriptu na premennú dátového objektu, ktorá určuje, či bola žiadosť duplicitná.

Príkaz na nájdenie počtu dátových objektov s hodnotou konkrétnej premennej (evidenčné číslo) bol dostupný v strome skriptových príkazov. Stačilo tak do jej parametru priradiť hodnotu aktuálneho evidenčného čísla a skontrolovať, či výstup príkazu, teda počet objektov obsahujúcich dané evidečné číslo neprevyšuje hodnotu 1 (query vráti aj aktuálne používaný dátový objekt). Výsledok kontroly je vrátený skriptom a táto hodnota sa pomocou operácie vloží do určenej premennej po ukončení skriptovej úlohy.

[15] `https://github.com/camunda-community-hub/camunda-bpm-mail`



Obr. 4.11: Skript na kontrolu duplicity s využitím stromu rýchlich príkazov (Bonita Studio)

Implementácia odoslania mailu s číslom zmluvy

V nástroji Bonita Studio je na výber z mnoha konektorov na rôzne použitie kvôli potrebám integrácie, ktorých definícia prebieha jednoducho vďaka sprievodcovi ich konfiguráciou. V ponuke je aj mailový konektor, ktorý som na úlohe odoslania čísla zmluvy definoval ako vstupný konektor a s implementáciou tak nebol problém.

Netgrif

Na vykonanie práce, ktorú je potrebné v procese automatizovať sa používajú akcie. Implementujú ako samotné udalosti, tak aj reakcie na ne.

Sú prítomné priamo v súbore formátu XML, ktorý obsahuje definíciu procesu. Píšu sa v jazyku špecifickom pre vývoj Petriflow procesov založenom na jazyku Groovy[16]. V oficiálnej dokumentácií[17] je možné zistiť, aké príkazy zabezpečujúce určitú procesnú funkcionalitu sú použiteľné a aké sa plánujú do aplikačného enginu ešte len implementovať. Akcie v sieti môžu taktiež volať metódy definované počas vývoja v triede `ActionDelegate`, čo je vhodné pre možnosť použitia debuggeru. Akcie je možné vytvárať priamo v nástroji na modelovanie siete procesu.

Na rozdiel od zápisu BPMN, nie je v Petriflow sieti vizuálne vynačené, aká úloha resp. prechod prebieha automaticky, dá sa to usudzovať z názvu prechodu.[18] Automatické vykonanie prechodu je potreba definovať manuálne v štruktúre XML prechodu tagom `<trigger type="auto"/>`

[16] Klasické príkazy z jazyka Groovy je v nich taktiež možné použiť

[17] `https://petriflow.com/`

[18] Prechodu je možné priradiť ikonu, ktorá by typ použitia prechodu zobrazovala, ale ich výber zatiaľ v nástroji NAB chýba.

Čas vykonania akcie

Akcie je možné vykonať počas priraďovania, alebo počas ukonnčenia určitého prechodu v úlohe (reprezentovaná podprocesom, viď. 2.1.3).

Vďaka možnosti spustiť akciu ako reakciu na zmenu hodnoty určitej premennej je automatizácia schopná komplexných, dynamických operácii.

Štruktúra akcie

- **Deklarácia**: vytvorenie referencí na premenné, alebo úlohy v procese, pomocou ktorých sa bude na dané dáta procesu odkazovať.
- **Práca**: skript v jazyku Groovy volajúci metódy z Petriflow API, alebo vlastné metódy z triedy *ActionDelegate*

Ako príklad uvediem akciu na zmenu hodnôt premenných určujúcich váhy hrán v procese žiadosti, volanej v procese schvaľovania. Váhy hrán sa určia podľa toho, či bola v databáze nájdená duplicitná žiadosť s rovnakým evidenčným číslom.[19]

```
cisloSpz: f.spz,
duplicitna: f.2112,
neduplicitna: f.2111;

def cases = findCases( { it.dataSet.get("spz").value.eq(cisloSpz.value) } );

if (cases.size() > 2) {
        change duplicitna value {1}
        change neduplicitna value {0}
}
```

Implementácia odoslania mailu s číslom zmluvy

Kvôli absencii podpory editoru konektorov alebo priamo preddefinovaných konektorov pripravených na použitie bola implementácia odosielania mailu najnáročnejšia v porovnaní so zvyšnými nástrojmi. Konektor bol vytvorený manuálne v kóde aplikácie.

Podpora určitých typov konektorov s jednoduchou konfiguráciou by bol hodnotný prínos pre tento nástroj.

[19] Ak bolo nájdených viac ako dve procesné inštancie, tak okrem aktuálnych inštanci procesu žiadosti a schvaľovana v databáze existuje ukončená inštancia s rovnakou hodnotou premennej „spz“

4.8 Rozhodovanie na základe podnikových pravidiel

Na začiatku schvaľovacieho procesu sa určuje riziko poisťovacej žiadosti. Toto riziko sa určuje na základe krítérii, resp. na základe podnikových pravidiel.

Určenie rizika prebieha na základe zjednodušených pravidiel, ktoré postačujú na porovnanie schopnosti nástrojov ich vytvorenia a použitia. Zameranie porovnania je na jednoduchosť a flexibilitu použitia rozhodovacích pravidiel.

Riziko sa určuje podľa:

- Veku vodiča (Skúsenosti)
- Veku vozidla (Počítaný od roku prvej evidencie)
- Výkonu vozidla
- Značky vozidla

Camunda

Rozhodovanie bolo realizované pomocou natívne podporovanej implementácie rozhodovacích tabuliek vo formáte DMN priamo v Modeleri.

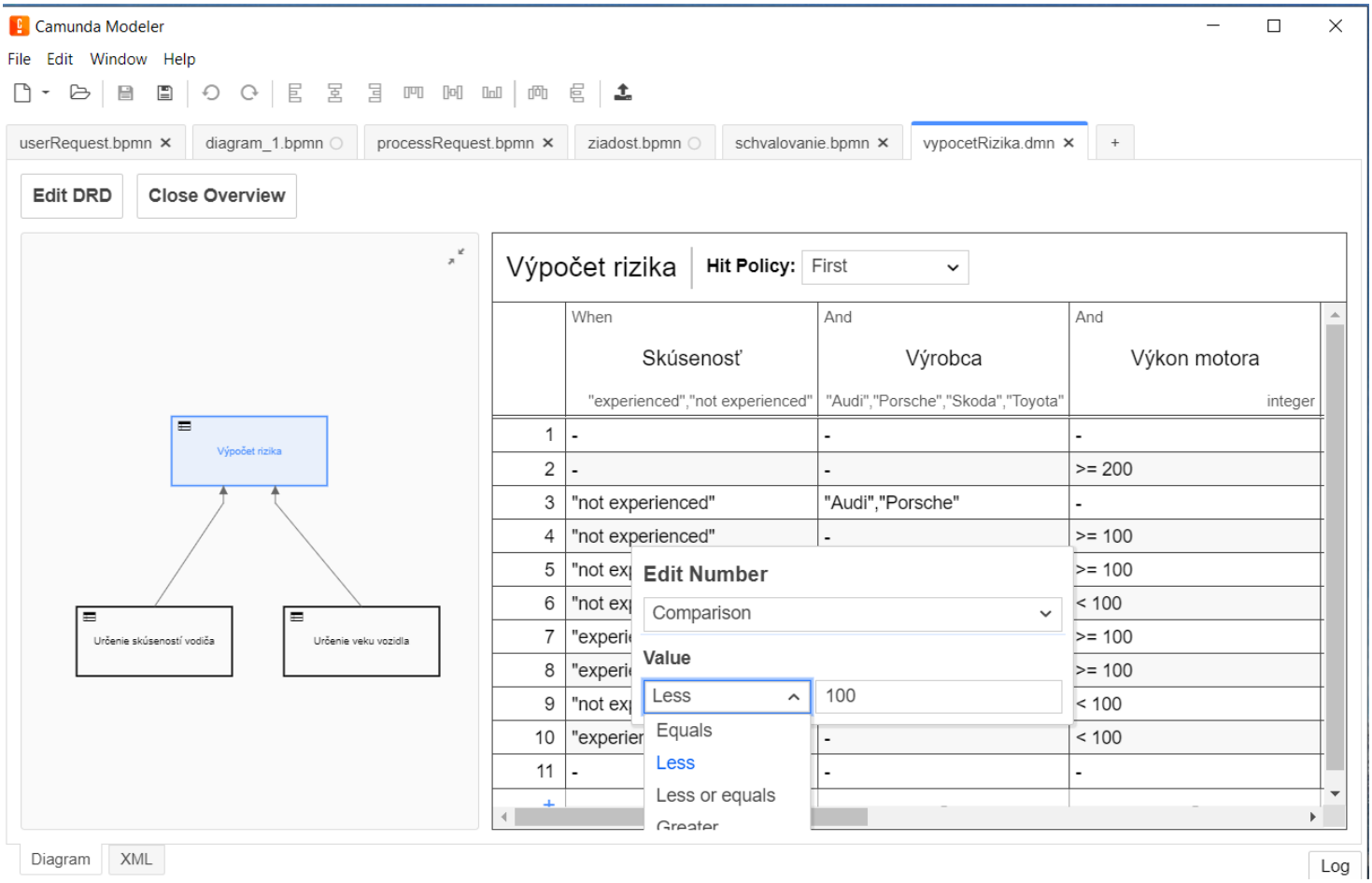
Modelovanie rozhodovacích tabuliek je intuitívne. Na uľahčenie vytvárania pravidiel sa môžu definovať textové hodnoty, ktoré môžu jednotlivé vstupné premenné nadobúdať. Následne pri vytváraní pravidla stačí jeden z ponúknutých vstupov vybrať a netreba ho ručne vypisovať. Pri číselnej hodnote v pravidle môže užívateľ určiť, či je očakávaný vstup menší alebo väčší ako zadaná hodnota, alebo môže priamo definovať interval prípustných hodnôt.

Jednotlivé tabuľky môžu na seba nadväzovať, kedy výstup z jednej tabuľky bude na vstupe druhej. Jednotlivé tabuľky a ich závisloti sú vizuálne reprezentované na ľavej strane obrazovky v aplikácii Modeler.

Možnosť napojenia tabuliek som využil tým spôsobom, kedy jedna tabuľka určuje podľa zadaného dátumu narodenia vodiča kategóriu jeho vodičských skúseností[20] a druhá tabuľka určuje vekovú kategóriu vozidla. Tieto tabuľky sú napojené na hlavnú tabuľku rozhodujúcu o výške rizika. Vďaka tomu je tabuľka rozhodujúca o riziku prehľadnejšia.

Výstup rozhodovania sa ukladá do premennej procesného rozsahu, ktorej meno sa špecifikuje v Modeleri v elemente rozhodovacej úlohy.

[20]Natívne podporované je porovnanie, či je dátum na vstupe menší alebo väčší ako konkrétny, manuálne zvolený dátum. To znamená nutnosť danú rozhodovaciu tabuľku aktualizovať a znovu nasadiť každý deň, čo nie je ideálne. No vďaka možnosti nasadiť novú verziu bez potreby zastaviť engine je to použiteľné riešenie

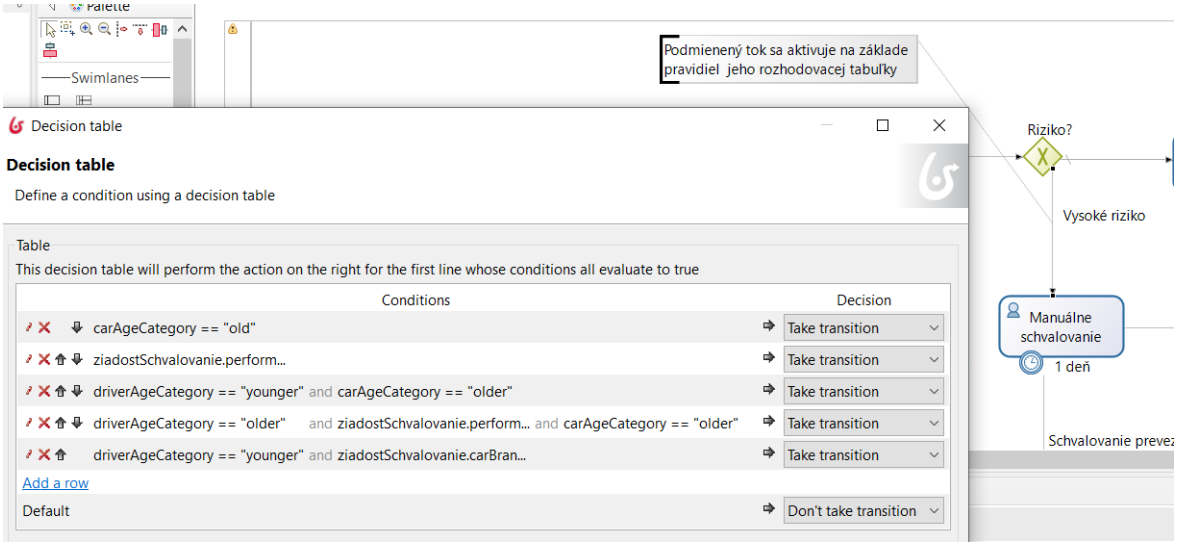


Obr. 4.12: Modelovanie rozhodovacej tabuľky v modeleri

Bonita

Rozhodovacie tabuľky sú síce súčasťou nástroja Bonita Studio, nedajú sa však použiť na samostatných úlohách. Neobsahujú výstup, čiže sa pomocou nich nedá určovať priamo hodnota rizika. Používajú sa na podmienených tokoch a jednotlivé pravidlá tabuľky určujú, či sa má daný tok aktivovať a v prípade mojej aplikácie tento spôsob stačil na implementáciu požadovanej funkcionality. Namiesto výberu ďalšej cesty v procese v exkluzívnej (XOR) bráne podľa dopredu určenej premennej obsahujúcej hodnotu rizika (vysoké riziko/stredne a nízke riziko), som podľa pravidiel určujúcich vysoké riziko rozhodoval, či pôjde žiadosť vetvou manuálneho schvalovania, alebo nie.

Ako som spomenul, v tomto prípade to funkcionalitu neovplyvnilo, ale nebolo možné pomocou rozhodovacích pravidiel určiť špecifickejšie výšku rizika.



Obr. 4.13: Rozhodovacie pravidlá na podmienenom prechode

.

Netgrif

Aplikačný engine čiastočne podporuje vytváranie podnikových pravidiel pomocou systému na správu obchodných pravidiel Drools[21]. Túto funkcionalitu je možné implementovať v aplikácii pomocou kódu v špecifickej syntaxi pre NAE. Nie je však podporované vizuálne vytváranie pravidiel, ktoré by ich tvorbu uľahčilo a implementácia bola pre potreby mojej aplikácie zbytočne náročná. Preto som určovanie rizika riešil programaticky vytvorením metódy `assessRisk` v triede *DemoActionDelegate* volanej v rámci akcie sieti.

[21] `https://www.drools.org/`

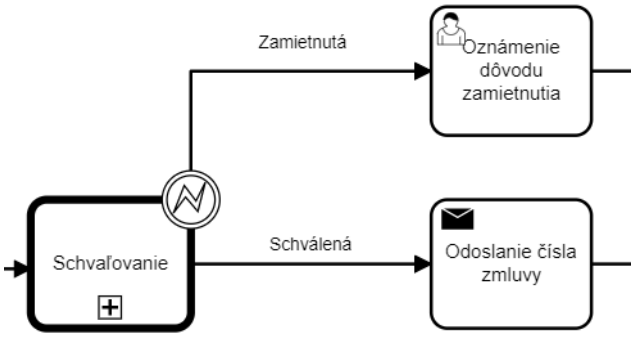
4.9 Volanie vzdialeného procesu

Z dôvodu prehľadnosti a praktickosti je časť schvalovania žiadosti definovaná ako samostatný proces. Tento proces obsahuje podnikovú logiku schvaľovania a kontroly žiadosti a je tak výhodné mať ho definovaný samostatne. Pokiaľ sa v budúcnosti bude meniť spôsob, akým má byť požiadavka schvaľovaná, alebo bude treba schvaľovať iné typy požiadaviek, môže sa proces rozšíriť a byť použiteľný ako súčasť viacerých podnikových procesov. Tento spôsob implementácie umožňuje aj prípadné hromadné schvaľovanie viacerých požiadaviek, kedy sa v inštancii, ktorá údaje požiadaviek vytvorí naraz zavolá tento proces pre každú požiadavku samostatne.

Inštancia tohto procesu je vytváraná automaticky potom, ako sa vyplnia údaje novej žiadosti. V tejto časti porovnávam implementáciu volania tohto vzdialeného procesu a to, akým spôsobom komunikuje s procesom, ktorý ho volá.

Camunda

Volanie vzdialeného procesu je v procese vytvárania žiadosti reprezentované elementom volania aktivity (Call activity). V Modeleri sa špecifikuje názov procesu, ktorý je nasadený v aplikácii. Na element som pridal udalosť zachytávajúcu predčasné ukončenie procesu schvaľovania (Error event), ktorá v prípade zachytenia udalosti predčasného ukončenia smeruje pracovný tok na úlohu, v ktorej oznamuje obchodný zástupca zákazníkovy dôvod tohto predčasného ukončenia schvalovania (v tomto prípade buď nájdenie duplicitnej žiadosti, alebo zamietnutie žiadosti schvalovateľom). Dôvod je špecifikovaný v správe udalosti zachytávajúcej toto ukončenie a uložený v procesnej premennej, ktorej obsah je následne zobrazený v úlohe zástupcu.



Obr. 4.14: Volanie aktivity s udalosťou zachytávajúcou jej chybné ukončenie

.

Zdieľanie dát medzi procesmi

Dáta, ktoré vstupujú do vzdialeného procesu som definoval v Modeleri v záložke *Variables*. Hodnoty špecifikovaných premenných sa skopírujú do volaného procesu, v ktorom sa automaticky inicializujú odpovedajúce procesné premenné s odpovedajúcimi hodnotami. Na schvaľovanie boli potrebné takmer všetky údaje o zákazníkovy a vozidle, a tým pádom mi stačilo ako vstupný parameter zadať kľúčové slovo *all*. To, že sa nezdieľajú rovnaké pre-

menné medzi procesmi môže byť nevýhoda. V mojom prípade tak po ukončení schvaľovania a uzavretí žiadosti ostávajú v databáze historické duplicitné hodnoty.

Pokiaľ je počas behu volaného procesu potreba zmeniť dáta volajúceho procesu, dá sa tak urobiť pomocou skriptu s využitím `RuntimeService` služby, ktorá umožňuje nastaviť hodnoty premenných na konkrétnej inštancii ktorá je špecifikovaná svojou id hodnotou. Id volajúceho procesu som mal uložené v samostatnej premennej, ktorá bola do volaného procesu skopírovaná spolu s ostatnými údajmi. Po určení ceny vo schvaľovacom procese sa jej hodnota aktualizovala aj vo volajúcom procese. Totiž pokiaľ má schvaľovací proces skončiť úspešne, musí byť najskôr vykonaná úloha akceptácie ceny zákazníkom, ktorú vykonáva obchodný zástupca. Akceptácia je oznámená procesu schvaľovania, ktorý tak končí a v otcovskom procese sa prechádza do poslednej aktivity v životnom cykle žiadosti: Odoslanie čísla zmluvy na mail zákazníka.

Bonita

Proces schvaľovania bol volaný ako vzdialený proces pomocou úlohy volania aktivity. Bežiaci engine v rámci prostredia nástroja Bonita Studio registruje všetky procesy, ktoré obsahuje a tak mi pri špecifikácii volaného procesu bol schvaľovací proces ponúknutý automaticky.

Vytvorená inštancia má natívne k dispozící identifikačné číslo koreňového procesu, ktorý ju vytvoril a tak je komunikácia medzi synovským a otcovským procesm uľahčená.

Zdieľanie dát medzi procesmi

Parametre úlohy volania aktivity obsahujú okrem iného dáta na odoslanie do aktivity a dáta na prijatie po jej ukončení. V schvaľovacom procese stačilo definovať rovnaký typ dátového objektu aký je v procese vytvárania žiadosti a následne som jednotlivé objekty v parametri spojil. Výhodou dátových objektov je, že sú zdielateľné medzi inštanciami procesov a tak je počas celého životného cyklu žiadosti používaný ten istý objekt a zmeny, ktoré sú v ňom vykonané vo volanom procese, sú prítomné aj vo volajúcom (otcovskom) procese.

Netgrif

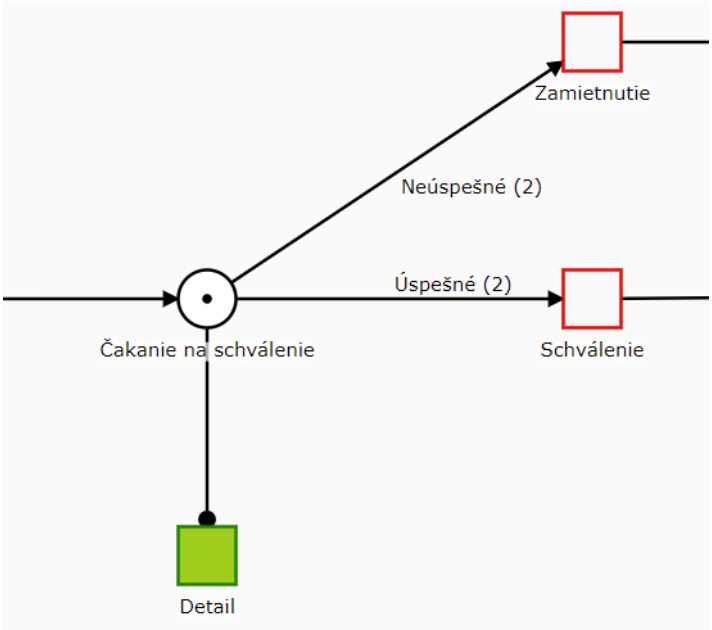
Z charakteru Petriho sietí nie je možné mať vykonávanie vzdialeného procesu reprezentované samostatným elementom napojeným na pracovný tok vo volajúcom procese ako v prípade notácie BPMN, kde je vzdialený proces vizuálne reprezentovaný elementom volania aktivity. Vytvorenie novej inštancie určitého procesu a zároveň zabezpečiť pokračovanie vykonávania práce vo volajúcom procese je potrebné vytvoriť ručne pomocou akcie. Znižuje to tak transparentnosť diagramu a pridáva prácu navyše. Spôsob medziprocesnej komunikácie je ale definovaný dobre a dosiahnuť túto funkcionalitu bolo pomocou API aplikačného enginu pomerne jednoduché a nebol s ňou problém.

Akcia vytvorenia novej inštancie

Na začiatok bolo treba špecifikovať, ktorú sieť má proces vytvoriť. To bolo docielené príkazom `def net = petriNetService.getNewestVersionByIdentifier("schvalovanie")`, ktorý objekt siete uložil do premennej *net*, ktorá bola predaná príkazu *createCase* spolu s názvom novej inštancie, ktorý je určený evidenčným číslom vozidla: `def caseSchvalovanie = createCase(net, cisloEvidencie.value)`

Medziprocesná komunikácia

Pri vytvorení inštancie som si ju v akcii priamo uložil do premennej *caseSchvalovanie*. Mohol som do tak inštancie vložiť identifikačné číslo inštancie žiadosti, aby sa nová inštancia mohla odkazovať na proces, ktorý ju vytvoril, keďže inštancie neexistujú v hierarchickej štruktúre ako v prípade nástroju od Bonitasoft alebo Camunda a treba s tým počítať. V príkaze na nastavenie dát vo vzdialenom procese je tiež potrebné špecifikovať *aktívny prechod* vytvorenej inštancie siete, na čo je treba tiež myslieť pri vytváraní siete[22]. Je to potrebné kvôli spôsobu reprezentácie dát v databáze. Z tohto dôvodu je v mieste procesu žiadosti, v ktorom sa čaká na schválenie, definovaný prechod spojený "read" hranou, ktorý je stále spustiteľný (viď. obrázok **??**). Tento prechod má sám o sebe funkciu zobrazenia detailov žiadosti, ale zároveň slúži ako spôsob prístupu k dátam a ich zmene v procese žiadosti počas vykonávania procesu schvaľovania.



Obr. 4.15: Spustiteľná úloha detailu počas čakania na schválenie

[22]Prechod, ktorý má byť stále aktívny (spustiteľný) v určitom stave procesu, je potrebné spojiť s miestom "read" hranou, ktorá sa používa na vytvorenie úlohy detailu inštancie

4.10 Implementácia udalostí v procese

V tejto sekcii je opísaná práca s udalosťami a reakciami na ne. Tie bolo potrebné nastaviť pri:

- Nájdení duplicitnej žiadosti
- Zamietnutí žiadosti schvalovateľom
- Neaktivite schvalovateľa po časovom limite
- Akceptácii ceny zákazníkom
- Neakceptácii ceny zákazníkom

Camunda

Chybové udalosti

Na implementáciu „negatívnych“ udalostí v procese schvaľovania (nájdenie duplicity, zamietnutie žiadosti) boli použité ukončujúce udalosti oznamujúce chybné ukončenie *Throwing Error End Event*, ktoré ukončia tok práce v schvaľovaní a túto udalosť spolu so správou chyby propagujú do otcovského procesu.

V otcovskom procese je na úlohe *volania aktivity (Activity Call)* pripojená udalosť zachytávajúca chyby v procese (Catching Error Event), ktorá uloží informáciu o chybe, ktorú tieto udalosti v správe odosielajú, do premennej `error_message`. Táto premenná je zobrazená v nasledujúcej užívateľskej úlohe „Oznámenie dôvodu zamietnutia“, kde obchodný zástupca oznamuje užívateľovi, prečo bola jeho žiadosť zamietnutá a celý proces sa ukončuje. Tieto udalosti sú vizuálne reprezentované v diagrame procesu a boli implementované pomocou parametrov v Modeleri.

Časová udalosť

Udalosť neaktivity schvaľovatela je v diagrame viditeľná ako priebežná prerušujúca udalosť časovača pripojeného k úlohe schvalovania. Z neho vychádza sekvenčný tok do úlohy schvalovania nadradeným, ktorý do tejto úlohy po uplynutí časového limitu automaticky posúva pracovný tok. Čas sa definuje zápisom vo formáte *ISO 8601*[23]. Časovač sa môže aktivovať aj periodicky, čo je výhodné použiť, pokiaľ je treba napríklad raz za určenú dobu poslať upozornenie, alebo v prípade potreby opakovane po určitej dobe vytvárať novú procesnú inštanciu určitého procesu (napr. proces mesačnej sumarizácie vykonanej práce v podniku).

Udalosti správ

Očakávané udalosti procesu (zobrazenie navrhnutej ceny po schválení žiadosti, akceptácia ceny), sú v diagrame zobrazené ako *úlohy odosielania správy (send message task)*. Implementované sú pomocou triedy `ProcessMessagingDelegate`, ktorá implementuje rozhranie `JavaDelegate`. V Modeleri je na daných úlohách odosielania správy špecifikovaná referencia na túto triedu, ktorej metóda `execute` sa po príchode tokenu do danej úlohy zavolá. Táto metóda pristupuje cez API procesného enginu k službe `RuntimeService` a vykonáva koreláciu správ na cieľovú udalosť prijatia správy podľa lokálnej premennej obsahujúcej

[23] `https://en.wikipedia.org/wiki/ISO_8601#Durations`

názov správy. Premenná lokálneho rozsahu *messageName*, ktorá špecifikuje názov správy, je pridaná na úlohu odosielania správy ako vstupná premenná v Modeleri. Názov správy je uvedený v parametri čakajúcej udalosti správy, ktorá po prijatí správy vytvára v odchádzajúcom sekvenčnom toku token a vytvára tak nový pracovný tok.

Bonita

Chybové udalosti

Rovnako ako v prípade nástroja od spoločnosti Camunda, aj tu je pri nájdení duplicity presmerovaný pracovný tok procesu schvaľovania do ukončujúcej chybovej udalosti (Throwing Error End Event). Tá ukončuje predčasne ukončuje proces schvaľovania a pracovný tok v otcovskom procese vychádza z udalosti zachytávajúcej tieto chyby do úlohy oznámenia dôvodu zamietnutia zákazníkovi. Na rozdiel od Camundy nie je možné definovať a odoslať spolu s udalosťou chyby aj chybovú správu. Toto správanie je potrebné ošetriť. Vďaka tomu, že oba procesy pracujú s tou istou inštanciou dátového objektu, všetky dáta zmenené počas procesu schvaľovania sú dostupné aj vo volajúcom procese. V úlohe oznámenia dôvodu zamietnutia som preto do formuláru vložil Boolean premennú `duplicity`. Pokiaľ je táto hodnota vo formulári zaškrtnutá, znamená to, že premenná `duplicity = true` a dôvodom je nájdenie duplicity. V druhom prípade hodnota nebude zaškrtnutá, čiže duplicita nebola nájdená a žiadosť musela byť zamietnutá schvalovateľom. Do formuláru bol spôsob vyhodnotenia dôvodu vložený ako text pomocou nástroja UI Designer. Absenciu chybových správ ale považujem za nedostatok nástroja, uľahčilo by to spracovávanie neočakávaných udalostí.

Časová udalosť

Táto udalosť je priradená k úlohe manuálneho schvalovania, z ktorej vychádza sekvenčný tok aktivovaný po určitom čase. Čas, kedy sa má prechod aktivovať sa nastavuje editorom časových podmienok, pričom je možné nastaviť konkrétny dátum a čas, alebo časový limit. Z pohľadu implementácie je to intuitívnejšie ako v prípade nástroja od Camundy, kde sa používa špecifický zápis vo formáte ISO8601. Na druhej strane, v komunitnej verzii nie je možné časovač spúšťať periodicky, čo môže byť nevýhodou a v prípade potreby vykonávať určité úlohy alebo udalosti cyklicky, je potrebné prispôsobenie diagramu.

Udalosti správ

Tieto udalosti mali v prípade mojej aplikácie slúžiť na kontrolu pracovného toku medzi dvoma procesmi (procesom vytvárania žiadosti a procesom schvaľovania). Čo sa mi v tomto nástroji pri implementácii správ páčilo, bola jednoduchosť konfigurácie správy, čo zahŕňa určenie cieľovej úlohy, ktorú mi ponúkol na výber procesný engine automaticky a spôsob korelácie udalosti odosielania a prijímania správy na konkrétnu inštanciu podľa vloženej premennej. Oproti nástroju od Camunda Modeler je jednoduchšie posielať so správami aj hodnoty, ktoré sa uloženia v cieľovom procese (V prípade Camundy to treba robiť v Java triede, alebo skripte).

Narazil som ale na problém, kedy udalosť odosielajúca správu rušila pracovný tok v procese v ktorom bola vyvolaná. V dokumentácii nie je toto správanie definované a tak neviem, čo bolo príčinou. Musel som tak diagram prerobiť a odstrániť udalosti správ, pretože všetky komunikovali medziprocesne a rušili svoje inštancie, čím zabraňovali dokončeniu inštancie procesu do konca. Posielanie správ v rámci jedného procesu funguje správne.

Netgrif

Chybové udalosti

Podľa výsledku úlohy hľadania duplicity a schvaľovania žiadosti som nastavoval hodnotu logickej premennej. Podľa nej sa následne určujú hodnoty premenných, ktoré predstavujú váhy variabilných hrán. Tak bol určený počet tokenov, ktorý sa po týchto hranách po aktivácii prechodu z ktorého vychádzajú odošle na miesta do ktorých vchádzaju. Inak povedané, ktorou vetvou sa pracovný tok v procese posunie ďalej. V prípade udalosti, kedy má byť žiadosť zamietnutá sa token presunul do vetvy obsahujúcej jeden automaticky spúšťaný prechod s akciou, v ktorej boli nastavené váhy variabilných hrán čakajúceho procesu žiadosti tak, aby sa token toho procesu presunul do vetvy s úlohou oznámenia dôvodu o zamietnutí. Potom, ako sa tento automatizovaný prechod spustí, token končí v mieste z ktorého nevedú žiadne výstupné hrany. Keďže iný token v procese nie je, proces schvaľovania automaticky končí.

Časovaný prechod

Pokiaľ prechod v štruktúre XML obsahuje tag `<trigger type="time">`, znamená to, že sa spúšťa po určitom čase automaticky[24]. Čas sa určí v tagu `<delay>` vo formáte ISO 8601, tak ako v prípade riešenia od spoločnosti Camunda. No na rozdiel od toho riešenia tu nie je možné zadať dobu, po ktorej by sa mal prechod opakovane vykonávať. Samozrejme, dá sa to vhodným namodelovaním siete docieliť, ale to má za následok horšiu vizuálnu kvalitu diagramu procesu.

Udalosti notifikácii

To, na čo sú určené v notácii BPMN 2.0 udalosti správ alebo signálov, je potrebné robiť v sieti Petriflow pomocou akcií, vhodného modelovania prechodov a manuálnej korelácie podľa určených hodnôt. V bežných prípadoch ako v prípade mojej aplikácie, je to dosť práce navyše. Zároveň je Petriho sieť spojitý graf, tým pádom nevie byť v jednej sieti oddelený podproces, ktorý by bol aktivovaný udalosťou, čo je možné v diagrame BPMN pomocou počiatočných zachycujúcich udalostí docieliť pomerne jednoducho.

Treba ale poznamenať, že pokiaľ už vývojár rozumie správaniu sietí v jazyku Petriflow a vie používať procesné API, môže správy medzi procesmi a v rámci procesu korelovať ľubovoľným spôsobom a nemusí sa obmedzovať štandardnými konvenciami.

[24]Prechod musí byť spustiteľný, čiže počas uvedeného časového limitu musí na vstupnom mieste prechodu existovať počet tokenov odpovedajúci váhe vstupnej hrany

4.11 Nasadenie a testovanie aplikácie

Rýchle prototypovanie je dôležitá súčasť každého z porovnávaných nástrojov a tak začnem porovnáním, do akej miery je možné implementovať proces už počas jeho prvotného návrhu v modelovacích nástrojoch. V súvislosti s testovaním aplikácie opisujem náročnosť zmien a opráv v procesoch a rýchlosť prenesenia daných zmien do aplikácie. Rýchlosť zmien je ovplyvňená aj spôsobom, akým sa aplikácia nasadzuje.

Testovanie

aplikácií prebiehalo vytváraním nových inštancí procesu vytvárania žiadosti a vykonávaním jednotlivých užívateľských úloh poverenými užívateľmi. Jednotlivým inštanciám boli vo formulároch zadávané rôzne hodnoty tak, aby sa overil správny výber smeru pracovného toku, dostupnosť všetkých miest, úspešné vykonanie automatizácie a tak splnenie celkovej požadovanej funkcionality aplikácie.

Počas testovania som narazil na viacero neočakávaných chýb. Pri testovaní som tak mohol overiť schopnosť nástrojov príčiny chýb nájsť pomocou monitorovania inštancií. Taktiež opisujem schopnosť nástrojov rýchlo uplatňovať zmeny po oprave chyby. To spočíva v tom. akým spôsobom sa nasadadzuje nová verzia zmeneného procesu do aplikácie a kedy je potrebné aplikáciu reštartovať.

Camunda

Vytvorenie prototypu v nástroji Modeler

Vytváranie diagramu je vďaka intuitívnemu rozhraniu nástroja Modeler rýchle. Výber napájaných elementov na aktívny element sa prispôsobuje jeho typu. Spôsob, akým bude tok práce postupovať je z charakteru notácie BPMN od začiatku viditeľný a rozhodovanie na základe procesných dát je možné definovať pomocou výrazového jazyka. Jednotlivým úlohám môžu byť špecifikované parametre, ktoré ich napájajú na triedy vo vývojovom prostredí. Elementom udalostí je možné tiež špecifikovať základné parametre, ale samotnú implementáciu je potrebné najprv zabezpečiť vytvorením metód alebo konektorov vo vývojovom prostredí. Vytváranie dát je možné aj v modeleri pomocou skriptu, ale nie je to transparentný spôsob definície procesných dát. Vytváranie a úprava formulárov na úlohách užívateľa obsahuje len základné možnosti. Modelovanie rozhodovacích tabuliek je intuitívne. Ich následné napojenie na proces tiež.

Zmena verzie a nasadenie procesu

Pri zmene, ktorá sa týka súboru XML, ktorý predstavuje definíciu procesu, resp. pri zmenách v Modeleri je možné do bežiaceho procesu nahrať jedným kliknutím novú verziu procesu. Bežiace inštancie pokračujú vo vykonávaní verzie procesu, ktorú začali vykonávať. Každá nová inštancia, pokiaľ to nemá špecifikované inak, bude spúšťať novú verziu procesu.

Jednotlivé verzie a ich bežiace inštancie sa dajú zobraziť vo webovej aplikácii *Cockpit*. Tá ponúka aj pohľad na diagram jednotlivých verzí procesu a zobrazuje tokeny, resp. miesta, v ktorých sa inštancie procesov práve nachádzajú. Tento nástroj bol počas vývoja a testovania veľmi užitočný, nakoľko zobrazuje aj procesné premenné a ich hodnoty.

Transakcie

Tento prístup je výhodný pre testovanie aplikácie. Pokiaľ napríklad užívateľ zadá chybný údaj používaný na určitý výpočet, ktorý nasleduje v procese, transakcia nebude úspešná a riadenie sa vráti na úlohu užívateľa. Užívateľ môže následne údaj opraviť a úspešne ukončiť svoju úlohu.

V mojom prípade mi tento prístup umožnil opraviť a nasadiť nanovo volaný proces schvaľovania za behu aplikácie, bez nutnosti vytvárať nový proces žiadosti. Po úlohe vyplnenia údajov žiadosti sa volal vzdialený schvalovací proces v ktorom vznikla chyba vo výpočte ešte predtým, ako sa riadenie stihlo dostať do čakajúceho stavu. Celá inštancia volaného procesu sa tak zahodila a po nasadení novej verzie schvaľovacieho procesu som mohol pokračovať znova od potvrdenia vyplnených údajov žiadosti a následne už bola volaná najnovšia verzia schvaľovacieho procesu.

Sledovanie inštancii počas vykonávania aplikácie

Webová aplikácie Tasklist (slúžiaca ako predvolené užívateľské rozhranie) umožňuje počas vypĺňania formuláru v úlohe zobraziť diagram procesu a miesto, v ktorom sa daná úloha nachádza, čo zamestancovi zlepšuje vhľad do pracovného postupu. Vo webovej aplikácii Cockpit je zas adminovi umožnené sledovať nasadené procesy, miesta v diagrame v ktorých sa nachádzajú procesné inštancie a jednotlivé procesné dáta ktoré obsahujú. Čo sa týka monitorovania procesov, Camunda to má zvládnuté veľmi dobre. Okrem monitorovania práce v podniku mi to pomohlo aj vo vývoji.

V platenej verzii sú možnosti monitorovania inštancii oveľa širšie. Umožňujú zobraziť kompletné zmeny stavov a dát počas jednotlivých krokov akými proces prebiehal. Pokiaľ niekde nastala chyba je tak možné dopracovať sa k jej príčine. To platí aj pri rozhodovacích tabuľkách, kde je zobrazané ako sa vyhodnotili ich pravidlá. Vizál tohoto nástroja je prítomný v sekcii predstavenia komponentov nástrojov 2.2.1.

Bonita

Vývoj v nástroji Bonita Studio je uspôsobený na rapídny vývoj tým, že je počas vytvárania procesu zapnuté behové prostredie s procesným Enginom a portálom. Je to odlišný spôsob vývoja od zvyšných nástrojov a má svoje výhody.

Postup vytvárania prototypu procesu

Vytváranie procesu v nástroji Bonita Studio by malo prebiehať konkrétnym postupom, čo zabezpečuje vývoj všetkých súčastí procesnej aplikácie ešte pred prvotným nasadením:

1. Vytvorí a nasadí sa podnikový model dát a jeho objekty
2. Vytvorí a nasadí sa organizačná štruktúra
3. Vytvorí sa diagram procesu

Týmto spôsobom je umožnené využiť výhody nástroja Bonita Studio. Po nasadení modelu dát je možné jeho jednotlivé objekty priradiť procesu, vďaka čomu sa môže automatizovať ich inicializácia. Jednotlivé položky objekt sa môžu priradiť do kontraktu úlohy užívateľa, čím sa automaticky vytvorí formulár obsahujúci položky z kontraktu a operácie,

ktoré vstup z formuláru na položky dátového objektu namapujú a po ukončení úlohy ich aktualizujú.

Formuláre je v prípade úloh užívateľa s rovnakými dátami možné recyklovať, čo taktiež napomáha rýchlosti vytvárania prototypu procesu.

Nasadenie novej verzie procesu

Upravený proces je možné nasadiť do behového prostredia bez zastavenia enginu. Samotné diagramy su v špecifickom formáte `.proc` a obsahujú všetku automatizáciu v procese. Tá môže byť v podobe skriptov vykonávaných v operáciach jednotlivých úloh, alebo vstupných a výstupných konektorov, ktoré môžu pristupovať k externým zdrojom alebo databázam a zabezpečovať tak komplexnú automatizáciu v procese.

Tým pádom aj komplexné úpravy procesu, ktoré by v iných vývojových prostrediach trvalo nasadiť a otestovať dlhšie, je v nástroji Bonita Studio možné vykonať pomerne rýchlo. Asi najväčšou nevýhodou s ktorou som sa počas vývoja stretol, je pri potrebe zmeniť dátové položky dátového objektu nutnosť nasadiť znova celý model podnikových dát. Treba preto dátový objekt definovať na začiatku opatrne. Pokiaľ sa z procesu odstráni, odstránia sa aj všetky skripty v procese, ktoré s ním manipulujú.

Sledovanie inštancii počas vykonávania aplikácie

Aplikácia Bonita Portal predstavujúca užívateľské rozhranie umožňuje v administrátorskom pohľade zobraziť bežiace aj ukončené inštancie procesov. Obsahuje však len základný pohľad. Zobrazuje v akom stave sa inštancia nachádza, aké úlohy boli vykonané a aké hodnoty dát obsahuje. Pokiaľ nastala niekde chyba, je možné zistiť v akej úlohe, ale čo ju spôsobilo nie a príčinu treba hľadať v logoch procesného enginu ukladaných do textového súboru v nástroji Bonita Studio. Nevýhodou tohto nástroja je absencia debugovacieho režimu, ktorým by sa dalo vykonávanie procesu postupne sledovať[25].

[25]Režim je možno dostupný v platenej verzii, ale nepodarilo sa mi to o tom nájsť bližšie informácie.

Netgrif

Vytvorenie prototypu v nástroji Builder

V nástroji je možné po vytvorení siete špecifikovať role a ich autorizáciu v rámci úloh alebo procesu. Môžu sa vytvoriť dáta, ktoré budú po nasadení procesu automaticky inicializované. Následne sa dáta môžu priradiť jednotlivým úlohám v ktorých majú byť viditeľné pomocou formulárov, alebo sa môžu konkrétne premenné priradiť hranám v rámci implementácie rozhodovania v procese. Formuláre je možné základne vizuálne definovať a upravovať ich rozloženie. Automatizáciu je možné na jednotlivé fázy prechodou alebo ako reakciu na zmeny hodnôt dát definovať pomocou akcii. Na komplexnú automaticky vykonávanu prácu je niekedy vhodnejšie a jednoduchšie použiť metódy v jazyku Groovy implementované vo vývojovom prostredí. Simuláciou siete je možné overiť, či vykonávanie procesu bude prebiehať očakávaným spôsobom a v tomto majú Petriho siete oproti notácii BPMN výhodu. Často je však vykonávanie procesu ovplyvňované akciami a tak to nemusí vždy stačiť. Proces je tak potrebné neskôr odkrokovať a odladiť v debugovacom režime vývojového prostredia.

Nasadenie novej verzie procesu

Pri zmenách štruktúry XML súboru siete, čo predstavuje zmeny pracovného toku, autorizačných rolí na prechodoch, dát a akcií, ktoré nepoužívajú metódy inej triedy je možné v bežiacej aplikácií nasadiť novú verziu procesu, ktorá bude automaticky použitá pri vytvorení novej inštancie daného procesu. Admin však musí znova manuálne jednotlivým užívateľom znova priradiť ich role.

Monitorovanie bežiacich inštancií

Na efektívne monitorovanie procesov a nájdenie chýb je potrebné dobre rozumieť spôsobu fungovania sietí jazyka Petriflow. Najlepšie je na monitorovanie použiť aplikáciu *MongoDB Compass*[26], ktorá dokáže zobraziť celú procesnú dátovú štruktúru bežiacich inštancií procesov. Tým, že nástroj ešte nie je vo vývoji tak dlho, nie všetko počas vývoja mojej aplikácie fungovalo tak ako by malo. Metódou pokus/omyl a pomocou tohoto montorovacieho nástroja sa však dalo s nátrojom naučiť efektívne pracovať a dosiahnuť tak požadovanú funkcionalitu.

[26] https://www.mongodb.com/products/compass

Kapitola 5

Zhodnotenie skúseností s vývojom

V tejto kapitole sú zhrnuté moje skúsenosti s vývojom procesnej aplikácie vo vybraných nástrojoch, ktorý je opísaný v predošlej kapitole. Pri každom nástroji je najskôr uvedené, či sa v vo vytvorenej aplikácii podarilo implementovať všetku požadovanú procesnú funkcionalitu špecifikovanú v kapitole 3.

Ďalej uvádzam silné a slabé stránky nástrojov a vzhľadom k nim vhodný spôsob ich použitia, resp. na implementáciu akých typov aplikácií je vhodné ich použiť. Ako silné stránky uvádzam tie vlastnosti a charakteristiky nástroja, ktoré vynikali svojím spôsobom implementácie jednotlivých aspektov vytvorenej procesnej aplikácie medzi porovnávanými riešeniami. Naopak slabé stránky nástroja predstavujú tie aspekty vývoja, v ktorých daný nástroj zaostával, prípadne aká funkcionalita natívne podporovaná vo zvyšných nástrojoch v ňom absentovala.

Camunda

Požadovanú funkcionalitu sa podarilo splniť. Výsledná aplikácia po otestovaní funguje podľa očakávaní.

Silné stránky

Vytvorenie diagramov procesov pomocou nástroja *Camunda Modeler* bolo z porovnávaných riešení najjednoduchšie. Modelovanie je intuitívne a rozhranie nástroja prehľadné. Vizuálny výsledok diagramov je tak z môjho pohľadu najlepší. Modeler umožňuje jednotlivé elementy diagramu prepájať s podnikovou vrstvou aplikácie, čo robí nástroj vhodný na použitie pri potrebe mať oddelené modelovanie a implementáciu aplikácie, alebo ako orchestračný prvok v mikroservisnej architektúre. Implementácia podnikových pravidiel vďaka intuitívnemu editoru rozhodovacích tabuliek v štandarde DMN (Desision Model and Notation) bola najjednoduchšia. Modeler umožňuje v procese na elementoch definovať hranice databázových transakcí, čo umožňuje regulovať spotrebu procesného výkonu a zlepšuje škálovateľnosť.

V ponuke nástroja je viacero webových aplikácií (Admin, Cockpit, Optimize) integrovateľných s procesným enginom pomocou jeho REST API, ktoré majú z porovnávaných nástrojov najviac zapracované možnosti monitorovania a analýzy procesov.

Treba spomenúť aj aktívnu komunitu na fóre nástroja, ktorá dosahuje skoro 8000 užívateľov.

Slabé stránky

Charakter nástroja je veľmi otvorený. V iných aspektoch vývoja samostatnej procesnej aplikácie tak zaostáva resp. v nich nepomáha a necháva implementáciu na používateľa. Nástroj síce ponúka základné používateľské rozhranie, ale jeho použitie je obmedzené a skôr zamerané na testovanie aplikácie. Formuláre nie je možné recyklovať, ani im pridávať dynamiku. Oproti porovnávaným nástrojom je definícia dát a práca s nimi zdĺhavá. Zoznam natívne podporovaných typov je taktiež najmenší.

Na základnú výpočtovú automatizáciu postačuje podpora skriptov v jazyku Groovy, no komplexnejšie operácie s množstvom medziprocesných závislostí si bude musieť vývojár implementovať samostatne v kóde aplikácie.

Vhodný spôsob použitia nástroja

Vzdialený prístup k procesnému alebo rozhodovaciemu enginu cez dobre zdokumentované REST API činí nástroj vhodným na orchestráciu viacerých služieb a ako komunikátor integrovaný do existujúcej architektúry. Ako doplnok k existujúcemu softwarovému riešeniu tak môže pristupovať k dátovým objektom podniku, serializovať ich a preposielať medzi jednotlivými službami.

Bonita

Splnenie požiadaviek špecifikovaného procesu

Z dôvodu rušenia procesu schvaľovania pri odosielaní udalostí do vzdialeného procesu, ktorým bol vytvorený, museli byť medziprocesné udalosti odstránené a diagram tomuto faktu prispôsobený. Určovanie rizika bolo zjednodušené a implementované pomocou skriptu z dôvodu absencie ukladania výsledku rozhodnutia založeného na podnikových pravidlách. Ich tvorba je podporovaná, ale použitie obmedzené len na určenie splnenia podmienky prechody.

Silné stránky

Z porovnávaných nástrojov bol tento najprívetivejší aj pre technicky menej zdatného užívateľa. V špecializovanom vývojovom prostredí *Bonita Studio* prebiehajú definície databázovej a organizačnej podnikovej štruktúry, konektorov, podmienok prechodu a mnoho iných pomocou špecifických editorov/sprievodcov, čo umožňuje vytvoriť všetky základné aspekty procesnej aplikácie pripravenej na nasadenie do produkcie aj neskúseným vývojárom v krátkom čase.

Práca s dátami, ktoré je v podobe podnikových dátových objektov možné zdielať a recyklovať medzi viacero procesmi je efektívna.

Dizajn formulárov je možné meniť pomocou nástroja *UI Designer* a uložené formuláre je možné priradiť viacerým úlohám. Frontendovým vývojárom tento nástroj umožňuje v krátkom čase vytvoriť jednak rôznorodé a dynamické formuláre, ale aj celé pohľady a navigáciu medzi nimi.

Uľahčené je aj programovanie skriptov, počas ktorého je k dispozícii strom s výberom príkazov databázového prístupu, ktoré umožňujú filtrovať a získavať dátové objekty podľa určítých kritérií.

Slabé stránky

Skúsenosť z vytvárania diagramu vo vývojovom prostredí bola kvôli slabej responzívnosti zdĺhavá. Pokiaľ bolo treba zmeniť definíciu dátového objektu aplikácie, bolo treba model znovu nasadiť, čo môže viesť k zmazaniu skriptov, ktoré k nim pristupujú alebo negatívnemu ovplyvneniu existujúcich inštancií procesov.

Co sa týka neplatenej verzie, ktorú som používal, chýbala podpora aplikačných profilov, ktoré by zamedzovali neoprávneným užívateľom pristupovať k administrátorskému pohľadu a debugovací režim programu v nástroji Bonita Studio.

Vhodný spôsob použitia nástroja

Nástroj by som použil v prípade procesnej aplikácie, ktorú netreba napojiť na existujúce užívateľské rozhranie. Táto aplikácia by nemala vyžadovať časté zásahy do dátovej štruktúry. Vďaka množstvu predpripravených konektorov na externé systémy a databázy si viem cieľovú aplikáciu predstaviť aj ako samostatný centrálny portál zoskupujúci dáta podniku.

Netgrif

Splnenie požiadaviek špecifikovaného procesu

Pravidlá určujúce výšku rizika boli implementované z dôvodu absencie natívnej podpory rozhodovacích tabuliek programaticky v kóde aplikácie. Špecifikovaná funkcionalita bola ale splená.

Silné stránky

Z dôvodu netradičného spôsobu procesného programovania a použitia petriho sietí na modelovanie procesov trvá v porovnaní so zvyšnými nástrojmi najdlhšie, kým si vývojár nástroj osvojí a bude ho efektívne používať. Z toho istého dôvodu ale nástroj ponúka najväčšiu flexibilitu a možnosti na implementáciu neštandardných procesov a pracovných postupov. Modelovací nástroj *NAB* (NetgrifApplicationBuilder) ako jediný natívne ponúka možnosť počas vytvárania procesu simulovať vykonávanie jeho prechodov, čím sa eliminuje riziko nedostupnosti častí procesu. Zoznam natívne podporovaných typov dát je z porovnávaných nástrojov najširší. Jednotlivé procesné premenné sa môžu použiť ako referencie na dáta úlohy hociktorého procesu aktívneho v aplikácií. Vďaka režimu editoru procesných premenných dáta umožňuje dopredu nadefinovať a tým automaticky inicializovať. S dátami je možné priamo manipulovať počas celého životného cyklu procesnej inštancie. Na to je vytvorené špeciálna programovacia syntax jazyka špeciálne vytvoreného pre účeli vývoja v tomto nástroji. Tento jazyk je určený aj na vytváranie akcii vykonávaných v rôznych fázach prechodou, alebo ako reakciu na zmenu hodnôt v procese, čo taktiež dodáva procesu dynamiku a najvyššiu prispôsobiteľnosť podnikovým požiadavkám z porovnávaných nástrojov. Pomocou akcii je možné pridať dynamiku aj formulárom, ktoré sa môžu nechať automaticky vygenerovať z dát úlohy a tak urýchliť vývoj užívateľského rozhrania.

Slabé stránky

Ako som už spomenul, efektívne si osvojiť nástroj trvá z porovnávaných riešení najdlhšie a mohlo by to začínajúceho vývojára odradiť[1]. Z dôvodu použitia Petriho sietí na vizualizáciu procesu nie sú ich diagramy na prvý pohľad tak transparentné ako vo zvyšných dvoch nástrojoch používajúcich notáciu BPMN 2.0, uspôsobenou na zrozumiteľnosť vytváraných procesných diagramov. Z dôvodu spojitého charakteru týchto sietí nie je možné intuitívne použiť viacero odlišných procesov ako súčasť jedného, hlavného procesu, ktorý by ich zastrešoval. Avšak z dôvodu flexibility a možností procesnej automatizácie je možné takéto správanie docieliť.

Užívateľské rozhranie funguje na technológii Angular a ponúka široké možnosti jeho úprav, no na druhej strane nemá predvolené rozhranie, ktoré by bolo možné rýchlo nakonfigurovať pre potreby jednoduchších aplikácii, ani nástroje na analýzu a monitorovanie bežiacich procesov, ktoré by boli od začiatku zahrnuté v riešení. Frontend tak vyžaduje osobitný vývoj skúsenými vývojármi, ktorý môže trvať dlhšie ako implementácia samotnej procesnej funkcionality.

[1]Momentálne nástroj nie je ponúkaný ako riešenie, v ktorom by si mohli vytvárať aplikácie zákazníci sami, no je to jeho ambícia a krivka náročnosti sa intenzívnym vývojom postupne znižuje.

Vhodný spôsob použitia nástroja

Výhody tohoto nástroja sa ukazujú pri potrebe vytvoriť komplexný systém, ktorý sa dokáže prispôsobiť aj netradičným pracovným postupom s množstvom úloh, ktoré prebiehajú paralelne a medzi ktorými existujú časté závislosti. Nástroj je možné použiť aj na vytvorenie jednoduchých aplikácií, ale jeho najväčší prínos vidím práve pri nahrádzaní komplikovaných systémov a architektúr s veľkým počtom zamestnancov so špecializovanými úlohami.

Kapitola 6

Záver

Cieľom tejto práce bolo porovnať a zhodnotiť spôsob návrhu a implementácie procesných aplikáci v troch nástrojoch s obdobným prístupom k ich vývoju. Na definovanie podnikových procesov nástroje používajú špecializované diagramy, ktorých základný princíp fungovania je opísaný v prvej časti práce. Tieto nástroje sú zložené z viacerých súčastí resp. nástrojov, ktoré sú spolu integrované a pokrývajú jednotlivé fázy vývoja alebo poskytujú dodatočnú funkcionalitu k vytvoreným aplikáciam. Ich ponuka a jednotlivé funkcie boli predstavené v druhej polovici teoretickej časti práce.

Následne bol vytvorený návrh demoštračnej aplikácie so špecifikáciou procesnej funkcionality, ktorú mala aplikácia implementovať. Najväčšia časť práce patrí opisu spôsobu a možností vývoja demonštračnej aplikácie v každom nástroji. Opis bol štruktúrovaný tak, aby jednotlivé časti odpovedali aspektom nevyhnutným na vytvorenie samostatne použiteľnej procesnej aplikácie. Na základe skúseností z vývoja sa mi podarilo identifikovať hlavné rozdiely prístupu vybraných nástrojov k vývoju jednotlivých aspektov procesných aplikácií. Tieto rozdiely boli zhrnuté v podobe silných a slabých stránok nástrojov v záverečnej časti práce. Na ich základe boli približené situácie, v ktorých by jednotlivé nástroje bolo vhodné použiť. Okrem toho, že mi táto práca poskytla skúsenosti a väčší rozhľad do problematiky vývoja procesných aplikácii si myslím, že môže byť použitá jednak ako referencia pre potenciálnych záujemcov o použitie jedného z vybraných nástrojov, ale aj ako prieskum potenciálnych smerov ďalšieho vývoja nástrojov pre spoločnosti, ktoré tieto nástroje ponúkajú.

Literatúra

[1] *Camunda Cockpit features* [online]. Navštívené: 2021-04-30. Dostupné z: https://camunda.com/products/camunda-platform/cockpit/.

[2] *Camunda Optimize features* [online]. Navštívené: 2021-04-30. Dostupné z: https://camunda.com/products/camunda-platform/optimize/.

[3] *Camunda platform introduction* [online]. [cit. 2021-01-29]. Navštívené: 2021-04-30. Dostupné z: https://docs.camunda.org/manual/7.15/introduction/.

[4] *Petri net (Wikipedia)* [online]. Navštívené: 2021-04-30. Dostupné z: https://en.wikipedia.org/wiki/Petri_net.

[5] *Petriflow language specification* [online]. Navštívené: 2021-04-30. Dostupné z: https://petriflow.com/.

[6] *What is BPMN?* [online]. [cit. 2021-02-05]. Navštívené: 2021-04-30. Dostupné z: https://www.bpmn.org/.

[7] Bonitasoft. *Bonita BPM portal interface overview.* [cit. 2021-02-28]. https://documentation.bonitasoft.com/bonita/2021.1/bonita-bpm-portal-interface-overview.

[8] Bonitasoft. *The ultimate Guide to BPMN 2.* [cit. 2021-02-06]. Navštívené: 2021-04-30. Dostupné z: https://www.bonitasoft.com/system/files/documentation_library/ultimate_guide_to_bpmn2_280116.pdf.

[9] Bonitasoft. *Define and deploy the Business Data Model (BDM)* [online]. 2021 [cit. 2021-04-25]. Navštívené: 2021-04-30. Dostupné z: https://documentation.bonitasoft.com/bonita/2021.1/define-and-deploy-the-bdm.

[10] Bonitasoft. *Overview of the Bonita solution architecture* [online]. 2021 [cit. 2021-04-10]. Navštívené: 2021-04-30. Dostupné z: https://documentation.bonitasoft.com/bonita/2021.1/what-is-bonita.

[11] Camunda. *Camunda Reference Architekture.* [cit. 2021-02-06]. Dostupné z: https://camunda.com/wp-content/uploads/2020/09/TB-Camunda_Reference_Architecture-092520.pdf.

[12] Deehan, N. *How CMMN never lived up to its potential* [online]. 2020 [cit. 2021-03-10]. Navštívené: 2021-04-30. Dostupné z: https://camunda.com/blog/2020/08/how-cmmn-never-lived-up-to-its-potential/.

[13] Desel, J. a Juhás, G. "What Is a Petri Net?" Informal Answers for the Informed Reader. In: *Unifying Petri Nets: Advances in Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 1–25 [cit. 2021-02-28]. DOI: 10.1007/3-540-45541-8_1. ISBN 978-3-540-45541-7.

[14] Freund, J. a Rücker, B. *Real-Life BPMN*. 4. vyd. Createspace, 2019 [cit. 2021-02-10]. ISBN 978-1086302097.

[15] Group, O. M. *Business Process Model and Notation (BPMN)*. [cit. 2021-02-06]. https://www.omg.org/spec/BPMN/2.0/PDF.

[16] Group, O. M. *Decision Model and Notation (DMN*. [cit. 2021-02-12]. https://www.omg.org/spec/DMN/1.3/PDF.

[17] Juhás, G., Mladoniczky, M., Mažári, J. a Gažo, T. *Petriflow: Rapid language for modelling Petri nets with roles and data fields*. Ilkovičova 3, 812 19 Bratislava, Slovakia: Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, 2020 [cit. 2021-04-02].

[18] Netgrif. *Process driven programming*. [cit. 2021-03-05]. https://netgrif.com/process-driven-programming/.

[19] Petri, C. A. a Reisig, W. Petri net. *Scholarpedia*. 2008, zv. 3, č. 4, s. 6477, [cit. 2021-03-26]. DOI: 10.4249/scholarpedia.6477. revision #91647.

[20] Riesz, M., Seckár, M. a Juhás, G. PetriFlow: A Petri Net Based Framework for Modelling and Control of Workflow Processes. In:. ACSD/Petri Nets Workshop, 2010 [cit. 2021-03-28]. ISSN 1613-0073.

Príloha A

Obsah priloženého pamäťového média

Pamäťové médium obsahuje tieto zložky:

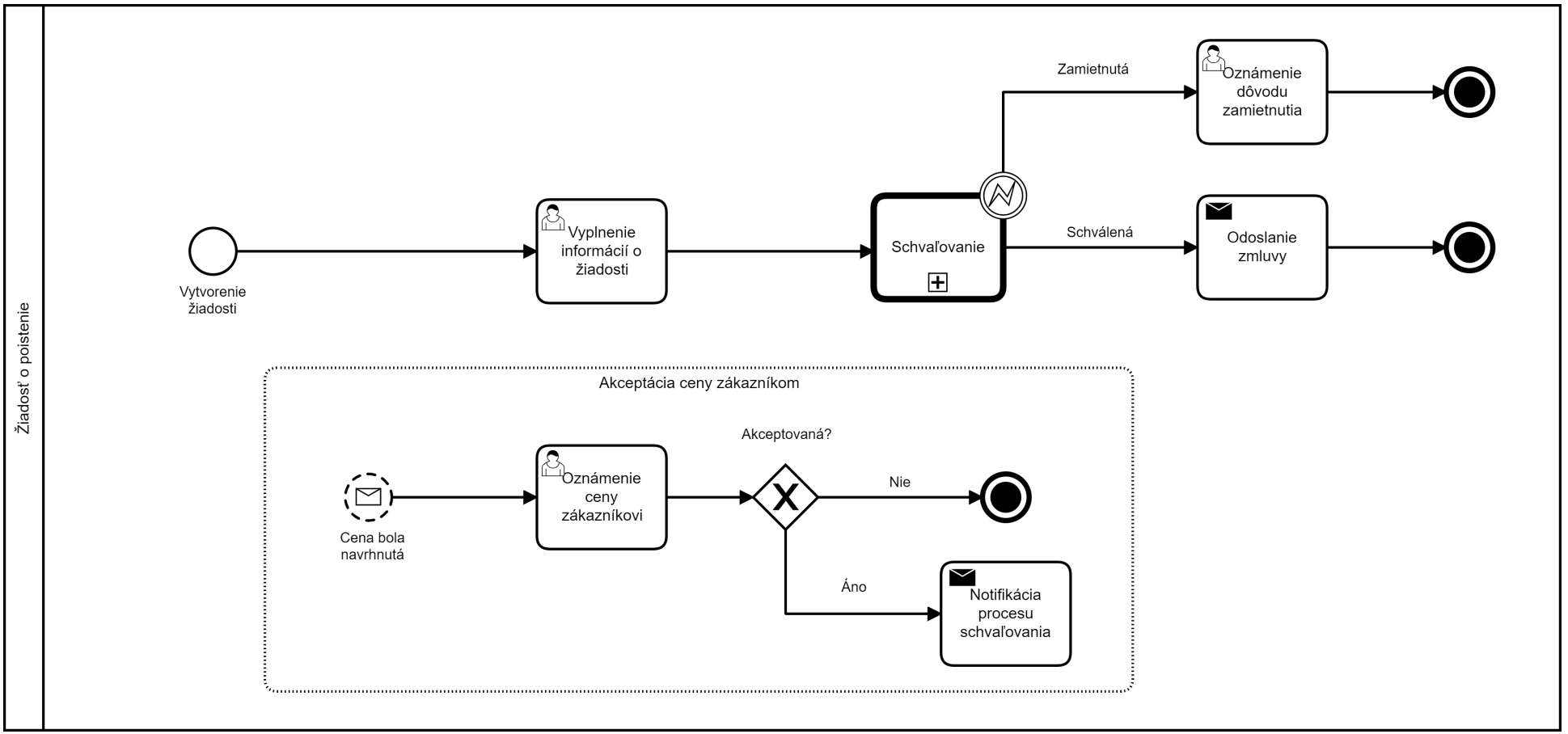
- Adresár *text* s technickou správou
- Adresár latex s kódom k vytvoreniu technickej správy
- Adresár *src* s podsložkami obsahujúcimi zdrojové kódy jednotlivých aplikácii
  - *camunda*
  - *bonita*
  - *netgrif*

Príloha B

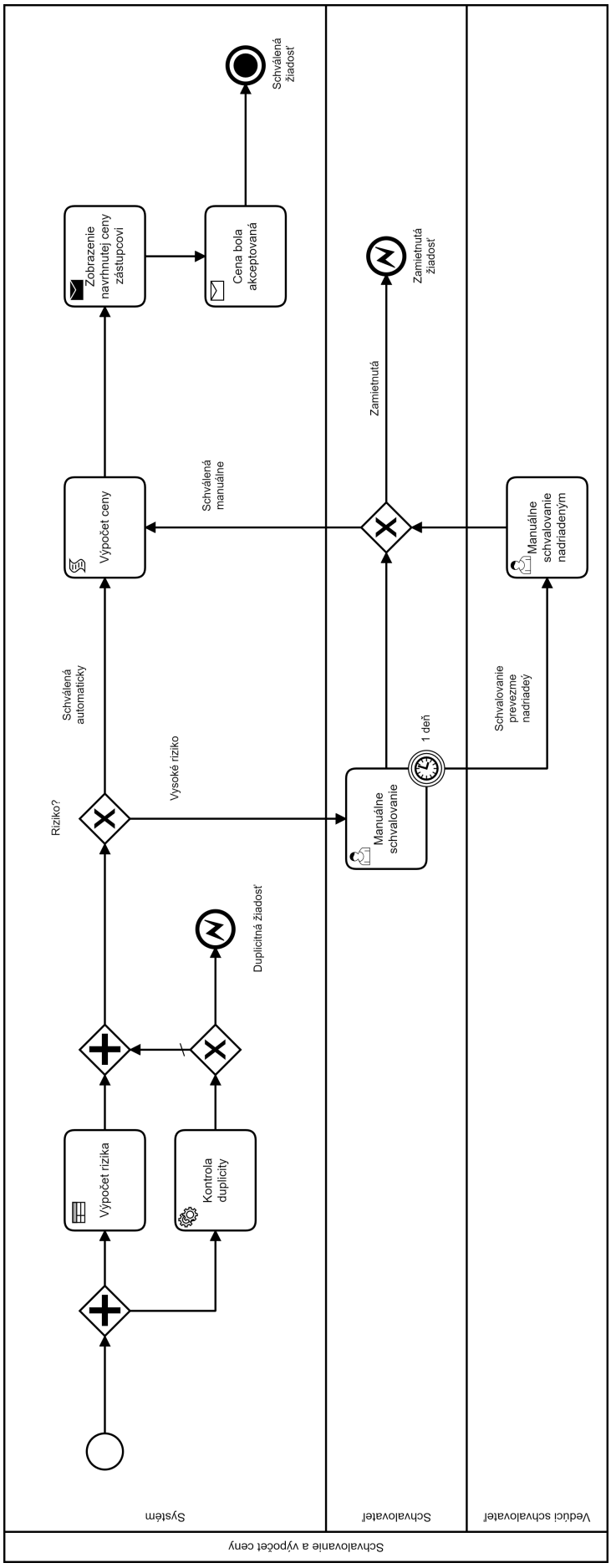
Diagramy vytvorených procesov aplikácie

Na ďalších stranách sú zobrazené diagramy procesu *vytvárania žiadosti* a procesu *schvaľovania* vytvorené pomocou modelovacích nástrojov porovnávaných platforiem. Poradie je nasledovné:

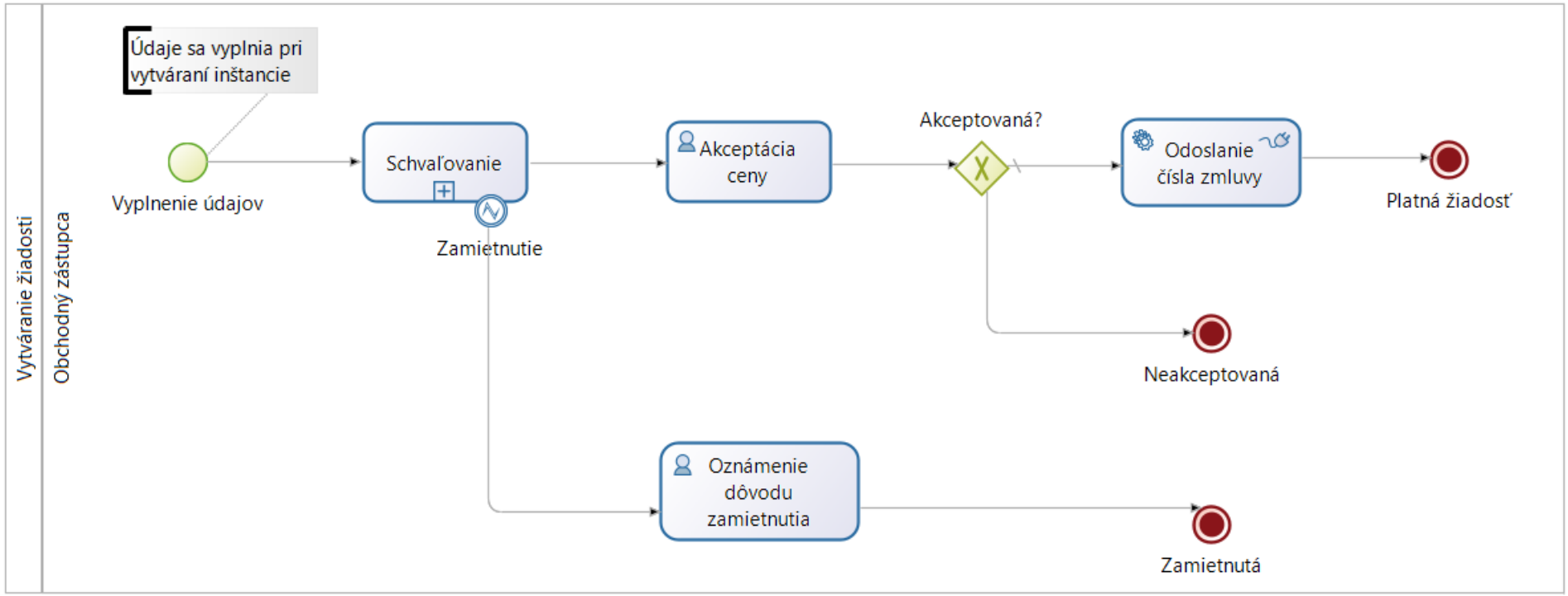
1. Diagramy procesov v notácii *BPMN2.0*, vytvorených v nástroji *Camunda Modeler*.
2. Diagramy procesov v notácii *BPMN2.0*, vytvorených vo vývojovom prostredí *Bonita Studio*.
3. Petriho siete procesov definované v jazyku Petriflow, vytvárané vo webovej aplikácii *Netgrif Application Builder*



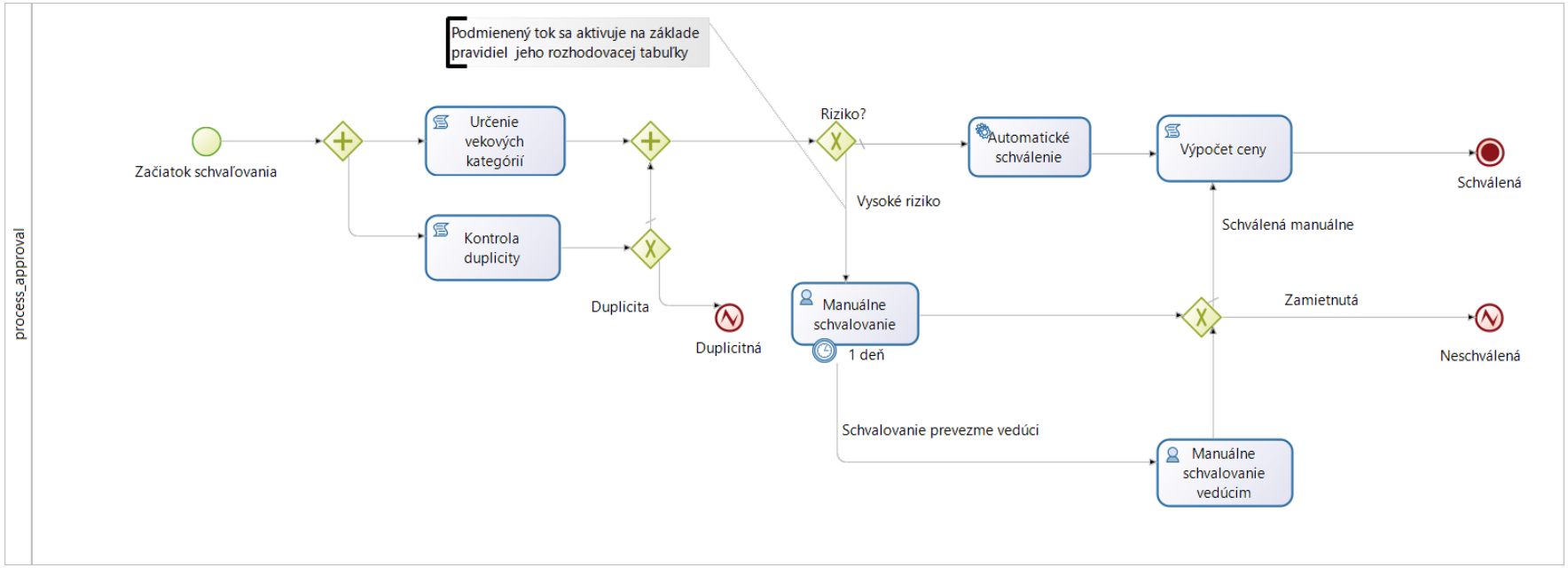
Obr. B.1: Diagram procesu vytvárania žiadosti (Camunda Modeler)



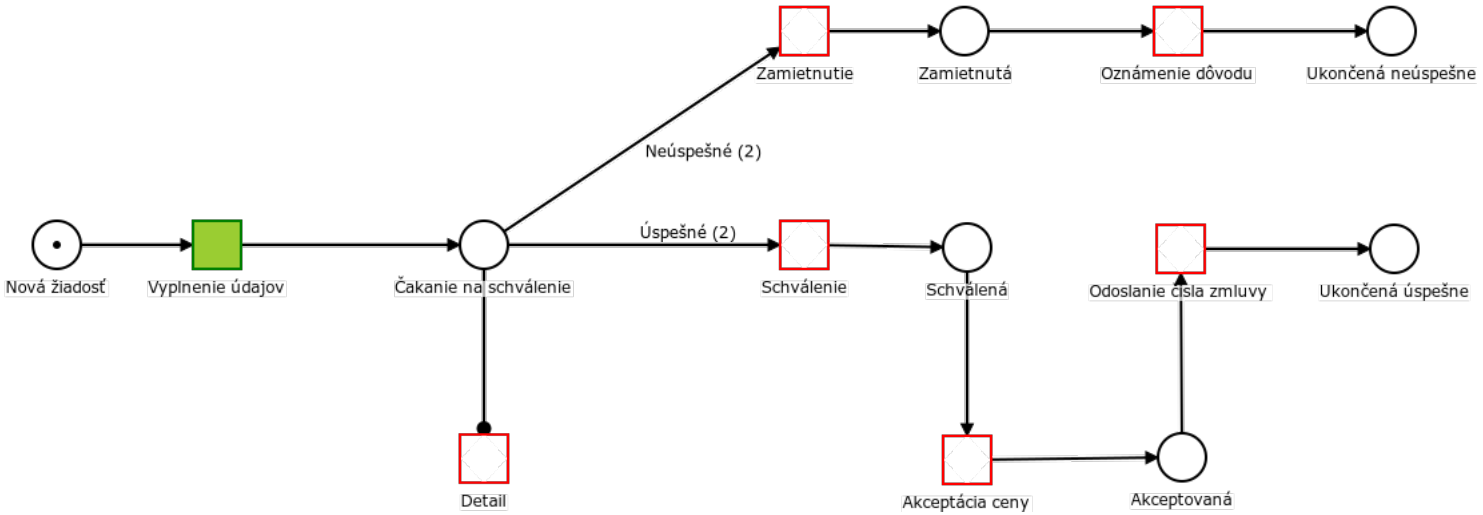
Obr. B.2: Diagram procesu schvaľovania (Camunda Modeler)



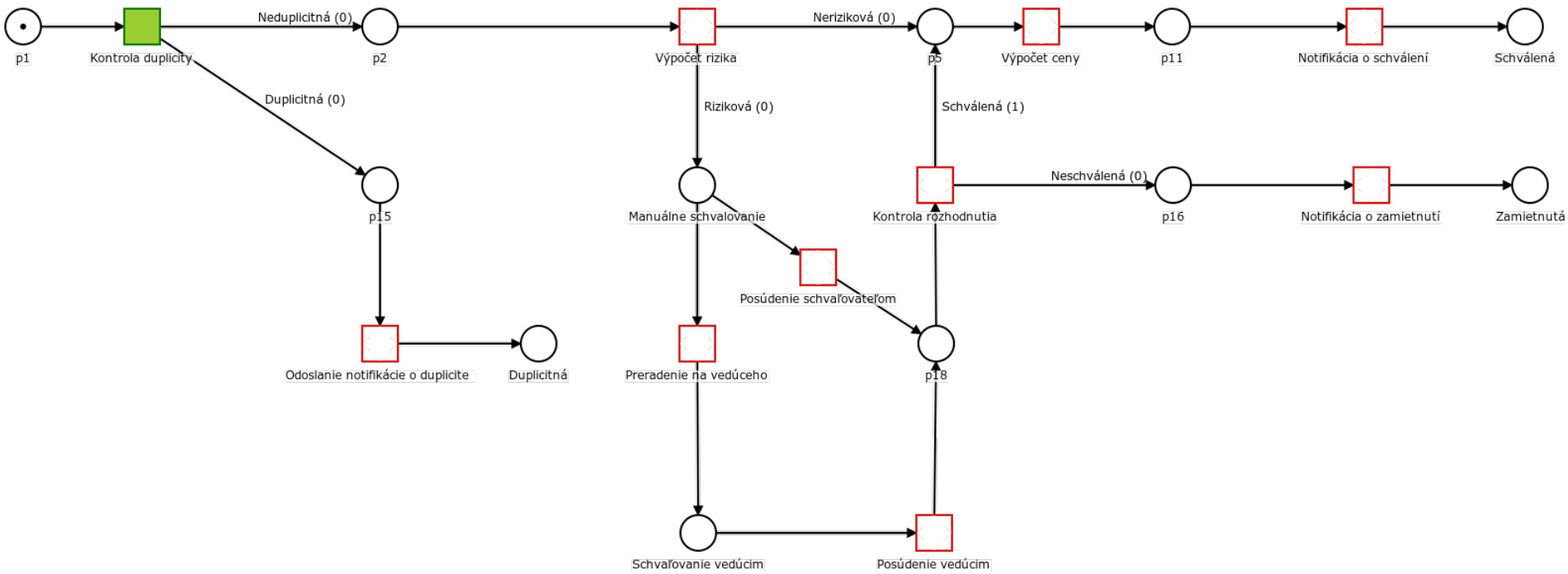
Obr. B.3: Diagram procesu vytvárania žiadosti (Bonita Studio)



Obr. B.4: Diagram procesu schvaľovania (Bonita Studio)



Obr. B.5: Sieť procesu vytvárania žiadosti (Netgrif Application Builder)



Obr. B.6: Sieť schvaľovania (Netgrif Application Builder)