VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZÍSKÁVÁNÍ ZNALOSTÍ Z PROCESNÍCH LOGŮ

KNOWLEDGE DISCOVERY FROM PROCESS LOGS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN KLUSKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Kluska Martin, Bc.**
Program: Informační technologie Obor: Bezpečnost informačních technologií
Název: **Získávání znalostí z procesních logů**
Knowledge Discovery from Process Logs
Kategorie: Data mining

Zadání:

1. Seznamte se s problematikou získávání znalostí se zaměřením na analýzu procesních logů.
2. Prostudujte existující nástroje pro analýzu procesů a po dohodě s vedoucím zvolte relevantní metody.
3. Navrhněte komponentu pro zpracování a analýzu procesních logů, která bude obsahovat zvolené algoritmy z oblasti dolování procesů.
4. Implementujte navržené řešení a proveďte experimenty se zvoleným vzorkem dat.
5. Zhodnoťte dosažené výsledky a diskutujte další možná rozšíření tohoto projektu.

Literatura:

- Han, J., Kamber, M.: Data Mining - Concepts and Techniques, 2nd Edition. Morgan Kaufmann Publishers, 2006.
- Van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer Verlag, 2016.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 23. října 2018

Abstrakt

Tato práce popisuje metody získávání znalostí o procesu využitím algoritmů z oblasti process miningu. V práci jsou detailně popsané vybrané algoritmy, jejichž cílem je vytvoření modelu procesu na základě analýzy logu událostí. Cílem je navrhnout komponenty, které budou umožňovat importování procesu a provádění následných simulací. Tyto výsledky poté mohou být použity pro krátkodobé plánování.

Abstract

This Master's describes knownledge discovery from process logs by using process mining algorithms. Chosen algorithms are described in detail. These aim to create process model based on event log analysis. The goal is to design such components, which would be able to import the process and run the simulations. Results from components can be used for short term planning.

Klíčová slova

Process Mining, Process Discovery, Simulation, α algoritmus, Heuristic minning

Keywords

Process Mining, Process Discovery, Simulation, α algoritmus, Heuristic minning

Citace

KLUSKA, Martin. *Získávání znalostí z procesních logů.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Získávání znalostí z procesních logů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Vladimír Bartíka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Kluska
22. května 2019

Poděkování

Rád bych poděkoval Ing. Vladimíru Bartíkovi, Ph.D. za cenné rady, osobní přístup a čas věnovaný vedení mé diplomové práce. Dále bych rád poděkoval panu Ing. et Ing. Vojtěchu Matesovi, Ph.D. za odbornou konzultaci.

Obsah

Úvod

Většina firem v dnešní době disponuje informačním systémem, která pomáhá s jejím řízením. Na základě jeho běhu vznikají data, která popisují chod podnikových procesů. Vzniká zde snaha tyto data dále využít pro získání informací důležitých pro optimalizaci chodu podniku. Vzhledem k velkému množství dat je ale téměř nemožné tato data analyzovat lidskou silou. Tento problém řeší oblast data science zvaná process mining.

V teoretické části této diplomové práce se nejdříve seznámíme se základními pojmy z process miningu. Ten nejdříve rozdělíme do jeho třech základních kategorií a u každé z nich definujeme její cíle. Seznámíme se s daty, která ke svému běhu využívají a dále se podíváme na konkrétní metody používané v oblasti process discovery.

Cílem této diplomové práce je implementace vhodných algoritmů, které dokáží z dostupného logu událostí vytvořit model procesu a zjistit informace o existující aktivitách, zdrojích a vztazích mezi nimi v rámci procesu. Na základě této analýzy zobrazíme statická data o době běhu aktivit a o požadované kapacitě zdrojů.

Výsledkem této práce je systém, který na základě zjištěných dat o procesu umožňuje provádění simulací. Jejich výsledky poté mohou pomáhat managementu firmy pomoci s procesem rozhodování při krátkodobém plánování.

Tato práce je rozdělena do sedmi kapitol. První kapitola zde slouží jako úvod do problematiky process miningu. Popíšeme si zde základní pojmy, se kterými budeme dále pracovat. Druhá kapitola se zaměřuje blíže na oblast process discovery, ve které si popíšeme principy vybraných algoritmů. Třetí kapitola popisuje existující komerční i akademické systémy, které implementují metody z process miningu. Ve čtvrté kapitole navrhneme systém, který bude implementovat zvolené algoritmy. Pátá kapitola podrobně popisuje implementační detaily. V šesté kapitole provedeme experimenty a zhodnotíme kvalitu implementovaných algoritmů. Zároveň zde srovnáme naše dosažené výsledky s vybranými komerčními nástroji. V závěrečné kapitole zhodnotíme dosažené výsledky a navrhneme další možná rozšíření.

Kapitola 1

Process mining

V této kapitole budeme seznámeni se základními pojmy z oblasti process miningu. Nejdříve si process mining rozdělíme do jeho základních kategorií a popíšeme si jejich cíle. Důležitou částí bude popis dat, která jsou v rámci process miningu využívána a nakonec se podíváme, jaké notace budeme používat pro znázornění modelu procesu.

1.1 Úvod

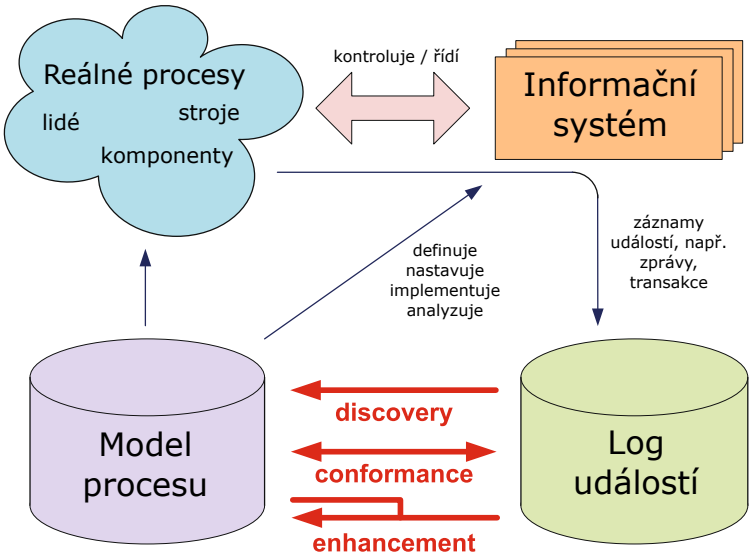
Většina firem využívá ke svému fungování nějaký informační systém, který pomáhá řídit její chod. Přesná funkce informačního systému je závislá na konkrétní oblasti podnikání, ale typicky se v těchto systémech setkáváme s informacemi o událostech, které vedou k vytváření výsledného produktu firmy. Tyto informace mohou být zobrazovány v různé úrovni agregace od detailů jedné konkrétní objednávky až po souhrnné informace o počtu objednávek za určité období.

Process mining si klade za cíl analyzovat data získaná v průběhu práce s informačním systémem. V rámci analýzy je cílem získat konkrétní údaje o podnikových procesech, které se zde odehrávají. Nejčastějším výstupem těchto metod jsou grafické modely, které znázorňují posloupnost a vztahy mezi aktivitami, které jsou v rámci procesu vykonávány. Metody process miningu se ale neomezují pouze na modely procesu. Můžeme se setkat i s metodami, jejichž cílem je například sestrojení statistik o době běhu procesu a co ji nejvíce ovlivňuje.

Získaná data mohou být poté vhodným způsobem zobrazena a dále analyzována. To může vést například k odhalení kritických míst procesů, na která by se měla firma v budoucnu zaměřit a tím zlepšit svoji efektivitu. Dalším možným využitím těchto dat je snazší vysvětlení chodu firmy třetí osobě. To je například výhodné ve chvíli, kdy přijmeme nového zaměstnance.

Jako příklad si můžeme vzít typický internetový obchod, kde je hlavním procesem zpracování objednávky. Jako proces zde chápeme posloupnost událostí, která začíná navštívením internetového obchodu zákazníkem a končí doručením zboží. Mezi těmito událostmi se může nacházet velké množství dalších událostí od vložení zboží do košíku, zaplacení za objednávku až po doručení zboží.

Analýzou dat z internetového obchodu můžeme například zjistit, že největší zdržení v celém procesu způsobuje doba vyskladnění produktů. Pokud bychom chtěli zákazníkům doručit zboží v co nejkratším čase, měli bychom se zaměřit právě na tuto aktivitu.



Obrázek 1.1: Přehled process miningu a jeho tří základních typů: discovery (vytváření modelu), conformance (validace modelu) a enhancement (vylepšování modelu) [5].

Na obrázku 1.1 je znázorněn vztah mezi podnikovým procesem, informačním systémem, logem událostí a modelem procesu.

- **Reálné procesy**. Touto částí rozumíme veškeré procesy a zdroje, které vedou k vytvoření reálného produktu.
- **Informační systém**. Je část podniku, která má za cíl řídit, či kontrolovat chod reálných procesů.
- **Log událostí**. Informační systém při svém běhu vytváří data o činnostech, které byly v rámci reálného procesu vykonány. V rámci těchto dat nalezneme například informace o tom, kdy byla spuštěna daná úloha, jaké zdroje k ní byly využity a nebo celkovou dobu vykonávání úlohy.
- **Model procesu**. Popisuje chování reálného procesu. Definuje vztahy mezi jednotlivými aktivitami, popisuje vztahy mezi aktivitami a zdroji a mnoho dalších. Na jeho základě můžeme analyzovat a zlepšovat reálný proces.

1.2 Cíle process miningu

Ve spodní části obrázku 1.1 je znázorněno několik různých vztahů mezi logem událostí a modelem procesu. Pro všechny typy vztahů platí, že ke svému běhu využívají log událostí. Liší se v přístupu k modelu procesu. V následujících kapitolách si tyto typy podrobně popíšeme.

1.2.1 Process discovery

Process mining je nejčastěji spojován právě s touto kategorií. Jejím cílem je získání znalostí o procesu bez žádných předchozích znalostí. K tomu využívá informace o událostech

v procesu, které jsou zaznamenané v logu událostí. Na rozdíl od ostatních kategorií je log událostí jediným vstupem, který tyto algoritmy využívají.

Nejčastěji se setkáme s metodami, jejichž cílem je sestavení odpovídajícího modelu procesu. Process discovery se ovšem nezaměřuje pouze na sestavení modelu procesu, ale obsahuje i metody, jejichž cílem je například získání organizačního pohledu, nebo odhalení úzkého místa celého procesu.

Pro vytváření modelu procesu byla vyvinuta celá řada různých algoritmů. Jedním z prvních známých algoritmů je α algoritmus, který je založen na vyhledávání relací mezi událostmi v logu událostí a bude dále popsán v sekci 2.3. Dále se můžeme setkat například s Heuristic mining algoritmem, který cílí na logy událostí, obsahující větší množství šumu. Dále se můžeme setkat Genetic Minerem [8], který využívá evoluční algoritmy.

1.2.2 Conformance checking

Další kategorie process miningu, *conformance checking* [16], ke svému běhu využívá již existující model procesu. Cílem metod z této kategorie je tento model validovat. Existující model procesu mohl vzniknout například z logu událostí, který neobsahoval všechny možné posloupnosti vykonávání úloh.

Dalším vstupem je log událostí, který vznikl během reálného procesu. Metody poté zkouší přehrát události zaznamenané v logu událostí v modelu procesu. Cílem je odhalení nesrovnalostí mezi modelem a zaznamenanou realitou. Konkrétně se zaměřujeme na:

- Chování popsané v logu, které není dle modelu možné.
- Chování, které je podle modelu možné, ale nikdy se nevyskytuje v logu událostí.

Jako příklad si můžeme vzít systém, který řídí vývoj software. V modelu procesu máme definováno, že každá změna v kódu má projít kontrolou kódu někým jiným, než je autor změn. Pomocí conformance checkingu můžeme zjistit, zda se toto pravidlo skutečně dodržuje.

1.2.3 Enhancement

Poslední kategorie process miningu má za cíl vylepšit existující model procesu na základě znalostí získaných z logu událostí. Na rozdíl od conformance checkingu je hlavním cílem této kategorie změna nebo rozšíření modelu procesu [5].

Jedním z typů vylepšení je oprava, tedy modifikování modelu, aby lépe popisoval skutečné procesy. Například se může stát, že jsou dvě úlohy modelovány sekvenčně, zatímco ve skutečnosti se mohou provádět souběžně. Tuto skutečnost můžeme odhalit analýzou času spuštění úloh pro jednotlivé instance.

1.3 Modely procesů

V této sekci budou formálně popsány notace, které se nejčastěji používají pro popis modelu procesu. Zaměříme se především na Petriho sítě a z nich odvozené Workflow sítě.

1.3.1 Petriho sítě

Petriho sítě jsou jednou z nejčastěji používaných notací pro modelování procesu [9]. Díky matematickému základu a podpoře modelování souběžných aktivit jsou vhodné pro modelování průběhu procesu. Další výhodou použití Petriho sítí je intuitivní grafické zobrazení

a dostupnost velké řady nástrojů, které s nimi umí pracovat. Následující definice 1.1 a 1.2 jsou převzaty ze zdroje [20].

Definice 1.1 *Petriho síť je trojice $N = (P, T, F)$, pro kterou platí:*

- P je konečná množina míst (*places*)
- T je konečná množina přechodů (*transitions*), kde $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ je binární relací, kterou nazveme tokovou relace sítě (*flow relation*)

Na obrázku 1.2 je ukázka jednoduché Petriho sítě. Grafem sítě nazýváme grafovou reprezentaci trojice $N = (P, T, F)$. Jedná se o orientovaný bipartitní graf, jehož uzly reprezentují místa (places) a přechody (transitions). Tokova relace představuje orientované hrany, které propojují místo s přechodem či naopak.

Obrázek 1.2: Jednoduchá Petriho síť.

Ještě než si definujeme samotnou Workflow síť, je vhodné si definovat množiny preset a postset.

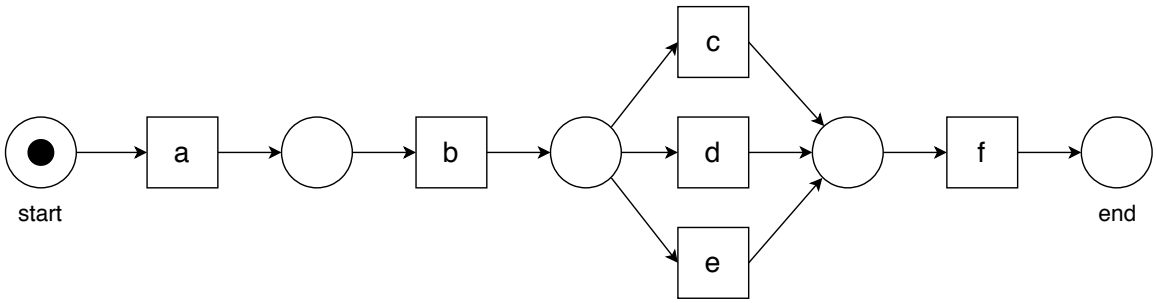
Definice 1.2 (Preset, postset) $\forall x \in (P \cup T)$*:*

- $\bullet x = \{y \mid yFx\}$ *jako preset* x
- $x\bullet = \{y \mid xFy\}$ *jako postset* x

1.3.2 Workflow sítě

Jedná se o speciální typ Petriho sítě [17]. Workflow sítě, na rozdíl od Petriho sítě, nahlíží na každé spuštění procesu jako na izolovanou událost [6]. Každá instance má definovaný její začátek a její konec.

Model procesu definuje pouze posloupnost vykonávání jednotlivých úloh. Každá instance procesu funguje izolovaně.



Obrázek 1.3: Ukázka Workflow sítě.

Definice 1.3 *Petriho síť $N = (P, T, F)$ je Workflow síť, pokud platí následující:*

- *Existuje právě jedno startovací místo start, takové že* $\{p \in P \mid \bullet p = \emptyset\} = \{i\}$
- *Existuje právě jedno koncové místo o, takové že* $\{p \in P \mid p\bullet = \emptyset\} = \{o\}$
- *Každý uzel se musí nacházet na alespoň jedné cestě vedoucí z uzlu i do uzlu o*

Definice Workflow sítě 1.3 je převzata ze zdroje [4].

1.4 Log událostí

Logy událostí používané pro process mining mají takovou strukturu, která umožňuje vhodnou reprezentaci spuštěných událostí spolu s jejich atributy. Log událostí může obsahovat informace o událostech spuštěných v rámci několika průběhů procesu. Algoritmy logy analyzují a na jejich základě sestaví model procesu (process discovery), nebo kontrolují do jaké míry model procesu odpovídá skutečnému procesu (conformance checking), případně se zaměřují na budoucí vylepšení modelu (enhancement).

Výsledky jednotlivých algoritmů jsou závislé na kvalitě logu událostí. V této práci budou dále popsány nejčastější problémy a jejich řešení týkající se dat v logu událostí.

Ne každý log událostí je vhodný pro process discovery. Vhodný log událostí obsahuje pouze informace týkající se pouze toho procesu, se kterým chceme pracovat.

Událost	**Případ**	**Čas spuštění**	**Úloha**	**Zdroj**
2049	11	13. 12. 2018 11:32:06	Podání žádosti	John
2050	11	13. 12. 2018 12:14:36	Přijetí žádosti	Alice
2051	11	13. 12. 2018 14:58:59	Základní oprava	Charles
2052	11	13. 12. 2018 15:00:11	Vrácení zboží	Alice
2053	12	13. 12. 2018 15:36:32	Podání žádosti	Bob
2054	12	13. 12. 2018 15:51:10	Přijetí žádosti	Alice
2055	12	13. 12. 2018 15:55:01	Pokročilá oprava	Charles
2056	12	13. 12. 2018 17:12:07	Vrácení zboží	Alice
2057	13	13. 12. 2018 20:45:12	Podání žádosti	Pete
2058	13	15. 12. 2018 10:43:27	Přijetí žádosti	Alice
2059	13	13. 12. 2018 20:45:12	Zamítnutí žádosti	Charles
2060	13	15. 12. 2018 10:43:27	Vrácení zboží	Alice

Tabulka 1.1: Ukázka událostí v informačního systému, který umožňuje ukládat záznamy o provedených reklamacích. Tabulka obsahuje záznamy o třech instancích procesu, přičemž má každý jiný průběh. Každá událost obsahuje informace o tom, kdy byla spuštěna, kdo ji spustil a do jaké skupiny patří uživatel, který ji spustil.

V tabulce 1.1 je ukázka událostí, které vznikaly v průběhu zpracování reklamace podle modelu z obrázku 1.3. Jsou zde zobrazeny záznamy z průběhu tří různých instancí stejného procesu. Každá událost má svůj unikátní identifikátor, číslo případu, ke kterému se vztahuje. Dále jsou zde informace o tom, kdy tato událost nastala a o jakou činnost se jednalo. V posledním sloupci je uveden zdroj, jaký byl pro vykonání události využit. K sestavení odpovídajícího modelu procesu potřebujeme mít v logu uložený alespoň jeden průběh pro každou možnost sledu událostí.

Na základě ukázky vidíme, že:

- proces je složen z průběhů (*case*),
- průběh je složen z událostí, kdy každá událost náleží právě k jednomu průběhu,
- události jsou v rámci průběhu seřazeny dle data spuštění,
- události mohou mít další atributy (typickými atributy jsou čas běhu, využité zdroje nebo cena spuštění aktivity).

1.4.1 Data

Log událostí, který budeme využívat pro získávání informací o procesu, musí splňovat určité podmínky. Základním požadavkem je schopnost rozlišit, ke které instanci procesu (sloupec „Případ“) se událost vztahuje. Dále musíme být schopni vyčíst identifikaci úlohy, která byla vykonána (sloupec „Úloha“) a musíme být schopni nějakým způsobem seřadit úlohy v rámci jedné instance. K tomu může sloužit například čas (sloupec „Čas spuštění“), kdy byl daný záznam vložen do logu událostí.

Kromě výše uvedených základních informací může obsahovat údaje, které jsou specifické pro konkrétní procesy nebo úlohy. V tabulce 1.1 máme například uložené informace o zdroji (sloupec „Zdroj“), který byl v rámci úlohy využit. Kromě údajů uvedených v tabulce zde dále mohou být informace o nákladech spojených s vykonáním události nebo o době vykonávání události.

Jako příklad si můžeme vzít proces reklamace. Jeden z klíčových atributů zde bude produkt, který je reklamován. Dalším z důležitých atributů v logu událostí může být například informace o zjištěné závadě produktu, která bude mít pravděpodobně velkou korelaci s následujícími spuštěnými aktivitami (u produktu s nižší hodnotou je vyšší pravděpodobnost výměny za nový kus, zatímco u produktu s vyšší hodnotou je vyšší pravděpodobnost provedení opravy).

Ve fázi sestavování modelu systému nás konkrétní data nebudou vůbec zajímat, ale budeme je využívat ve fázi plánování. V této fázi můžeme využít nějaký zjištěný vztah mezi datovým atributem a chováním systému [19].

1.4.2 Vytvoření posloupnosti událostí

Log událostí můžeme chápat jako posloupnost událostí, které nastaly v několika instancích zkoumaného procesu. Pro zjišťování informací o chování procesu potřebujeme na jednotlivé instance nahlížet odděleně. Prvním krokem tedy bude posloupnost událostí rozdělit na několik menších posloupností, ve kterých budou pouze události, které se vztahují ke stejné instanci procesu.

Každá událost má u sebe mimo jiné údaje o datu jejího vykonání a identifikaci instance, ke které se vztahuje.

Pro další zpracování logu budeme transformovat log událostí do podoby množiny posloupností průběhů jednotlivých instancí. Tato množina bude mít stejný počet prvků, jako je počet zaznamenaných instancí procesu v logu událostí. Každá posloupnost tedy odpovídá jedné instanci.

Posloupnost obsahuje aktivity spuštěné v rámci jedné instance. U těchto aktivit nás nezajímá přesný čas spuštění události, ale tento údaj využijeme k seřazení úloh v rámci jedné instance.

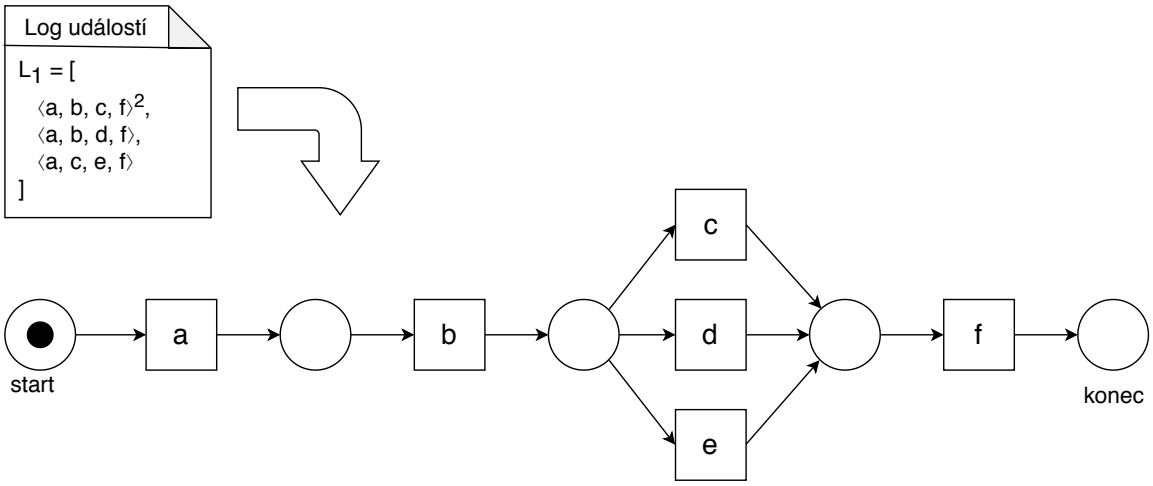
V této diplomové práci budeme log událostí zapisovat následujícím způsobem:

$$L_1 = [\langle a, b, c, f\rangle, \langle a, b, d, f\rangle, \langle a, b, e, f\rangle]$$

Tento log událostí vznikl spuštěním tří instancí zkoumaného procesu, přičemž každá instance měla jiný průběh. Často se setkáme s tím, že bude v logu událostí více instancí procesů se stejným průběhem. V takovém případě zapisujeme pouze unikátní průběhy, které mají v horním indexu označen počet výskytů instancí procesu se stejným průběhem v logu událostí, přičemž implicitní hodnotu 1 nemusíme uvádět.

$$L_1 = [\langle a, b, c, f\rangle^2, \langle a, b, d, f\rangle, \langle a, b, e, f\rangle]$$

Log událostí L_1 vznikl během čtyř instancí procesu, přičemž průběh $\langle a, b, c, f\rangle$ se zde vyskytuje dvakrát.



Obrázek 1.4: Vstupem pro process discovery je log událostí L_1, který obsahuje průběhy jednotlivých instancí. Každý průběh popisuje, jaké úlohy byly spuštěny v jakém pořadí. Na základě vygenerovaného modelu bychom měli být schopni zpětně reprodukovat jednotlivé průchody.

Obrázek 1.4 zobrazuje vztah mezi logem událostí a modelem procesu. Log událostí obsahuje posloupnosti průběhů jednotlivých instancí. Na log událostí použijeme některý z algoritmů process miningu, který jej transformuje do modelu popisujícího zachycené chování.

1.4.3 Šum v datech

V kontextu logu událostí nechápeme šum jako nesprávné zaznamenání hodnot. Rozumíme tím, že log událostí obsahuje taková data, která jsou v rámci procesu výjimečná, či málo frekventovaná. Algoritmy z kategorie process discovery se liší přístupem k šumu v datech.

Některé algoritmy (např. α algoritmus) předpokládají, že log událostí neobsahuje žádný šum. Případný šum v datech by měl být odstraněn ve fázi předzpracování dat.

Byly ovšem vyvinuty i takové algoritmy (např. Heuristic miner), které s existencí šumu počítají a zvládají případný šum samy odstranit.

Jako příklad si můžeme vzít log událostí $L_1 = [\langle a, b, c\rangle^{987}, \langle a, b, d\rangle^{741}, \langle b, a, c\rangle]$. Lze vidět, že událost b je v 1728 případech spuštěna hned po události a a jenom v jednom případě je tomu tak naopak. Jinými slovy v 99,94 % případů, bylo b spuštěno po a, opačný případ s pravděpodobností 0,06 % tedy můžeme chápat, jako šum.

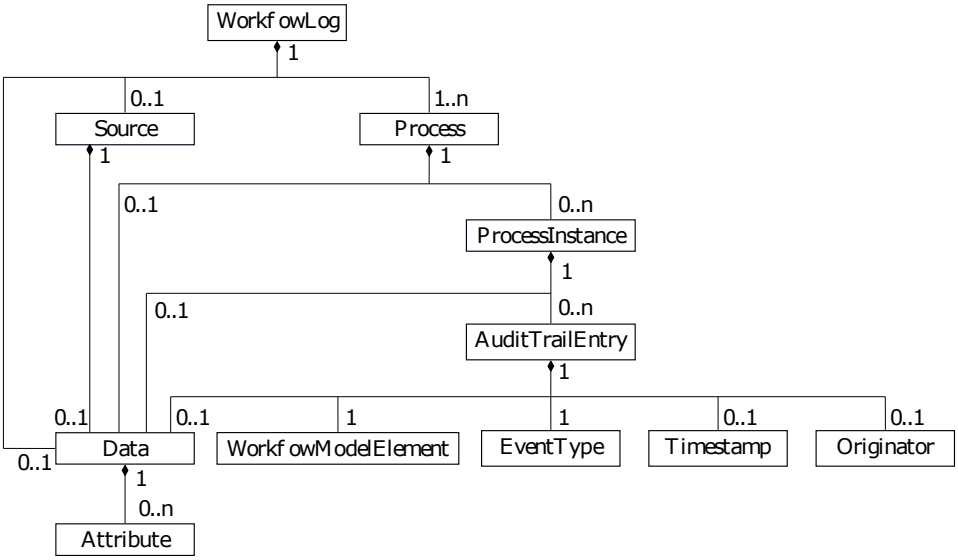
1.4.4 Nekompletnost

Zatímco šum v datech obsahuje takové průběhy, které jsou pro výsledný proces nerelevantní, v případě nekompletnosti nám relevantní data chybí.

Jako příklad si opět můžeme vzít $L_1 = [\langle a, b, c\rangle^{987}, \langle a, b, d\rangle^{741}, \langle b, a, c\rangle]$. Zde máme zaznamenané tři různé průběhy, ale reálný model může povolovat ještě průběh $\langle a, b, c, d\rangle$. Tento průběh se však po dobu zaznamenávání neobjevil v logu událostí.

1.4.5 MXML

Události vznikající v informačních systémech mohou mít různou podobu. Často se setkáváme s ukládáním hodnot do databáze, ale mohou být uloženy například i v obyčejném textovém souboru. Pro možnost zpracování těchto dat vznikla snaha vytvořit jednotný formát. První takovou snahu představil Van Dongen a Van der Aalst ve své publikaci A Meta Model for Process Mining Data [12].



Obrázek 1.5: UML schéma formátu MXML [14].

Na obrázku 1.5 je znázorněno UML schéma standardu MXML. Log událostí je zde reprezentován elementem `WorkflowLog`, který obsahuje jeden či více procesů (element `Process`). Log událostí může dále v elementu `Source` volitelně obsahovat informace o svém zdroji. Každý `Process` dále obsahuje libovolný počet `ProcessInstance`, který koresponduje s instancí spuštěného procesu, kde každou událost reprezentuje element `AuditTrailEntry`.

```
<WorkflovLog>
    <Source program="X-ray machine"/>
    <Process id="FieldServiceFSCommands">
        <ProcessInstance id="3439242_2007-04-03_2007-04-03_1">
            <AuditTrailEntry>
                <Data>
                    <Attribute name="Main Menu">Field Service Windov</Attribute>
                    <Attribute name="Procedure">Service Login</Attribute>
                </Data>
                <WorkflovModelElement>Login</WorkflovModelElement>
                <EventType>complete</EventType>
```

```
12                <Timestamp>2007-04-03T15:48:09.000+01:00</Timestamp>
13            </AuditTrailEntry>
14            <AuditTrailEntry>
15                <Data>
16                    <Attribute name="Group">SelectFluoPlavour</Attribute>
17                    <Attribute name="Main Menu">Acquisition Presetting</Attribute>
18                </Data>
19                <WorkflovModelElement>SelectFluoFlavour</WorkflovModelElement>
20                <EventType>complete</EventType>
21                <Timestamp>2007-04-03T15:48:13.000+01:00</Timestamp>
22            </AuditTrailEntry>
23            ..
24        </ProcessInstance>
25    </Process>
26 </WorkflovLog>
```

Výpis 1.1: Ukázka části souboru obsahující informace o procesu [3]

V ukázce 1.1 je znázorněna část logu událostí ve formátu MXML z procesu vytváření rentgenových snímků. Tento log obsahuje informace o jednom průběhu procesu s názvem `FieldServiceFSCommands` v jehož průběhu bylo vykonáno několik aktivit.

Jedním z problémů tohoto standardu je neexistující podpora pro dodatečná rozšíření. V praxi se začaly využívat elementy typu `Data`, pro ukládání dodatečných informací. Nicméně kvůli neexistujícímu standardu nebývala tato data dále zpracována.

1.4.6 XES

Formát XES (eXtensible Event Stream) vznikl jako reakce na nedostatky formátu MXML [2]. V roce 2010 byl přijat pracovní skupinou IEEE jako standard pro ukládání logu událostí. Tento standard není přímo svázán s konkrétní syntaxí, nejčastěji se ale setkáme s podobou serializace do XML souboru.

Zatímco formát MXML umožňoval ukládání informací o více procesech v rámci jednoho souboru, formát XES poskytuje podporu pouze pro jeden proces v rámci jednoho logu událostí.

Meta model formátu XES definuje základní datové typy atributů. Konkrétně se jedná o datové typy: `String`, `Date`, `Int`, `Float`, `Boolean` a `ID`, které korespondují s vestavěnými datovými typy formátu XML `xs:string`, `xs:date`, `xs:int`, `xs:float`, `xs:boolean` [3].

```
1 <string key="name" value="Tom" />
2 <date key="startAt" value="2009-11-25T19:45:32.345+02:00" />
3 <int key="counter" value="236366" />
4 <float key="percentage" value="75.68" />
5 <boolean key="success" value="true" />
6 <id key="customer" value="f81d4fae-7dec-11d0-a765-00a0c91e6bf6" />
```

Výpis 1.2: Základní datové typy podporované standardem XES.

Kromě základních typů umožňuje vytvářet i složené typy:

- **Seznam** (List). Seznam může obsahovat libovolný počet vnořených elementů. Vnořené elementy jsou seřazené a mohou mít shodný identifikátor klíče (`key`). Hodnota seznamu je odvozena z hodnot vnořených elementů.

```
1 <list key="revisions">
2     <string key="name" value="XES standard" />
```

```
3       <boolean key="stable" value="true" />
4       <string key="revision" value="2.0" />
5       <string key="revision" value="1.2" />
6       <string key="revision" value="1.1" />
7       <string key="revision" value="1.0" />
8   </list>
```

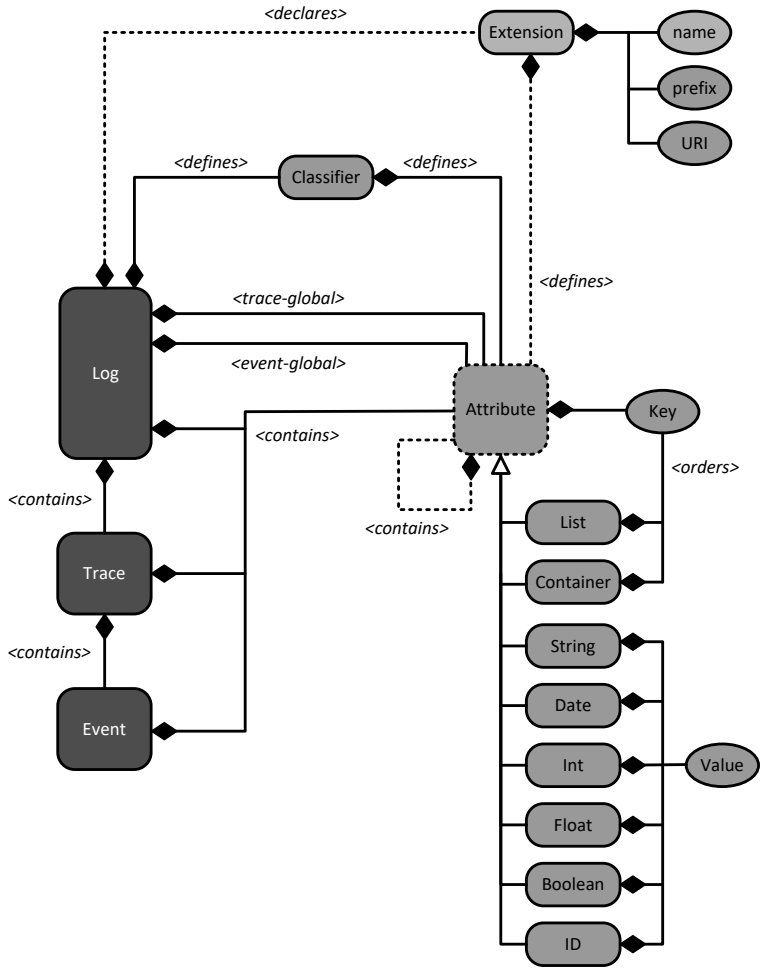
Výpis 1.3: Základní datové typy podporované standardem XES.

- **Kontejner** (Container). Kontejner může obsahovat libovolný počet vnořených elementů. Tyto elementy nejsou seřazené. Hodnota kontejneru je odvozena z hodnot vnořených elementů.

```
1   <container key="location">
2       <string key="street" value="Božetěchova" />
3       <int key="number" value="2" />
4       <string key="zip" value="612 00" />
5       <string key="city" value="Brno" />
6       <string key="country" value="Czech republic" />
7   </container>
```

Výpis 1.4: Datové typy kontejner podporované standardem XES.

MXML umožňuje v rámci jednoho logu událostí ukládat události z více procesů, zatímco formát XES povoluje ukládat informace pouze o jednom procesu.



Obrázek 1.6: XES meta model [2]. Každý záznam (*Log*) obsahuje kolekci stop (*Trace*) a každá stopa obsahuje kolekci událostí (*Event*). V rámci záznamu může být přidán libovolný počet rozšíření (*Extension*), které přidává nové atributy.

Obrázek 1.6 zobrazuje meta model XES standardu pomocí UML diagramu. Vyskytují se v něm objekty několika typů:

- **Záznam** (Log). Na nejvyšší úrovni dokumentu je definován právě jeden objekt typu `Log`, který obsahuje kolekci stop (`Trace`), definuje dostupná rozšíření (`Extension`).
- **Běh** (Trace). Obsahuje informace o jedné konkrétní instanci procesu.
- **Událost** (Event). Je objekt reprezentující konkrétní událost, která nastala v rámci procesu.
- **Klasifikátor** (Classifier). Žádný z výše uvedených objektů nemá dle standardu definované povinné atributy. Klasifikátory umožňují vytvoření povinných atributů v rámci dokumentu.
- **Rozšíření** (Extension). Definuje, jaká rozšíření XES standardu budou v logu používána. Nachází se zde informace o názvu rozšíření, prefixu používaném v tomto dokumentu a URI na které se nachází definice rozšíření.

Objekty typu záznam, stopa a událost neobsahují žádná data, ale pouze definují strukturu dokumentu. Veškeré informace o událostech jsou uloženy ve formě atributů.

Logy událostí často obsahují další informace o procesu, jeho instanci nebo konkrétní aktivitě, které nejsou definovány v jádru XES Standardu, ale jsou definovány v některém rozšíření. Následující seznam popisuje několik nejčastěji používaných rozšíření.

- **Organizační rozšíření** (Organizational Extension)[1]. Rozšiřuje standard o informace důležité z hlediska struktury organizace. Doplňuje atributy události o zdroj (`resource`), roli (`role`) a skupinu (`group`).

```
<event>
    ...
    <string key="org:resource" value="112"/>
</event>
```

Výpis 1.5: Ukázka organizačního rozšíření.

- **Časové rozšíření** (Time Extension)[2]. Většina systémů umožňuje přesné ukládání času, kdy byla spuštěna konkrétní úloha. Toto rozšíření umožňuje ukládání času u úloh.

```
<event>
    ...
    <date key="time:timestamp" value="2013-04-16T12:08:22.874+02:00" />
</event>
```

Výpis 1.6: Ukázka časového rozšíření.

- **Rozšíření o náklady** (Cost Extension)[3]. Umožňuje ukládání nákladů spojených s vykonáváním procesu. Cenu lze přiřadit k celému běhu, ke konkrétní události nebo k celému logu.

```
<event>
    ...
    <string key="cost:currency" value="CZK" />
    <float key="cost:total" value="123.50" />
</event>
```

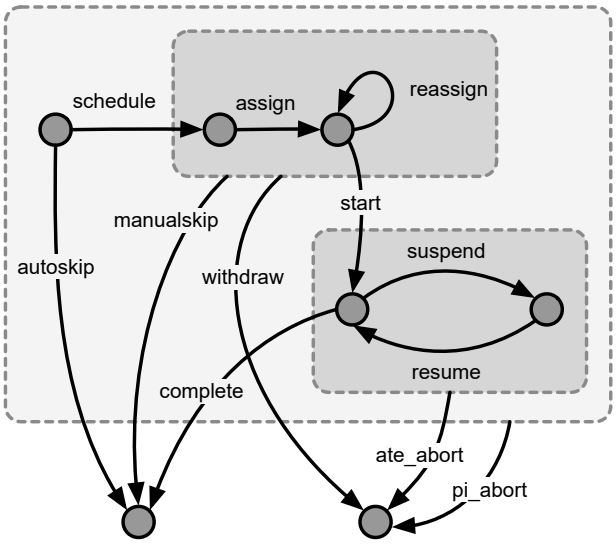
Výpis 1.7: Ukázka rozšíření o náklady.

- **Rozšíření o životní cyklus** (Lifetime extension)[4]. Tento standard přináší podporu životního cyklu události.

```
<event>
    ...
    <string key="lifecycle:transition" value="complete" />
</event>
```

Výpis 1.8: Ukázka rozšíření o životní cyklus aktivit.

[1] http://www.xes-standard.org/org.xesext
[2] http://www.xes-standard.org/time.xesext
[3] http://www.xes-standard.org/cost.xesext
[4] http://www.xes-standard.org/lifecycle.xesext



Obrázek 1.7: Životní cyklus události podle standardu XES [2].

Na obrázku 1.7 je znázorněn standardem definovaný životní cyklus události v podobě konečného automatu. Standard neomezuje použití pouze výše uvedeného životního cyklu, nicméně v této práci budeme předpokládat, že je tento standard dodržován.

Kapitola 2

Process discovery

V této kapitole se podíváme blíže na metody používané v oblasti process discovery. Nejprve si popíšeme základní metriky, kterými budeme hodnotit výsledné modely. Dále si popíšeme principy vybraných základních algoritmů.

2.1 Metriky modelů

V sekci 1.4 jsme si popsali některé požadavky na data v logu událostí, které mohou přímo ovlivňovat kvalitu výstupního modelu. V této sekci si podrobněji popíšeme, jaká kritéria nás u modelu procesu zajímají.

Je téměř nemožné získat takový model procesu, který bude zároveň přesně popisovat veškeré vztahy mezi aktivitami a bude jednoduše čitelný. Často rovněž předpokládáme, že vstupní log událostí neobsahuje žádné chyby. Tedy že jsou přesně zaznamenány všechny události, žádné nejsou vynechány. V reálném světě ovšem logy událostí obsahují chyby.

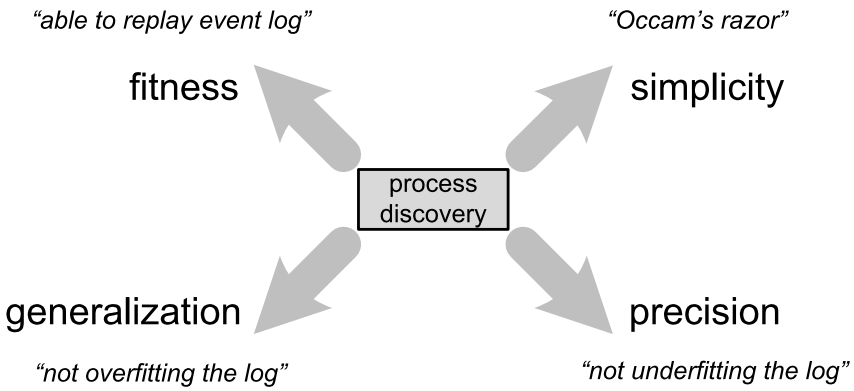
Pro základní ohodnocení přesnosti modelu budeme porovnávat všechny možné běhy, které je schopný model vykonávat.

1. **True positives** - běhy jsou dostupné v reálném procesu i v jeho modelu.
2. **True negatives** - běhy nejsou dostupné v reálném procesu ani v jeho modelu.
3. **False positives** - běhy nejsou dostupné v reálném procesu ale jsou dostupné v jeho modelu.
4. **False negatives** - běhy jsou dostupné v reálném procesu ale nejsou dostupné v jeho modelu.

Abychom dokázali srovnat jednotlivé metody, popíšeme si nejdříve některé metriky, které budeme používat k hodnocení výstupních modelů.

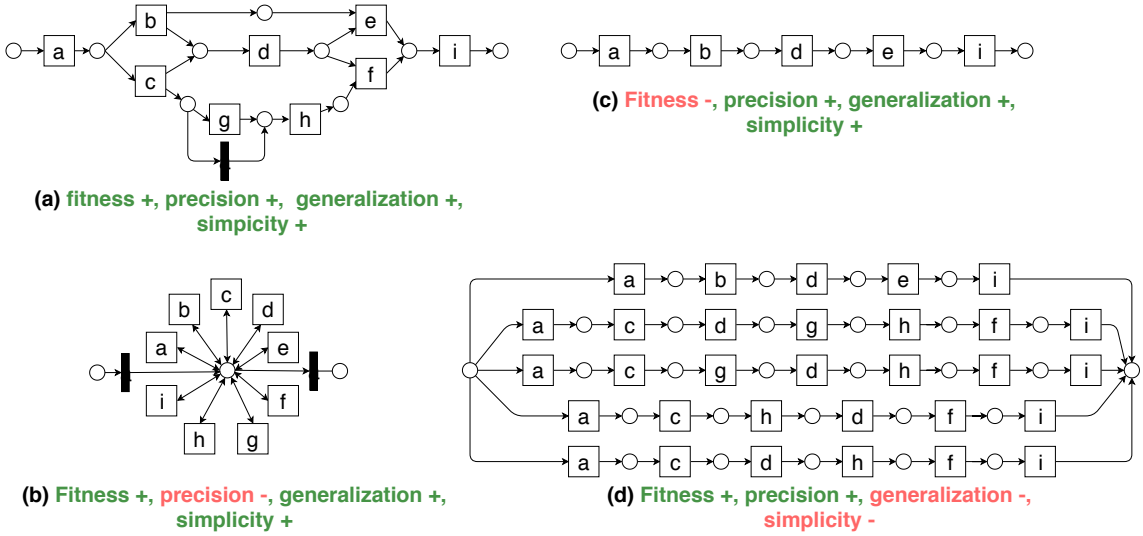
Jedním ze základních předpokladů je, že neexistuje model procesu, který dokonale popisuje chování celého reálného procesu. Můžeme se setkat s tím, že některé události nejsou uložené v logu událostí nebo že v něm nemáme zaznamenané veškeré průběhy. Případně můžeme vytvořit takový model, který sice přesně popisuje chování procesu, ale je tak složitý, že nepřináší žádnou reálnou hodnotu.

Vždy je důležité vytvářet takový model, který nám umožní získat co nejvíce informací. Jednou z hlavních částí je process discovery. V této fázi je sestaven takový model procesu, který popisuje chování popsané v logu událostí.



Obrázek 2.1: Čtyři základní kritéria hodnocení kvality procesních modelů.

Obrázek 2.1 zobrazuje čtyři základní vlastnosti modelu procesu, které budeme hodnotit. Jedná se o fitness, simplicity (jednoduchost), generalization (obecnost) a precision (přesnost).



Obrázek 2.2: Ukázka čtyř různých variant modelu pro vygenerovaných ze stejného vstupního logu událostí.

Na obrázku 2.2 jsou zobrazeny čtyři různé modely procesu (a-d), které byly vygenerovány z logu událostí $L_3 = \{\langle a, b, d, e, i\rangle, \langle a, c, d, g, h, f, i\rangle, \langle a, c, g, d, h, f, i\rangle, \langle a, c, h, d, f, i\rangle, \langle a, c, d, h, f, i\rangle\}$. Na první pohled je zřejmé, že každý z modelů vypadá jinak, a tudíž bude mít i jiné vlastnosti. V následujících sekcích si blíže popíšeme jednotlivá kritéria, kterými budeme hodnotit výsledný model.

- **Fitness.** První dimenzí, kterou budeme hodnotit, je fitness. Tato metrika určuje, do jaké míry je model schopný replikovat všechny průběhy zaznamenané v logu událostí. Nejvyšší hodnotu fitness má takový model, který dokáže přehrát všechny průběhy z logu událostí.

  Například model (c) z obrázku 2.2 má nízkou hodnotu fitness, protože dokáže reprodukovat pouze jeden konkrétní typ průběhu.

- **Simplicity.** Model s dobrou hodnotou simplicity (jednoduchost), by měl být co nejjednodušší. V kontextu procesních modelů tím můžeme rozumět například nejmenší možný počet míst a přechodů výsledné sítě, které dokáží modelovat veškeré chování popsané v logu událostí.
- **Precision.** Další zkoumanou vlastností je precision (přesnost). Zde se zaměřujeme na příliš obecné modely. Takovým případem je model (b) z obrázku 2.2. Ten sice umožňuje běh všech procesů z logu událostí (takže má dobrou fitness), ale zároveň dovoluje jakoukoliv posloupnost.
- **Generalization.** Třetí dimenze se zaměřuje na příliš precizní modely. Takovým případem je model (d) z obrázku 2.2, který povoluje přesně 5 různých průběhů.

2.2 Algoritmy

Pro sestavení modelu z dostupného logu událostí v minulosti vzniklo mnoho algoritmů. Tyto algoritmy používají odlišné přístupy k sestavení modelu procesu.

Tyto algoritmy můžeme řadit do dvou kategorií dle jejich vztahu k šumu v logu událostí. První kategorie algoritmů předpokládá, že log událostí neobsahuje žádný šum a tedy že všechny průběhy procesů jsou zaznamenány správně.

V následujících kapitolách si představíme zástupce algoritmů z každé kategorie.

2.2.1 Používaná terminologie

Než se budeme zabývat principy samotných algoritmů, definujeme základní pojmy, které budeme dále využívat.

Definice 2.1 (Událost, Průběh, Log událostí) *Nechť T je seznam aktivit. $\sigma \in T^*$ je poté průběh instance procesu a log událostí je $L \subseteq T^*$.*

2.3 α algoritmus

Jedním z prvních představených algoritmů pro process discovery je α algoritmus. Tento algoritmus zkoumá sekvence aktivit v rámci průběhů a na jejich základě vytvoří binární relace mezi aktivitami.

Definice 2.2 (Relace přímého následníka - directly follows) *$a, b \in T$, Mezi aktivitami a a b existuje relace přímého následníka $a > b$, pokud pro nějaký průběh procesu $\sigma = (t_1, t_2, t3, ..., t_n)$ a $i \in 1, ..., n-1$ platí, že $t_i = a$ a $t_{i+1} = b$*

Na základě relace přímého následníka vytváříme mezi aktivitami a a b relace:

- **Sekvence**: $a \rightarrow b$ pokud platí $a > b$ a $a \not> b$
- **Výběr**: $a \# b$ pokud platí $a \not> b$ a $b \not> a$
- **Souběžnost**: $a || b$ pokud platí $a > b$ a $b > a$

	a	b	c	d	e
a	$\#_{L_1}$	$\rightarrow_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$
b	$\leftarrow_{L_1}$	$\#_{L_1}$	$\|_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
c	$\leftarrow_{L_1}$	$\|_{L_1}$	$\#_{L_1}$	$\#_{L_1}$	$\#_{L_1}$
d	$\#_{L_1}$	$\leftarrow_{L_1}$	$\leftarrow_{L_1}$	$\#_{L_1}$	$\leftarrow_{L_1}$
e	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$

Tabulka 2.1: Ukázka relací mezi aktivitami v procesu.

V tabulce 2.1 jsou zobrazeny vztahy mezi aktivitami z logu událostí, ve kterém se vyskytují následující průběhy $L_4 = [\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^2, \langle a, e, d\rangle]$.

Následující definice 2.3, převzananá ze zdroje [8], popisuje základní postup algoritmu.

Definice 2.3 (α algoritmus) *Nechť L je log událostí*

1. $T_L = \{t \in T \mid \exists_{\sigma \in L} t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in L} t = first(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in L} t = last(\sigma)\}$,
4. $X_L = \{(A, B) \mid A \subseteq T_L \wedge B \subseteq T_L \wedge \forall_{a \in A} \forall_{b \in B} a \longrightarrow_L b \wedge \forall_{a_1, a_2 \in A} a_1 \#_L a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_L b_2\}$,
5. $Y_L = \{(A, B) \in X_L \mid \forall (A', B') \in X_L A \subseteq A' \wedge B \subseteq B' \implies (A, B) = (A', B')\}$,
6. $P_L = \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$,
7. $F_L = \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$
8. $\alpha(L) = (P_L, T_L, F_L)$

Algoritmus v kroku (1) získá seznam T_L, který obsahuje všechny aktivity, které se v logu vyskytují. Tento seznam koresponduje s výslednými přechody (*transitions*) výsledné Workflow sítě. V kroku (2) získáme seznam všech počátečních aktivit T_I a v kroku (3) získáme seznam všech ukončujících aktivit T_O.

Kritickou částí tohoto algoritmu jsou kroky (4) a (5), ve kterých sestavujeme množiny míst (*places*) výsledné Workflow sítě. Krok (4) vytvoří všechna místa $p(A, B)$, kde $A \subseteq T_L$ a všechny aktivity z množiny A jsou se všemi ostatními aktivitami z množiny A v relaci výběru $\forall_{a_1, a_2 \in A} a_1 \#_L a_2$. Obdobně pro všechny aktivity z množiny B platí, že jsou se všemi ostatními aktivitami z množiny B v relaci výběru $\forall_{b_1, b_2 \in B} b_1 \#_L b_2$. Krok (5) zde poté slouží jako redukce množiny možných míst vygenerovaných v předchozím kroku. Pokud bychom použily všechny dvojice množin $(A, B) \in X_L$, vyniklo by v síti příliš mnoho míst. Množina Y_L tedy obsahuje „maximální dvojice“. Tím chápejme, že pro všechny dvojice $(A, B) \in Y_L$ platí, že pokud $A' \subseteq A$ a $B' \subseteq B$, pak $(A', B') = (A, B)$.

Krok (6) poté sestaví množinu míst, která je složená ze všech prvků množiny Y_L a k nim přidanému startovacímu místu i_L a konečnému místu o_L. Krok (7) poté vytvoří tokovou relaci sítě propojením odpovídajících míst a přechodů.

Jako příklad si opět vezmeme log událostí $L_4 = [\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^2, \langle a, e, d\rangle]$. V tabulce 2.1 jsou znázorněny relace mezi aktivitami z tohoto logu. Aplikací α algoritmu na log události $L = L_4$ získáme:

$$T_L = \{a, b, c, d, e\}$$

$$T_I = \{a\}$$

$$T_O = \{d\}$$

$$X_L = \Big\{\big(\{a\}, \{b\}\big), \big(\{a\}, \{c\}\big), \big(\{a\}, \{e\}\big), \big(\{a\}, \{b, e\}\big), \big(\{a\}, \{c, e\}\big), \big(\{b\}, \{c\}\big), \big(\{c\}, \{d\}\big), \big(\{e\}, \{d\}\big), \big(\{c, e\}, \{d\}\big)\Big\}$$

$$Y_L = \Big\{\big(\{b\}, \{c\}\big), \big(\{a\}, \{b, e\}\big), \big(\{a\}, \{c, e\}\big), \big(\{c, e\}, \{d\}\big)\Big\}$$

$$P_L = \{p_{(\{b\},\{c\})}, p_{(\{a\},\{b,e\})}, p_{(\{a\},\{c,e\})}, p_{(\{c,e\},\{d\})}, i_L, o_L\}$$

$$F_L = \{\big(b, p_{(\{b\},\{c\})}\big), \big(p_{(\{b\},\{c\})}, c\big), \big(a, p_{(\{a\},\{b,e\})}\big), \big(p_{(\{a\},\{b,e\})}, b\big), \big(p_{(\{a\},\{b,e\})}, e\big), \big(a, p_{(\{a\},\{c,e\})}\big), \big(p_{(\{a\},\{c,e\})}, c\big), \big(p_{(\{a\},\{c,e\})}, e\big), \big(c, p_{(\{c,e\},\{d\})}\big), \big(e, p_{(\{c,e\},\{d\})}\big), \big(p_{(\{c,e\},\{d\})}, d\big), \big(i_L, a\big), \big(d, o_L\big)\}$$

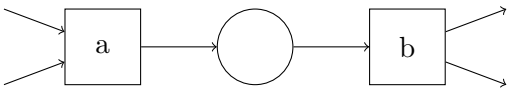
$$\alpha_L = (P_L, T_L, F_L)$$

Po sestavení množiny relací začneme budovat jednotlivé bloky, které budou tvořit výsledný model. Takovými základními bloky jsou:

2.3.1 Převod zjištěných relací do bloků

V této sekci bude popsán princip převodu zjištěných relací mezi jednotlivými úlohami do jejich grafické reprezentace [21].

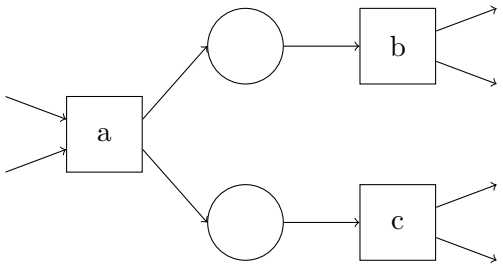
- **Sekvence.** Základním stavebním prvkem je sekvence dvou úloh. Sekvence definuje nejjednodušší vztah mezi dvěma úlohami. Slouží k propojení jednotlivých úloh v systému, které se budou vykonávat postupně. Každý úkol se spouští vždy po dokončení předchozího úkolu.



Obrázek 2.3: Sekvence: $a \rightarrow b$.

Na obrázku 2.3 je ukázka sekvence. Po dokončení úlohy a bude vždy spuštěna úloha b.

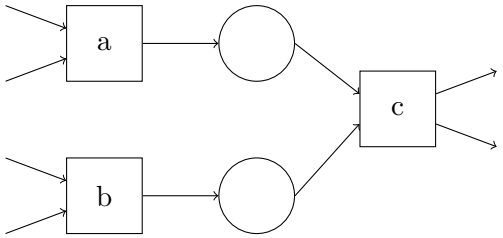
- **AND split** *(Parallel split)*. Rozděluje vykonávání procesu do dvou nebo více vláken, které vykonávají úlohy souběžně. Tato vlákna mohou být volitelně synchronizována v nějakém dalším úkolu (například použitím AND join).



Obrázek 2.4: AND split: $a \rightarrow b$, $a \rightarrow c$ a $b||c$.

Na obrázku 2.4 je ukázka AND split. Po dokončení úlohy a jsou spuštěny obě úlohy b a c.

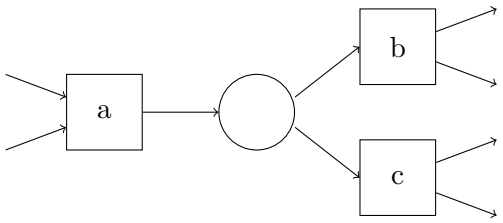
- **AND join** *(Synchronizer)*. Slouží ke sloučení běhu dvou souběžných vláken. Další úloha je spuštěna až po dokončení všech předchozích úloh.



Obrázek 2.5: AND join: $a \rightarrow c$, $b \rightarrow c$ a $a||b$.

Na obrázku 2.4 je ukázka AND join. Úloha c je spuštěna až po dokončení obou úloh a a b.

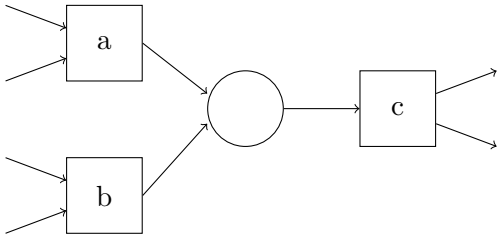
- **XOR split**. Narozdíl od AND joinu, který spouští všechny následné úlohy, XOR split spouští pouze některé. Slouží tedy jako rozhodovací člen.



Obrázek 2.6: XOR split: $a \rightarrow b$, $a \rightarrow c$ a $b\#c$.

Na obrázku 2.4 je ukázka XOR split. Po dokončení úlohy a je spuštěna jedna z úloh b nebo c. Typicky se tak děje na základě vyhodnocení nějaké podmínky.

- **XOR join** *(Synchronizer)*. Slouží ke sloučení dvou souběžných vláken. Další úloha je spuštěna až po dokončení všech předchozích úloh.



Obrázek 2.7: XOR join: $a \rightarrow c$, $b \rightarrow c$ a $b||c$.

Na obrázku 2.4 je ukázka XOR join. Úloha c je spuštěna po dokončení jedné z úloh a nebo b.

2.3.2 Omezení

Smyčky o velikosti jedna. Nedokáže například sestavit modely obsahující krátké smyčky o velikosti jedna. V logu událostí to vypadá jako sekvence stejné aktivity.

Smyčky o velikosti dva. Pro ilustraci tohoto problému si představme log událostí $L_5 = \{\langle a, b, d\rangle, \langle a, b, c, b, d\rangle, \langle a, b, c, b, c, b, d\rangle\}$. α algoritmus předpokládá, že aktivity b a c jsou v relaci $b \parallel c$, neboť v se logu událostí vyskytuje $b > c$ i $c > b$.

Neviditelné aktivity. Některé aktivity nejsou uloženy v logu událostí, protože nekorespondují s žádnou reálnou aktivitou. Nejsou tedy započteny v T_L. Neviditelné aktivity se mohou vyskytovat pokud (i) aktivita slouží jako zjednodušení modelu procesu nebo (ii) se vyskytuje šum v logu událostí, který způsobí, že v něm chybí některé průběhy.

Na základě α algoritmu vyniklo několik odvozených algoritmů (např. $\alpha+$ algoritmus [18], $\alpha++$ algoritmus [24] a $\alpha^{\#}$ algoritmus [25]), které mají za cíl odstranit některá jeho omezeni.

2.4 Heuristic mining

Heuristic mining algoritmus uvažuje, že log událostí může obsahovat méně časté průběhy událostí [23][22]. Při sestavování relací bere v potaz i četnost výskytů zkoumaných sekvencí.

Mezi aktivitami $a, b \in T$ zjišťuje následující relace:

1. $a >_L b$, pokud existuje průběh procesu $\sigma = t_1, t_2, .., t_n$ a $i \in \{1, ..., n-1\}$, kde $\sigma \in L$ a $t_i = a$ a $t_{i+1} = b$. Tuto relaci nazýváme přímý následník (*direct successor*).

2. $a >>_L b$, pokud existuje průběh procesu $\sigma = t_1, t_2, .., t_n$ a $i \in \{1, ..., n-2\}$, kde $\sigma \in L$ a $t_i = a$, $t_{i+1} = b$ a $t_{i+2} = a$. Tuto relaci nazýváme smyčky délky dva (*length-two loops*).

3. $a >>>_L b$, pokud existuje průběh procesu $\sigma = t_1, t_2, .., t_n$ a $i < j$ a $i, j \in \{1, ..., n\}$, kde $\sigma \in L$ a $t_i = a$, $t_j = b$. Tuto relaci nazýváme přímý či nepřímý následník (*direct or indirect successor*).

Dále využíváme hodnoty $|a >_L b|$, která definuje počet výskytů $a >_L b$ v L a $|a >>_L b|$, která definuje počet výskytů $a >>_L b$. Na jejich základě sestavíme míry závislosti:

$$a \Rightarrow_L b = (\frac{|a >_L b| - |b >_L a|}{|a >_L b| + |b >_L a| + 1}), \quad kde\, a \neq b \tag{2.1}$$

$$a \Rightarrow_L a = (\frac{|a >_L a|}{|a >_L a| + 1}) \tag{2.2}$$

$$a \Rightarrow_L^2 b = (\frac{|a >>_L b| - |b >>_L a|}{|a >>_L b| + |b >>_L a| + 1} \tag{2.3}$$

Hodnota $a \Rightarrow_L b$ vždy vychází v rozsahu -1 až 1. Pokud se hodnota $a \Rightarrow_L b$ blíží 1, poté je mezi aktivitami a a b silná pozitivní závislost. To nastává v případě, že je aktivita a často následovaná aktivitou b a jen zřídka kdy je tomu naopak. Je zde jeden speciální případ. Pokud se v posloupnosti událostí často objevuje $a >_L a$, v procesu se pravděpodobně vyskytuje smyčka. Nicméně dle definice 2.1 po dosazení aktivity a vychází $a \Rightarrow_L a = (\frac{|a>_L a|-|a>_L a|}{|a>_L a|+|a>_L a|+1}) = 0$. Z toho důvodu byla zavedena další relace 2.2, která tento případ řeší.

Ve výsledném modelu poté budou spolu propojeny ty hodnoty, které překročí hodnotu prahu:

- **Práh závislosti** *(Dependency threshold).*
  Tento práh určuje minimální hodnotu $a \Rightarrow_L b$, pro kterou je ještě vytvořeno propojení mezi aktivitou a a b.
- **Práh smyček délky jedna** *(Length-one loop threshold).*
  Tento práh určuje minimální hodnotu $a \Rightarrow_L a$, pro kterou je vytvořena smyčka.
- **Práh smyček délky dva** *(Length-two loop threshold).*
  Tento práh určuje minimální hodnotu $a| \Rightarrow_L^2 b|$, pro kterou je ještě vytvořeno propojení mezi aktivitou a a b.

Další heuristikou, která se zde používá, je tzv. „všechny aktivity musí být propojeny“ *(All tasks connected heuristic).* Tato heuristika nabývá pouze hodnoty pravda nebo nepravda. Pokud je heuristika nastavena na hodnotu pravda, vyhledá všechny aktivity, které nemají žádné propojení s ostatními aktivitami a vytvoří propojení s takovou aktivitou, s níž má nejvyšší míru závislosti.

Postup algoritmu si ilustrujeme na následujícím logu událostí L_5:

$$L_5 = [\langle a, e\rangle^5, \langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^{10}, \langle a, b, e\rangle^1, \langle a, c, e\rangle^1, \langle a, d, e\rangle^1 0, \langle a, d, d, e\rangle^2, \langle a, d, d, d, e\rangle^1]$$

$\lvert >_L \rvert$	a	b	c	d	e
a	0	11	11	13	5
b	0	0	10	0	11
c	0	10	0	0	11
d	0	0	0	4	31
e	0	0	0	0	0

Tabulka 2.2: Počet výskytů relace přímého následníka $|a >_{L5} b|$.

Tabulka 2.2 počet výskytů relace přímého následníka v logu událostí L_5.

$\lvert\Rightarrow_{L2}\rvert$	a	b	c	d	e
a	$\frac{0}{0+1} = 0$	$\frac{11-0}{11+0+1} = 0.92$	$\frac{11-0}{11+0+1} = 0.92$	$\frac{13-0}{13+0+1} = 0.93$	$\frac{5-0}{5+0+1} = 0.83$
b	$\frac{0-11}{0+11+1} = -0.92$	$\frac{0}{0+1} = 0$	$\frac{10-10}{10+10+1} = 0$	$\frac{0-0}{0+0+1} = 0$	$\frac{11-0}{11+0+1} = 0.92$
c	$\frac{0-11}{0+11+1} = -0.92$	$\frac{10-10}{10+10+1} = 0$	$\frac{0}{0+0+1} = 0$	$\frac{0-}{0+0+1} = 0$	$\frac{11-0}{11+0+1} = 0.92$
d	$\frac{0-13}{0+13+1} = -0.93$	$\frac{0-0}{0+0+1} = 0$	$\frac{0-0}{0+0+1} = 0$	$\frac{4}{4+1} = 0.80$	$\frac{13-0}{13+0+1} = 0.93$
e	$\frac{0-5}{0+5+1} = -0.83$	$\frac{0-11}{0+11+1} = -0.92$	$\frac{0-11}{0+11+1} = -0.92$	$\frac{0-13}{0+13+1} = -0.93$	$\frac{0}{0+1} = 0$

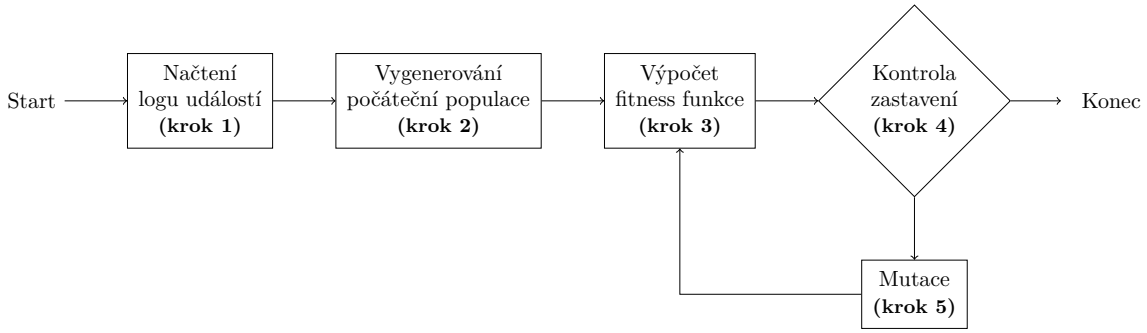
Tabulka 2.3: Míra závislosti.

V tabulce 2.3 je znázorněn výpočet závislosti z logu událostí L_2. Na základě zvolených prahových hodnot poté může vzniknout více modelů.

2.5 Genetické algoritmy

Genetické algoritmy [7][15] ke svému fungování využívají evolučních algoritmů, které iterativně upravují vygenerované modely a hodnotí jejich fitness funkci. V případě generických algoritmů se jako fitness funkce považuje, jak přesně dokáže model reprodukovat chování popsané v logu událostí.

Genetické algoritmy bývají obvykle složené z pěti kroků (viz obrázek 2.8). V prvním kroku je přečten vstupní log událostí. Krok 2 poté vytvoří n modelů, které chápeme jako počáteční populace. Tato populace může být vytvořena úplně náhodným procesem, nebo může pro svůj běh využívat nějakou heuristiku. V kroku 3 se poté vypočítá fitness hodnota každého z procesů. Pokud v kroku 4 zjistíme, že některý z jedinců splňuje naše požadavky, vrátíme daného jedince jako výsledný model procesu.



Obrázek 2.8: Základní schéma genetických algoritmů pro process mining.

2.5.1 Generování počáteční populace

Počáteční fází genetických algoritmů je vygenerování tak zvané inicializační populace. V tomto kroku máme dostupnou informaci o všech aktivitách T, které byly zaznamenány v logu událostí. Každý jedinec bude obsahovat právě tyto aktivity. Lišit se ale budou v relacích T mezi těmito aktivitami.

Můžeme se setkat s přístupy, kdy jsou počáteční relace vygenerovány zcela náhodně (tedy pro každý pár aktivit existuje například 50% šance, že bude relace vytvořena), nebo může být použita nějaká heuristika. Často se setkáme, že heuristika provádí statistickou

analýzu logu událostí. Například pokud se v logu událostí častěji vyskytuje t_1 následovaná t_2 než t_2 následovaná t_1, bude vyšší pravděpodobnost, že se vytvoří relace směřující z t_1 do t_2.

Bylo dokázáno, že použití heuristiky neovlivňuje výsledný model (pokud nemáme omezený počet generací), ale urychlují počáteční fáze evoluce [7].

2.5.2 Kontrola zastavení

Genetický algoritmus zastaví pokud: (i) najde jedince, jehož hodnota fitness je 1; nebo (ii) vypočítá n generací, kde je n maximální povolený počet generací; nebo (iii) nejvyšší hodnota fitness se v posledních $\frac{n}{2}$ generacích ani jednou nezměnila.

2.5.3 Mutace

Základní myšlenkou genetických algoritmů je mutace nejlepších kandidátů z aktuální generace. Z celé generace jsou vybrání ti jedinci, kteří mají nejlepší fitness a je z nich vygenerována nová generace provedením některých operací. U genetických algoritmů pro process mining se nejčastěji používají operace křížení (*crossover*) a náhodná mutace (*mutation*).

2.6 Další algoritmy

Existuje velké množství různých algoritmů pro process mining. Je nad rámec této práce zde všechny algoritmy detailně popsat. Následující sekce si klade za cíl rozšířit již představené algoritmy o některé další zajímavé principy.

- **Fuzzy miner** [13]. Fuzzy miner přináší do process discovery pojmy agregace uzlů (méně časté aktivity jsou sloučeny do tzv. podprocesů), abstrakce (málo četné aktivity mohou být z modelu odebrány) a zdůraznění (ve výsledném modelu jsou četnější aktivity zdůrazněny).
- **Inductive mining.** Je celá kategorie algoritmů, která využívá procesní stromy. Základní Inductive mining algoritmus sestavuje strom aktivit v procesu na základě relace přímého následníka [5].

Kapitola 3

Existující systémy

V této kapitole se podíváme na zástupce nejznámějších nástrojů, které implementují různé algoritmy z oblasti process miningu.

3.1 ProM

ProM je akademický nástroj s otevřeným zdrojovým kódem vyvíjený na Eindhoven University of Technology, na které se nachází výzkumná skupina se zaměřením na oblast process miningu [10].

Hlavní snahou autorů je vytvoření standardního prostředí pro process mining na akademické půdě. Prostředí má umožňovat co nejsnazší vytvoření rozšíření do systému, která implementují teoreticky navržené metody.

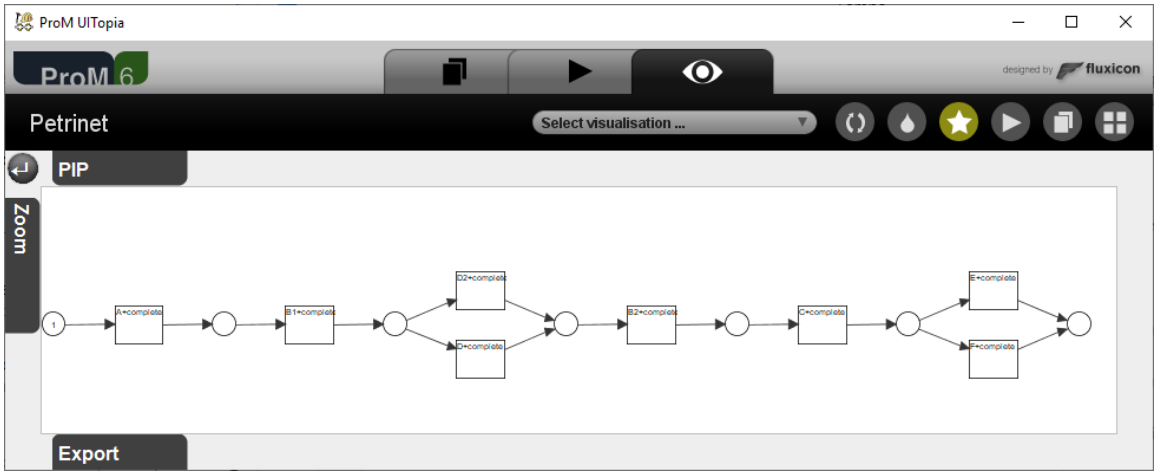
Funkcionalita ProM se skládá ze zásuvných modulů. V současné chvíli je dostupných více než 230 rozšíření, které lze rozdělit do kategorií: [11]

Import a konverze. Moduly pro import umožňují nahrání a zpracování modelů procesů z několika různých formátů. Moduly pro konverzi umožňují převádět mezi těmito formáty. ProM obsahuje podporu například formátů jako je BPMN, PNML (Petriho sítě), nebo YAWL.

Process discovery. Moduly pro dolování implementují funkcionalitu pro process mining, pro který je ProM primárně vyvíjen.

Dostupné jsou některé základní algoritmy, jakou jsou α algoritmus, Genetic mining, Heuristics Miner, nebo Multi-phase mining. Kromě nich existují ale například i rozšíření, která doplňují algoritmy pro méně strukturované procesy (Fuzzy miner).

Moduly pro analýzu. Tyto moduly implementují techniky z oblasti conformance checking. Kontrolují například zda model procesu povoluje veškeré události zaznamenané v logu událostí.



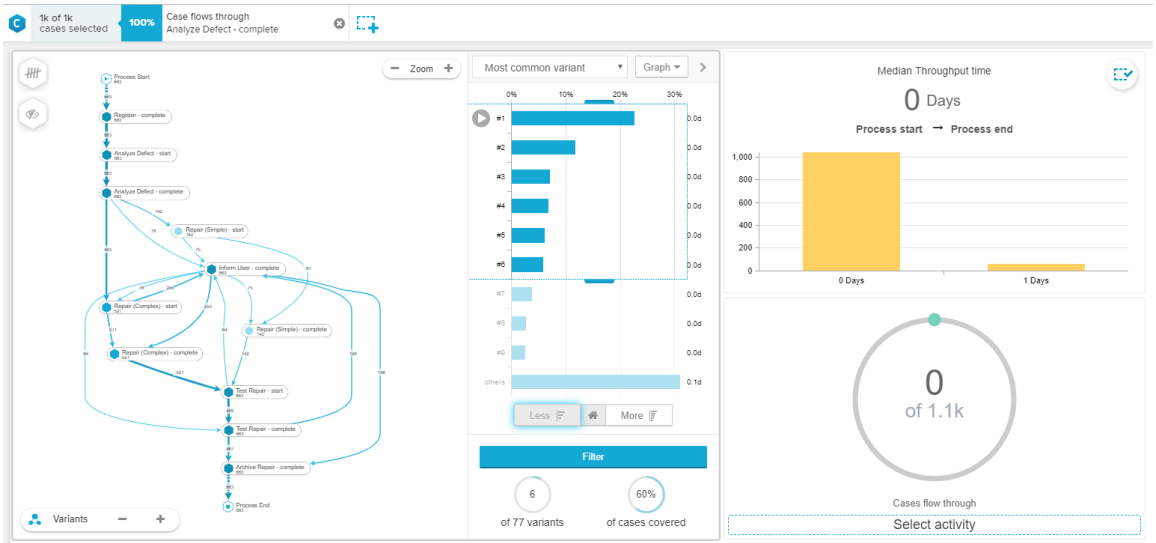
Obrázek 3.1: Zobrazení modelu procesu v prostředí ProM.

Na obrázku 3.1 je ukázka modelu procesu v prostředí ProM.

3.2 Celonis

Jedná se o komerční nástroj vyvíjený stejnojmennou německou firmou. Systém je implementovaný formou webové aplikace. Z uživatelského hlediska se jedná o nejpropracovanější nástroj. Umožňuje jednoduché nahrání logu událostí. Na jeho základě je vytvořeno několik pohledů na proces:

- model procesu
- počet spuštěných instancí (aktivit) za den
- průměrná doba trvání jedné instance



Obrázek 3.2: Zobrazení modelu procesu v prostředí Celonis [1].

Na obrázku 3.2 je znázorněn základní pohled na proces v prostředí Celonis. Kromě samotného modelu zde vidíme i pohled na počet instancí, které trvaly dané období.

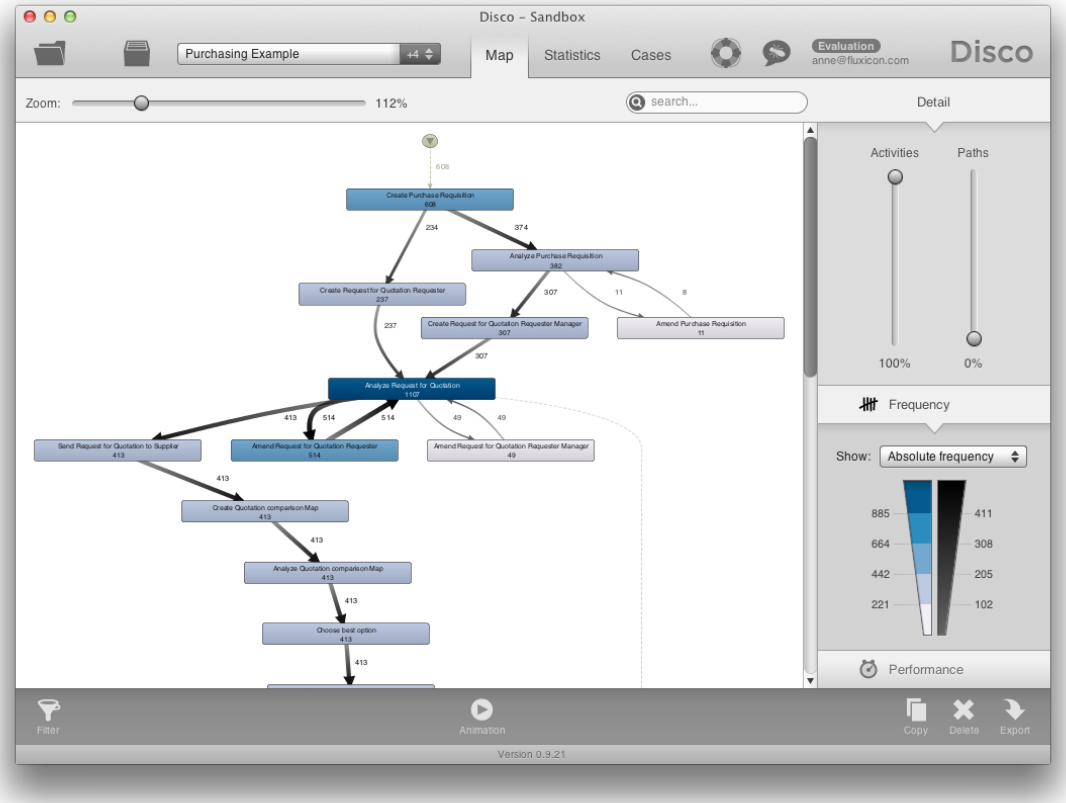
3.3 Fluxicon Disco

Fluxicon Disco je rovněž komerční nástroj, který ovšem poskytuje akademickou licenci. Jedná se o desktopovou aplikaci dostupnou na platformách Windows a MAC. Ke svému běhu využívá upravený Fuzzy Miner [1].

Model procesu může být v tomto nástroji vytvořen jak z celého logu událostí, tak i z jeho vyfiltrované podoby.

Kromě pohledu na celkový proces lze zobrazit i detail konkrétní instance.

Pomocí několika metrik umožňuje zobrazit požadovanou úroveň abstrakce.



Obrázek 3.3: Zobrazení modelu procesu v prostředí Fluxicon Disco [1]

Kromě pohledu na model procesu umožňuje i zobrazení statistik o agregovaných údajích procesů, ve kterých najdeme například průměrnou dobu trvání aktivit, průměrnou dobu obsazení zdroje nebo detailní zobrazení jedné konkrétní instance.

3.4 Závěr

V této kapitole byly popsány vybrané systémy implementující algoritmy z oblasti process miningu. Jednotlivé nástroje se lišily implementovanými přístupy. V sekci 6.4 budou dále

porovnány vlastnosti process discovery přístupů s vlastní implementací na konkrétním příkladu.

V rámci průzkumu trhu nebyl nalezen takový nástroj, který by podporoval predikci chodu rozeběhnutých instancí a na jeho základě vygenerovaném odhadu obsazených zdrojů. V následující kapitole bude navrhnut takový nástroj, který bude umět analyzovat vstupní log událostí. Na jeho základě dojde k vytvoření modelu procesu a bude umožněno provádění simulací procesu.

Kapitola 4

Návrh

Hlavním cílem této práce je vytvoření systému, který umožní provádění simulací nad procesy. V této kapitole si blíže popíšeme jednotlivé komponenty, které má výsledný systém obsahovat.

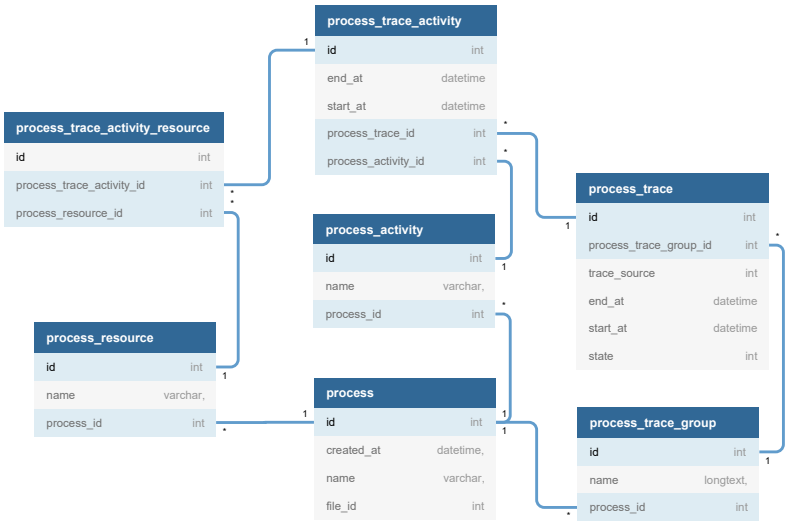
4.1 Import logu událostí

Než začneme analyzovat jakýkoliv proces, musíme o něm umět získat informace. K tomu bude sloužit komponenta pro import souboru ve formátu XES.

Tato komponenta umožní uživateli nahrát nový soubor, nebo použít již v minulosti nahraný soubor. Po odeslání z tohoto souboru jej zpracujeme a uložíme z něj získaná data do databáze.

4.1.1 Ukládání dat

Data o procesech a jejich instancích budou ukládána na MySQL databáze. Na 4.1 obrázku je znázorněno základní schéma databáze pro ukládání informací o procesech a jejich instancích. Návrh schématu vychází z formátu XES představeném v kapitole 1.4.6. Nicméně je zde provedeno několik úprav, které mají zajišťovat statistickou analýzu procesu. Největší nastala u vazby mezi entitami `Process` a `ProcessTrace`, kde mezi ně přibyla nová entita `ProcessTraceGroup`, která slučuje instance se stejným průběhem.



Obrázek 4.1: Schéma části databáze používané pro uložení informací o procesech a jeho instancích

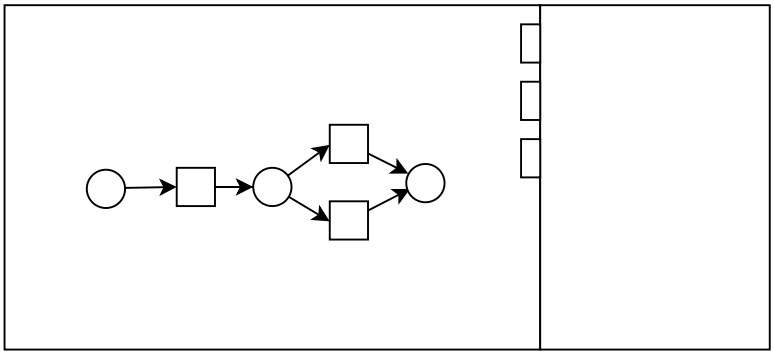
Na obrázku 4.1 je znázorněna část databázového schématu zodpovědná za ukládání informací o procesu a jeho instancích.

- **Process**. Reprezentuje celý proces. Obsahuje referenci na entitu `File`, která obsahuje informace o souboru, ze kterého byl proces importován.
- **ProcessActivity**. Seznam aktivit, které byly v rámci procesu využity. Obsahuje pouze jejich název. Je využíván dalšími tabulkami, které se na tyto aktivity odkazují.
- **ProcessResource**. Seznam všech zdrojů, které byly v rámci procesu použity. Je využíván dalšími tabulkami, které se na tyto zdroje odkazují.
- **ProcessTraceGroup**. Skupina průběhů, které měli stejný průběh. Tato tabulka zde existuje, aby byl jednodušší přístup ke konkrétním datům.
- **ProcessTrace**. Reprezentuje konkrétní instanci procesu. Kromě informací o čase spuštění instance obsahuje i informace o jejím zdroji (v současné chvíli uvažujeme o třech dostupných zdrojích - log události, vytvořená instance v rámci systému a simulace).
- **ProcessTraceActivity**. Konkrétní událost, která nastala v rámci běhu instance procesu. Obsahuje informace o času jejího spuštění a času jejího ukončení.
- **ProcessTraceActivityResource**. V rámci jedné aktivity může být využito více zdrojů (typicky stroj a jeho obsluha). Tato tabulka ukládá k aktivitě informaci o využitém zdroji.

4.2 Model procesu

Tato komponenta bude graficky zobrazovat vytvořený model. Model procesu se bude přepočítávat při zobrazení komponenty, protože uživatel může interaktivně volit, z jakých průběhů

procesu bude model sestaven. Pro zobrazení modelu procesu bude využit Heuristic mining algoritmus, protože předpokládáme, že se v něm mohou vyskytovat i méně časté průběhy. Heuristic mining algoritmus získává v takových případech výrazně kvalitnější modely (jak bylo dokázáno v experimentech v sekci 6.3).



Obrázek 4.2: Komponenta obsahuje v levé části model procesu. V pravé části je postranní panel, který obsahuje detailní informace o aktivitách, zdrojích a průbězích v procesu.

Kromě zobrazení náhledu procesu zde bude postranní panel obsahující detailní informace o procesu. Bude obsahovat čtyři záložky, které budou rozebrány v následujících kapitolách.

4.2.1 Obecné

Zobrazuje celkový počet spuštěných instancí procesu spolu se statistickými údaji o době běhu jednotlivých instancí. Konkrétně zde bude průměrná doba běhu instance, minimální doba běhu instance, maximální doba běhu instance a medián doby běhu instance.

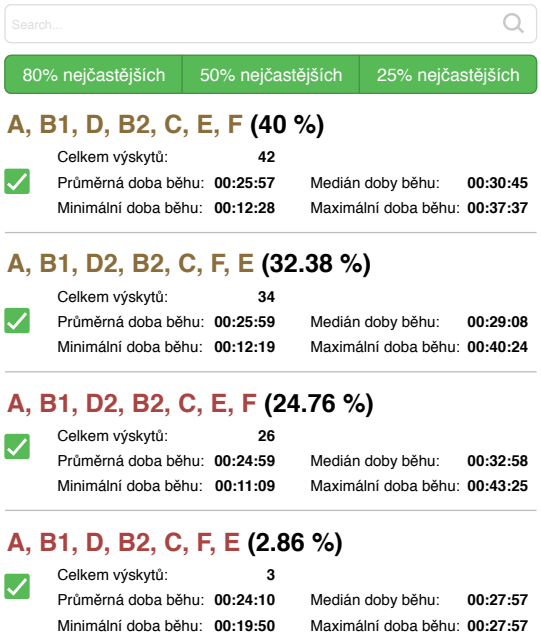
4.2.2 Aktivity

Obsahuje seznam všech aktivit, které se vyskytují v rámci procesu. Každá aktivita má rovněž statistické údaje. Konkrétně zde bude průměrná doba běhu aktivity, minimální doba běhu aktivity, maximální doba běhu aktivity a medián doby běhu aktivity.

4.2.3 Průběhy

Zobrazuje seznam všech unikátních průběhů spolu s informacemi, kolikrát daný průběh nastal. Rovněž obsahuje statistické informace o běhu instancí procesu z dané skupiny.

Každý z průběhů u sebe bude mít zaškrtávací políčko, které bude přímo ovlivňovat, zda bude daný průchod započítán do celkového modelu procesu.



Obrázek 4.3: Seznam průběhů z modelu

Na obrázku 4.3 je znázorněna komponenta zobrazující seznam různých průběhů, které v procesu nastávají. Každá skupina průběhů má u sebe zaškrtávací políčko, které definuje, zda má být tato skupina průběhů zobrazena v modelu procesu. Uživatel díky tomu bude moci interaktivně zobrazovat pouze ty části modelu procesu, které jej v danou chvíli zajímají. Další údaje, které v komponentě zobrazujeme, jsou průměrná, minimální a maximální doba průběhu instance z dané skupiny.

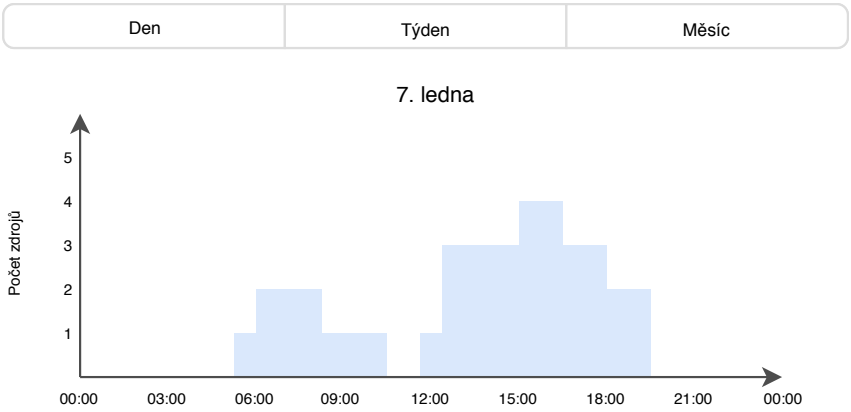
Pro snazší ovládání obsahuje seznam průběhů i dva typy filtru. První filtr omezuje výsledky dle sekvence názvů spuštěných aktivit. Druhý filtr omezuje model takovým způsobem, který zobrazuje pouze nejčastější zaznamenané průběhy.

4.3 Přehled zdrojů

Tato komponenta zobrazuje seznam zdrojů, které byly využity v rámci běhu procesů. U každého zdroje lze vidět, kolikrát byl celkově využit.

4.3.1 Obsazenost zdroje

Na základě spuštěných instancí máme informace o spuštěných aktivitách a o zdrojích, které aktivity využívaly.



Obrázek 4.4: Komponenta zobrazující využití zdroje v čase.

Obrázek 4.4 znázorňuje obsazenou kapacitu zdroje v čase. Je možné využít tří pohledů na časové období - den, týden a měsíc.

Tato vizualizace slouží k přehlednému zjištění požadované kapacity na jednotlivé zdroje.

4.4 Krátkodobé plánování

Jako krátkodobé plánování v kontextu této práce rozumíme predikci vývoje právě spuštěných instancí procesu. K tomu, abychom mohli predikci provádět, musíme nejdříve sestavit model procesu z dodaného logu událostí.

Důležitou komponentou ve vytvořeném systému bude predikce průběhu rozeběhnutých instancí procesu. Tím rozumíme takové průběhy, u kterých byly spuštěny nějaké aktivity, ale proces jako takový ještě nebyl dokončen. Našim cílem bude odhadnout, jaké aktivity budou pravděpodobně spuštěny a jaké zdroje k tomu budou využity.

4.4.1 Zadání rozběhlé instance

Systém bude umožňovat ke každému procesu přiřadit takzvané rozběhlé instance. Jedná se o komponentu, ve které bude moci uživatel vytvořit novou instanci procesu a zvolit:

- jaké aktivity byly vykonány,
- kdy byly vykonány,
- kdy byly ukončeny,
- jaké zdroje byly u jejich běhu využity.

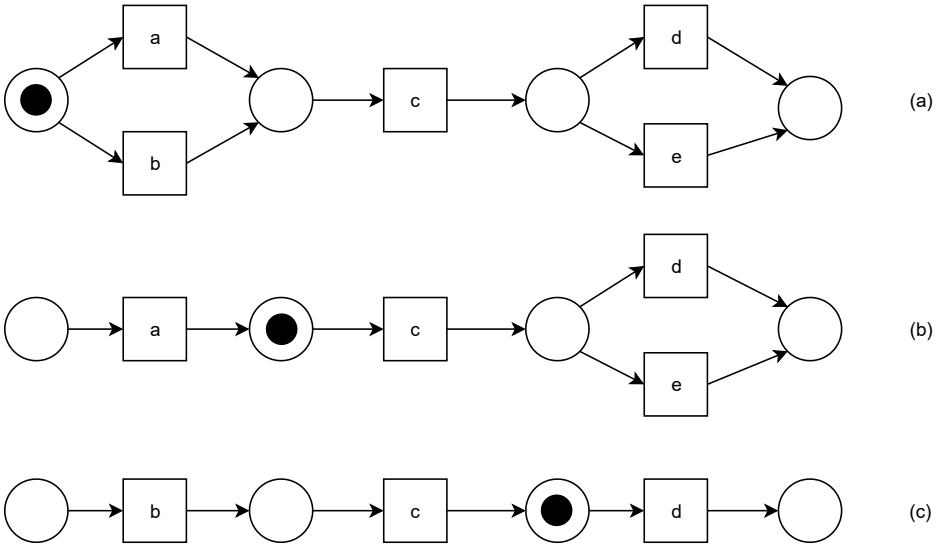
Na základě těchto dat bude systém vytvářet odhady o dalších spuštěných aktivitách a obsazených zdrojích.

4.4.2 Predikce dalšího průběhu instance

Důležitou komponentou ve vytvořeném systému bude predikce průběhu rozeběhnutých instancí procesu. Tím rozumíme takové průběhy, u kterých byly spuštěny nějaké aktivity, ale proces jako takový ještě nebyl dokončen. Našim cílem bude odhadnout, jaké aktivity budou pravděpodobně spuštěny a jaké zdroje k tomu budou využity.

K predikci následujícího průběhu instance budeme využívat znalosti již spuštěných aktivit spolu s jejich parametry a v rámci již dokončených instancí procesu budeme hledat takové průběhy, které se aktuální instanci co nejvíce podobají.

Model procesu sestavíme pouze z té podmnožiny již dokončených instancí, které začínají stejnými aktivitami, jako aktuální instance. Musíme si ovšem dát pozor na aktivity, které mohou probíhat souběžně. Nemůžeme tedy porovnávat sekvenci aktivit jako takovou, ale musíme ji porovnávat v kontextu modelu procesu.



Obrázek 4.5: Model (a) reprezentuje model celého procesu. Modely (b) a (c) představují dva zjednodušené pohledy na proces.

Obrázek 4.5 znázorňuje celkem tři různé modely procesu. Model (a) představuje model celého procesu, který byl vytvořený z logu událostí $L = [\langle a, c, d\rangle^{43}, \langle a, c, e\rangle^{27}, \langle b, c, d\rangle^{26}]$. Modely (b) a (c) zobrazují již vypočtené modely, na základě spuštěné instance. Model (b) vznikl na základě průběhu $\langle a, c\rangle$ a model předpovídá, že se následně spustí ještě aktivita d nebo e.

4.4.3 Predikce času ukončení instance

Jednou z důležitých otázek, kterou při krátkodobém plánování řešíme, je odhadovaná doba ukončení instance. Systém dokáže na základě informací o odhadovaném průběhu procesu predikovat, kdy pravděpodobně skončí.

K tomu budeme využívat model získaný v kapitole 4.4.2. Na jeho základě vidíme, jaké aktivity budou ještě pravděpodobně spuštěny a z minulých průběhů získáme následující informace:

- minimum, maximum, medián a průměr doby vykonávání aktivit
- minimum, maximum, medián a průměr prodlevy mezi spuštěním dvou instancí

Při vykonávání procesu může kdykoliv nastat nějaké zdržení. Například se může stát, že je proces započat před koncem pracovní doby a je dokončen až následující den. Při predikci času dokončení tedy nebereme v potaz čas, kdy byl proces započat, ale vše odvíjíme od

doby spuštění poslední známé aktivity. To sice nevyřeší přesnost prvotního odhadu, ale s následující aktivitou se odhad dokáže odpovídajícím způsobem zpřesnit.

4.4.4 Predikce využitých zdrojů

Předpokládáme, že obecně může existovat více souběžně rozeběhnutých instancí, přičemž každá má přiřazený svůj upravený model procesu.

Na predikci zabraných zdrojů můžeme zvolit dva principy:

1. **Výpočet pravděpodobnosti spuštění aktivity.** Pro každou aktivitu v upraveném modelu vypočítáme procentuální šanci, že bude spuštěna a pro každou aktivitu získáme pravděpodobnosti, že daná aktivita v těchto průbězích využívala určitý zdroj.

   Tyto pravděpodobnosti poté zaneseme do grafu obsazenosti zdrojů. Pro modely obsahující větší množství možných variant průběhů se tato metoda stává méně přehlednou.

2. **Náhodná volba zbylého průběhu instance.** U upraveného modelu si zjistíme, z jakých konkrétních průběhů (spolu s počtem instancí procesů, které v minulosti tento průběh následovaly) je složen. Jednotlivým průběhům přiřadíme váhu podle počtu průběhů - čím více průběhů, tím vyšší váha. Poté vygenerujeme náhodné číslo, které namapujeme na jeden průběh a budeme dále předpokládat, že takový bude vývoj celé instance.

Obě varianty musí umět reagovat na změnu proběhlé instanci. V případě, že se spustí další aktivita, dochází k přepočítání odhadovaného modelu a spolu s tím i k aktualizace predikce zabraných zdrojů.

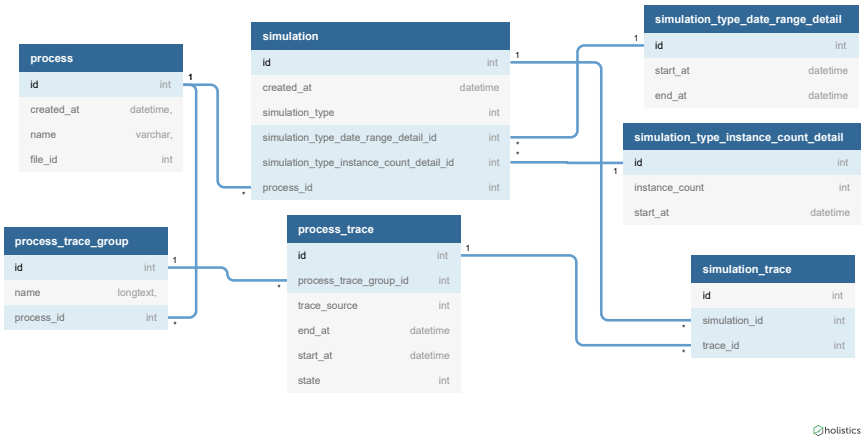
Ve výsledné komponentě byla nakonec zvolena varianta číslo 2, náhodná volba zbylého průběhu instance, z důvodu větší přehlednosti při větším množství spuštěných instancí.

4.5 Simulace

Další komponentou, která bude sloužit k podpoře plánování, je simulace. Hlavním účelem této komponenty je analýza kapacity produkce. K tomu bude vytvořeno několik umělých instancí procesu. Každá instance bude mít svůj průběh, na jehož základě budou sestaveny grafy obsazenosti zdrojů.

4.5.1 Schéma databáze

Pro účely simulace procesů budou využity některé tabulky ze schématu pro ukládání procesů a jeho instancí, ke kterým ještě přibudou čtyři další.



Obrázek 4.6: Schéma databáze pro ukládání informací o simulacích

Obrázek 4.6 znázorňuje část schématu databáze, do kterého ukládáme informace o simulacích.

- **Simulation.** Reprezentuje jednu konkrétní provedenou simulaci. Ukládáme u ní referenci na proces, ze které byla prováděna simulace.
- **SimulationTypeDateRangeDetail** a **SimulationTypeInstanceCountDetail.** Obsahuje informace využité v rámci strategie generování instancí procesů.
- **SimulationTrace.** Tabulka, která propojuje simulaci s konkrétní instancí procesu.

4.5.2 Zpoždění spouštění instancí

Předpokládáme, že v reálném světě nepřicházejí požadavky na spuštění v jednu chvíli, ale je mezi nimi nějaký časový rozestup. Z minulosti známe statistické údaje o době mezi spuštěním jednotlivých instancí, jako je minimální rozestup, maximální rozestup, jeho medián, průměrná hodnota a standardní odchylka.

- **Uniformní rozložení**. Časové rozestupy mezi spuštěním jednotlivých instancí jsou voleny náhodně v rozmezí od minimálního rozestupu až po maximální.
- **Normální rozložení**. Časové rozestupy mezi spuštěním jednotlivých instancí jsou voleny dle normálního rozložení, kde je použita průměrná hodnota rozestupu spolu se standardní odchylkou.

4.5.3 Počet spuštěných instancí

Další hodnotou, kterou uživatel při vytváření simulace nastavuje, je počet spuštěných simulací. Zde existují opět dvě strategie:

- **Přímo zadaný počet instancí.** V rámci této strategie uživatel zadá datum spuštění první instance a celkový počet spuštěných instancí. Systém poté dopočítá předpokládaná data spuštění a dokončeních instancí. Tuto strategii můžeme využít například v případě, že očekáváme zakázku na výrobu konkrétního počtu kusů produktu.

- **Počet instancí dle období spuštění.** U druhé strategie zadáváme datum spuštění první instance a datum spuštění poslední instance. Systém poté generuje události tak dlouho, dokud nepřesáhne datum spuštění poslední instance. Tuto strategii můžeme využít například pokud chceme zjistit kapacitu výroby.

4.5.4 Zvýšení zátěže

Důležitým atributem při vytváření simulace je multiplikátor rychlosti vzniku nových instancí. Pokud je tento multiplikátor nastaven na hodnotu 1, budou nové instance procesu vznikat s takovým časovým rozestupem, který byl zjištěn analýzou logu událostí. Systém ovšem umožní tento čas libovolně upravovat. Pokud nastavíme hodnotu multiplikátoru na 2, budou nové instance procesu vznikat dvakrát rychleji. Díky tomu můžeme zjistit, jak se proces chová v případu zvýšení zátěže a zda má podnik například uvažovat nad zvýšením kapacity výrobní linky.

Kapitola 5

Implementace

V této kapitole popíšeme vybrané problémy, které byly řešeny v rámci implementace. Nejprve se podíváme na použité technologie. Dále si popíšeme způsob zpracování logu a nakonec si popíšeme optimalizace použité v rámci α algoritmu.

5.1 Použité technologie

Komponenty navržené v předchozí kapitole budou implementovány do sytému, který bude založen na principu klient-server.

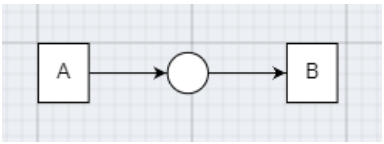
5.1.1 Klientská část

Klientská část systému bude mít podobu webové aplikace. Její jádro bude napsáno s využitím MVC frameworku AngularJS.

Model procesu

Pro zobrazení modelu procesu bude využita knihovna GoJS[1]. Jedná se o knihovnu napsanou v jazyce TypeScript, která slouží k vytváření a zobrazování interaktivních grafů. Do této knihovny bylo dopsáno rozšíření, které umožňuje zobrazení Workflow sítí.

```
1  const nodes = [
2      {key: "A", nodeType: NodeType.Transition},
3      {key: "B", nodeType: NodeType.Transition},
4      {key: "P(A, B)", nodeType: NodeType.Place},
5  ];
6
7  const links = [
8      {from: "A", to: "P(A, B)"},
9      {from: "P(A, B)", to: "B"},
10 ];
```



Obrázek 5.1: Ukázka kódu pro zobrazení modelu procesu

V ukázce 5.1 je znázorněno použití této knihovny pro zobrazení jednoduchého grafu, který obsahuje dvě místa a jeden přechod.

[1] https://gojs.net

Grafy

Pro zobrazení obsazenosti zdroje budou využity grafy, které na X ose zobrazují datum a na Y ose zobrazují počet alokovaných jednotek. Při implementaci bude využita knihovna HighCharts[2], která umožňuje jednoduché zobrazení interaktivních grafů.

5.1.2 Serverová část

Serverová část je napsaná v jazyce Java za použití frameworku Spring Boot[3]. Tento framework umožňuje jednoduché vytváření webových aplikací.

Pro práci s databází byla zvolena knihovna Hibernate. Jedná se o framework poskytující pokročilé funkce ORM[4] napsaný v jazyce Java.

5.2 Načtení a zpracování logu událostí

Vstupem do analýzy procesu je log událostí ve formátu XML. Tento soubor je dále zpracován do interní struktury, se která bude poté dále zpracovávat. Cílem je z logu událostí získat následující údaje:

- kolekci průběhů,
- posloupnost aktivit v rámci každého průběhu,
- u každé spuštěné aktivity zjistit její začátek, konec a zdroj, jaký byl využit.

V následujících sekcích si blíže popíšeme některé vlastnosti obou přístupu a následně v sekci 6.2 oba přístupy porovnáme.

5.2.1 OpenXES

OpenXES je referenční implementace standardu formátu XES. Data zpracovává do takové struktury, která přesně odpovídá standardu.

Tato knihovna má pro naše využití několik nedostatků:

- **Zpracování informací o životním cyklu události.** Knihovna zpracovává událost přesně tak, jak se vyskytují v logu událostí. V případě, že se u události nachází informace o životním cyklu aktivity, projeví se to tak, že se za název události přidá textový popis popis životního cyklu.

  Pokud se podíváme na log událostí uvedený v kódu 5.2, získáme sekvenci $\langle a{+}start, a{+}completed, b\rangle$. Pro účely plánování potřebujeme získat informace o době trvání události, proto bychom museli provést další zpracování těchto dat a události sloučit do jedné.

- **Vyšší paměťové nároky.** Vyšší paměťová náročnost této implementace je daná její obecností. Pro účely krátkodobého plánování nám stačí získávat informace pouze o názvu aktivity, času spuštění, životním cyklu aktivity a zdrojích. Knihovna OpenXES zpracovává všechny atributy, které se v logu nachází a ukládá je do vnitřní struktury.

  V logu událostí se typicky často opakuje několik stejných atributů (konkrétně máme na mysli název aktivity a identifikaci zdroje).

[2] https://www.highcharts.com/
[3] https://spring.io/projects/spring-boot
[4] Objektově relační mapování

5.2.2 Vlastní implementace

V rámci této diplomové práci byla vytvořena vlastní implementace, která si kladla za cíl zrychlení parsování a zmenšení paměťové náročnosti s ohledem na data, která jsou potřebná pro sestavení modelu.

Parser zpracovává pouze atributy, které bude dále využívat. Konkrétně se jedná o informace o času spuštění události, aktivitě, zdroji a případně o životním cyklu. Přímo při zpracování kontroluje životní cyklus událostí. Události, týkající se jednoho životního cyklu aktivity jsou slučovány.

Atributy reprezentující aktivitu a zdroj jsou ukládány do hash mapy. Pokud při zpracování narazíme na nějakou aktivitu, vložíme ji do hash mapy a dále pracujeme pouze s její referencí.

5.3 Zpracování posloupnosti událostí

Pro sestavení odpovídajícího modelu procesu musíme nejprve získat informace o spuštěných instancích procesu. Na jejich základě sestavíme odpovídající model, který budeme dále používat pro predikci dalších instancí.

Systém bude umožňovat zpracovat soubory ve formátu XES, který byl popsán v kapitole 1.4.6. Důležitou částí zpracování logu událostí je získání informací o době běhu jednotlivých aktivit. V následujících dvou kapitolách si popíšeme principy, které budou použity v závislosti na dostupných datech v logu událostí.

5.3.1 Aktivity bez životního cyklu

Ve většině typů logu událostí se setkáme s aktivitami, které nemají svůj životní cyklus (mohou sice implementovat lifecycle extension jako v ukázce 5.1, ale používají pouze stav *completed*).

```
1  <trace>
2    <event>
3      <date key="time:timestamp" value="2013-04-16T10:19:36.723+02:00"/>
4      <string key="concept:name" value="A"/>
5      <string key="lifecycle:transition" value="complete"/>
6      <string key="org:resource" value="SolverC3"/>
7    </event>
8    <event>
9      <date key="time:timestamp" value="2013-04-16T10:21:57.327+02:00"/>
10     <string key="concept:name" value="B"/>
11     <string key="lifecycle:transition" value="complete"/>
12     <string key="org:resource" value="SolverC3"/>
13   </event>
14
15   ..
16 </trace>
```

Výpis 5.1: Ukázka logu událostí obsahující informace o životním cyklu událostí

V takovém případě předpokládáme, že aktivita končí ve chvíli, kdy bude spuštěna další aktivita. Na příkladu z kódu 5.3 tedy předpokládáme, že aktivita A byla spuštěna 16. dubna 2013 v 10:19:36 a byla ukončena 16. dubna 2013 v 10:21:57.

5.3.2 Aktivity s životním cyklem

V tomto případě některé aktivity v logu událostí obsahují informace o životním cyklu události. Tyto data budou využity ke zjištění přesné doby běhu konkrétní aktivity a doby mezi ukončením jedné aktivity a spuštěním druhé aktivity.

```
<trace>
  <event>
    <date key="time:timestamp" value="2013-04-16T10:19:36.723+02:00"/>
    <string key="concept:name" value="A"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="org:resource" value="SolverC3"/>
  </event>
  <event>
    <date key="time:timestamp" value="2013-04-16T10:21:57.327+02:00"/>
    <string key="concept:name" value="A"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="org:resource" value="SolverC3"/>
  </event>
  <event>
    <date key="time:timestamp" value="2013-04-16T11:22:32.167+02:00"/>
    <string key="concept:name" value="B"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="org:resource" value="Tester1"/>
  </event>
  ..
</trace>
```

Výpis 5.2: Ukázka logu událostí obsahující informace o životním cyklu událostí

V kódu 5.2 je ukázka logu, která obsahuje životní cyklus aktivity A. Tato aktivita byla spuštěna 16. dubna 2013 v 10:19:36 a byla ukončena 16. dubna 2013 v 10:21:57. Na rozdíl od předchozího případu zde víme, že aktivita B byla spuštěna až přibližně půl minutovým zpožděním po ukončení aktivity A.

```
<trace>
  <event>
    <date key="time:timestamp" value="2013-04-16T10:19:36.723+02:00"/>
    <string key="concept:name" value="A"/>
    <string key="lifecycle:transition" value="start"/>
  </event>
  <event>
    <date key="time:timestamp" value="2013-04-16T10:21:57.327+02:00"/>
    <string key="concept:name" value="A"/>
    <string key="lifecycle:transition" value="complete"/>
  </event>
  <event>
    <date key="time:timestamp" value="2013-04-16T11:22:32.167+02:00"/>
    <string key="concept:name" value="A"/>
    <string key="lifecycle:transition" value="start"/>
  </event>
  ..
</trace>
```

Výpis 5.3: Ukázka logu událostí obsahující informace o životním cyklu událostí

Pokud log událostí obsahuje aktivity s životními cykly, musíme při jeho zpracování dále kontrolovat:

- **Cyklus stejné aktivity.** V procesech se můžeme setkat s krátkými cykly (stav, kdy se stejná aktivita spustí vícekrát přímo za sebou). V kódu 5.3 je znázorněna část průběhu instance procesu, kdy je dvakrát po sobě spuštěna aktivita A.

  Při zpracování této sekvence musíme brát ohled na definici životního cyklu aktivity (viz obrázek 1.7). Na první pohled se může zdát, že se všechny tři události vztahují k jedné aktivitě, ale po bližším prozkoumání zjistíme, že se pravděpodobně jedná o dvě aktivity, přičemž první dvě události se vztahují k prvnímu spuštění a třetí událost ke druhému.

- **Vnořená událost souběžné aktivity.** Dalším z problémů, které při zpracování logu událostí může nastat, je vnoření události souběžné aktivity. Tím myslíme, že se zde vyskytují události A s životním cyklem „start“, B a A s životním cyklem „complete“.

V praxi oba případy řešíme pomocnou strukturou, která obsahuje již zpracované aktivity, které nebyly dle životního cyklu ukončeny. Při příchodu každé aktivity nejdříve zkontrolujeme, zda se v této struktuře nenachází nějaká nedokončená aktivita, která může dle definovaného životního cyklu přejít do stavu právě zpracovávané aktivity.

5.4 Process discovery

Při implementaci process discovery algoritmů byl kladen důraz na co nejjednodušší rozšířitelnost o další algoritmy. V rámci diplomové práce byly implementovány následující algoritmy:

- α algoritmus,
- Heuristic mining algoritmus.

5.4.1 Rozšiřitelnost

Pro implementaci Process Discovery algoritmů bylo vytvořeno rozhraní `IProcessDiscovery`, které konkrétní algoritmy implementují. Díky tomu lze snadno doplnit další algoritmy process discovery, přičemž každý z nich může mít definovaný formát výstupního modelu.

5.4.2 α algoritmus

Při vlastní implementaci se ukázalo, že klíčovým parametrem pro rychlé vygenerování modelu bylo vytvoření množiny XL. K jejími generování byly použity následující postupy:

- **Generování potenční množiny aktivit**. Jako první naivní postup pro generování množiny XL bylo využito vygenerování potenční množiny aktivit, z nichž byly poté odfiltrovány množiny obsahující aktivity, pro které neplatí $\forall a_1, a_2 \in A, a_1 \# a_2$.

  Tento přístup stačil na procesy obsahující menší množství aktivit. Nicméně potenční množina má mohutnost 2^N. Složitost tohoto postupu tedy byla exponenciální.

- **Heuristika generování množin** Jako reakce na složitost prvního postupu vznikly dvě heuristiky, které vedly k několikanásobnému zrychlení celého algoritmu.

  1. $X_L = \{\{t\} \mid t \in T_L\}$

2. $X_L^2 = \emptyset$
3. Pro každé P v X_L a pro každé t v T_L:
   (a) Přidej $P \cup \{t\}$ do X_L^2 pokud:
      i. $\forall a \in P : a \#_L t$ a
      ii. $\exists b \in T, \forall a \in (P \cup \{t\}) : a \rightarrow_L b$
4. Pokud $X_L^2 = \emptyset$, vrať X_L, jinak přidej X_L^2 do X_L a pokračuj bodem 2.

V rámci optimalizace byla snaha co nejvíce omezit množinu všech podmnožin T_L. Nejdříve vymyšlen bod (i) algoritmu, který kontroluje, zda je nově přidávaná aktivita a v relaci výběru se všemi stávajícími aktivitami v množině aktivit P. Tento bod sám o sobě zrychlil výpočet množiny X_L pro $|T_L| = 24$ z několika desítek minut na přibližně 30s.

Další optimalizaci přinesla implementace bodu (ii), který kontroluje, zda existuje alespoň jedna další aktivita $b \cup T$, která je v relaci se všemi aktivitami v množině P a zároveň je ve stejné relaci s nově přidávanou aktivitou. Tato optimalizace dále zrychlila celý výpočet množiny X_L pro $|T_L| = 24$ z původních 30s na 27ms.

Kapitola 6

Testování

Tato kapitola se zaměřuje na popis experimentů prováděných na implementovaných algoritmech. Nejdříve si popíšeme datovou sadu, která byla pro testy využita. Následně si otestujeme výkon parsování logů událostí. Dále si porovnáme výstupy implementovaných process discovery algoritmů a na konec porovnáme jejich kvalitu s existujícími řešeními.

6.1 Datová sada

Pro testování jednotlivých komponent byly využity logy událostí, které jsou volně dostupné na portálu nizozemského centra pro výzkum 4TU[1]. Datové sady, které jsou zde dostupné, můžeme dělit do dvou základních kategorií:

- **Syntetické**. Data vytvořená pouze pro účely testování metod process miningu. Často jsou vytvořené za účelem kontroly konkrétní problematiky (například detekce krátkých smyček).
- **Reálné**. Data z reálných aplikací. Na rozdíl od syntetických dat se tato data vyznačují větší mírou šumu v datech.

6.1.1 Použité datové sady

Z dostupné sady byly vybrány takové datové sady, které mají testovat různé problémy, se kterými se můžeme setkat při zpracování logu událostí.

Název datasetu	Počet aktivit	Počet různých průběhů	Typ dat
ETM_configuration1	7	11	Syntetická
ETM_configuration2	7	2	Syntetická
ETM_configuration3	5	2	Syntetická
ETM_configuration4	8	4	Syntetická
Phone_repair	8	60	Syntetická
BPI_challenge_2012	24	13 680	Reálná

Tabulka 6.1: Porovnání datasetů

V tabulce 6.1 je seznam datových sad, které byly využity v rámci testování implementace.

[1] $https://data.4tu.nl/repository/collection:event_logs$

6.2 Srovnání parserů logu událostí

Tyto experimenty mají za cíl porovnat přístupy k parsování logu událostí představené v kapitole 5.2. Knihovna OpenXES načítá veškeré informace dostupné v logu událostí. Vlastní implementace parseru zpracovává pouze informace o názvu aktivity, jejím životním cyklu, času spuštění a využitých zdrojích. Na druhou stranu přímo do procesu parsování přidává kontrolu a zpracování údajů o životním cyklu události. Z těchto důvodů předpokládáme, že vlastní implementace bude mít srovnatelnou časovou náročnost, ale bude mít výrazně nižší požadavky na paměťový prostor.

Velikost souboru	Počet průběhů	Počet aktivit	Počet zdrojů	OpenXES		Vlastní		Porovnání	
				Paměť	Čas	Paměť	Čas	Paměť	Čas
160 KiB	105	8	0	991 KiB	813 ms	151 KiB	146 ms	15,2 %	17,9 %
73 958 KiB	13 087	24	68	132 651 KiB	5 832 ms	42 953 KiB	4 353 ms	32,3 %	74,6 %
343 259 KiB	16 384	128	0	690 569 KiB	18 934 ms	278 976 KiB	14 742 ms	40,3 %	77,8 %
1 360 573 KiB	262 144	512	0	2 768 347 KiB	76 748 ms	1 270 510 KiB	63 767 ms	45,8 %	83,3 %

Tabulka 6.2: Porovnání vlastní implementace XES parseru s referenční implementací OpenXES

Tabulka 6.2 zobrazuje naměřené časové a paměťové nároky obou implementací. Bylo použito několik různých vstupních logů události, které se lišily především ve velikosti vstupního souboru a počtu zaznamenaných aktivit.

Z porovnání vyplývá, že vlastní implementace má výrazně nižší paměťové nároky. Spotřebovaná paměť byla vždy menší než velikost vstupního souboru, zatímco u knihovny OpenXES byla paměťová náročnost dokonce větší než je velikost vstupního souboru.

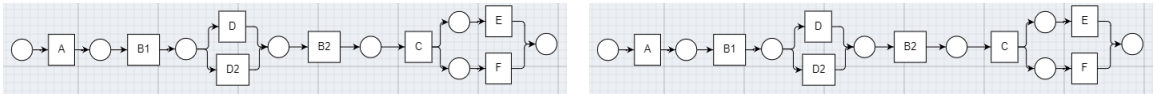
Časová náročnost obou implementací je srovnatelná, přičemž vlastní implementace si ve všech testovacích případech vedla lépe.

6.3 Process discovery

Následující testy si kladou za cíl porovnat výstupy algoritmů použitých pro process discovery. Nejdříve algoritmy otestujeme na jednoduchých procesech. U jednoduchých procesů předpokládáme, že ani jeden z představených algoritmů nebude mít problém se sestavením modelů. Dále se ovšem zaměříme na takové procesy, u kterých se ukáží přednosti či nedostatky zkoumaných algoritmů.

6.3.1 Jednoduchý proces

Základním testem bude porovnání logu událostí na logu událostí `ETM_configuration4.xes`. Tento log událostí obsahuje následující průběhy $L_4 = [\langle a, b1, d, b2, c, e, f\rangle^{42}, \langle a, b1, d2, b2, c, f, e\rangle^{34}, \langle a, b1, d2, b2, c, e, f\rangle^{26}, \langle a, b1, d, b2, c, f, e\rangle^{3}]$.



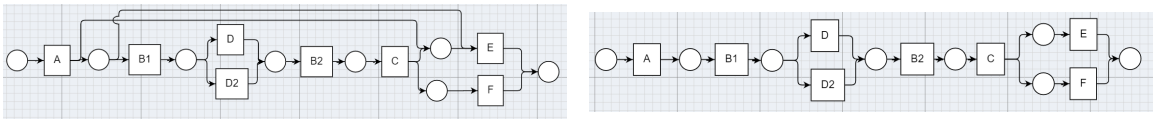
(a) Model získaný α algoritmem (b) Model získaný Heuristic mining

Obrázek 6.1: Porovnání jednoduchého modelu

Obrázek 6.1 zobrazuje modely získané pomocí α algoritmu, respektive Heuristic mining algoritmu. Tento test dokazuje, že si oba přístupy vedou srovnatelně na jednoduchých a dobře strukturovaných procesech.

6.3.2 Log událostí obsahující šum

Další test ověřuje, jakým způsobem se metody chovají u procesů, které obsahují málo četné průběhy. Pro tento test byl využit log událostí L_4, do kterého byl doplněn šum a vznikl tak $L_5 = [\langle a, b1, d, b2, c, e, f\rangle^{42}, \langle a, b1, d2, b2, c, f, e\rangle^{34}, \langle a, b1, d2, b2, c, e, f\rangle^{26}, \langle a, b1, d, b2, c, f, e\rangle^{3}, \langle a, e\rangle]$.



(a) Model získaný α algoritmem

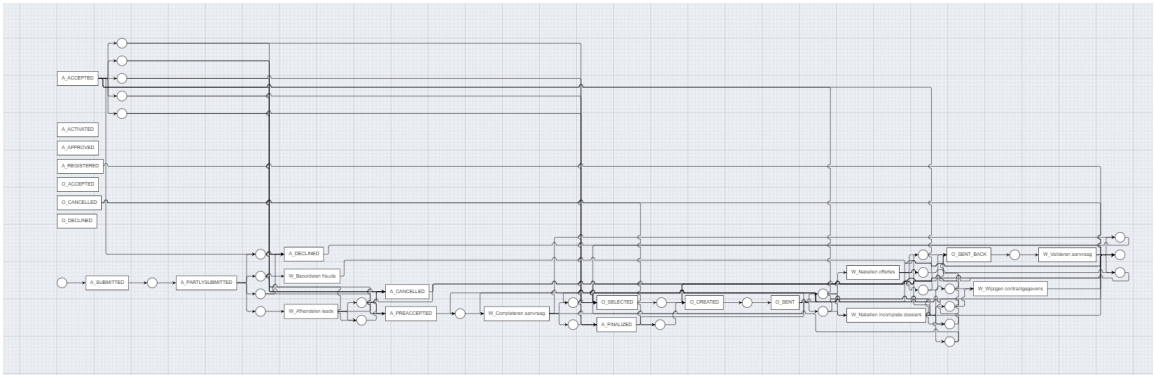
(b) Model získaný Heuristic mining

Obrázek 6.2: Porovnání modelu vygenerovaného z logu obsahující šum

Na obrázku 6.2 jsou znázorněny dva modely vzniklé ze stejného logu událostí. Log událostí obsahuje průběh $\langle a, e\rangle$, který se zde vyskytuje pouze jednou a má relativní zastoupení 0,94 %. Model 6.2a, získaný pomocí α dovoluje i tento průběh. Model 6.2b tento průběh nedovoluje. Jinými slovy model 6.2a má oproti modelu 6.2b lepší hodnotu fitness.

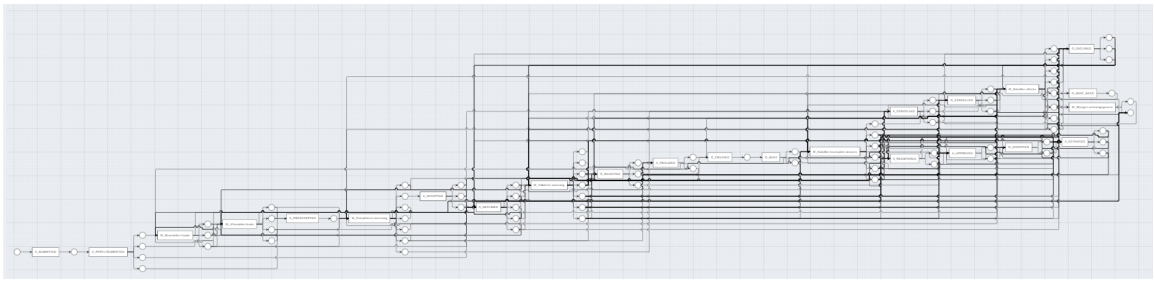
6.3.3 Log událostí s velkým množstvím šumu

Hlavní síla Heuristic mining algoritmu spočívá v jeho schopnosti zpracování šumu. Tato vlastnost byla testována na logu událostí `BPI challenge 2012`.



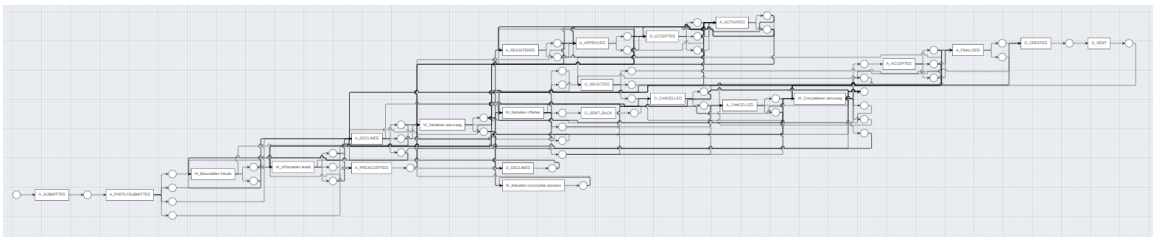
Obrázek 6.3: Model procesu z logu událostí BPI_challenge 2012 získaný pomocí α algoritmu

Obrázek 6.3 zobrazuje model procesu, který vznikl aplikací α algoritmu na log událostí `BPI_challenge 2012`. Vzniklý model obsahuje sedm přechodů, do kterých nevede hrana z žádného místa (na obrázku jsou vlevo nahoře), nelze je tedy spustit.



Obrázek 6.4: Model procesu z logu událostí BPI_challenge 2012 získaný pomocí Heuristic mining algoritmu

Na obrázku 6.4 je model vygenerovaný Heuristic mining algoritmem. Na rozdíl od předchozího algoritmu jsou zde všechny přechody dosažitelné.



Obrázek 6.5: Model procesu z logu událostí BPI_challenge 2012 získaný pomocí α algoritmu s omezením na 75 % nejčastějších průběhů.

Obrázek 6.5 zobrazuje model, který vznikl ze 75 % nejčastějších typů průběhu v logu událostí `BPI_challenge 2012`. Výstupní již model neobsahuje nepropojené aktivity na rozdíl od modelu z obrázku 6.3.

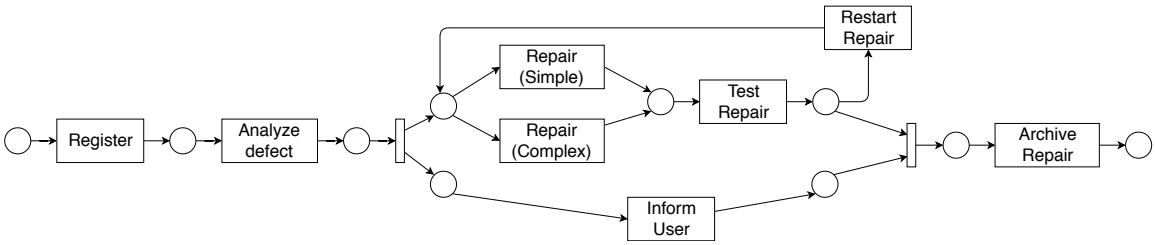
Tento model můžeme dále srovnat s modelem vzniklým pomocí Heuristic mining algoritmu (obrázek 6.4). Struktura obou modelu je velmi podobná. Hlavním rozdílem je absence jedné aktivity v modelu procesu vygenerovaného α algoritmem.

6.3.4 Závěr

Na výše uvedených experimentech byla ověřena vlastnost algoritmů. U jednoduchých procesů, které neobsahují málo četné průběhy, se oba algoritmy chovají srovnatelně. Hlavním rozdílem je právě jejich chování u logů událostí obsahující méně četné průběhy. V takových případech byly lepší výsledky z Heuristic mining algoritmu.

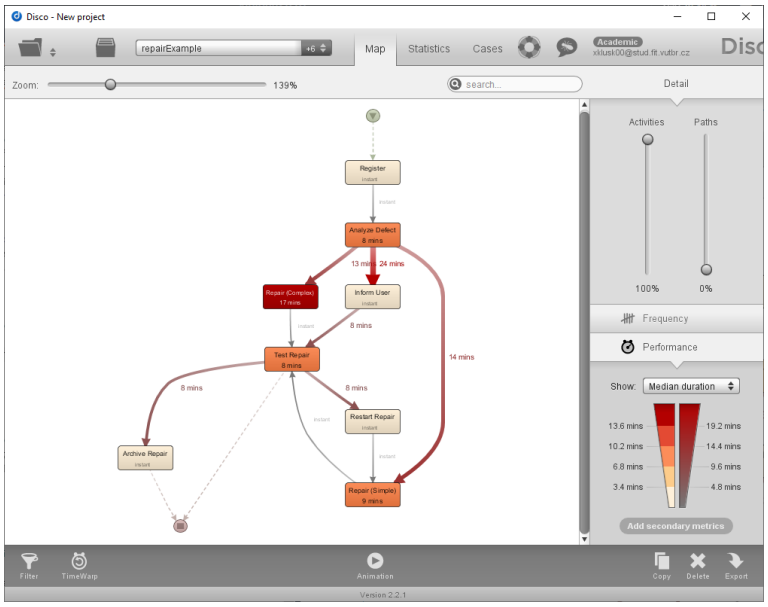
6.4 Srovnání s existujícími systémy

Implementace process discovery algoritmů byla dále srovnána se systémy ProM, Fluxicon Disco a Celonis. Cílem bylo zjistit odlišnosti při sestavení modelu procesu. Testování proběhlo na logu událostí `Phone repair`.



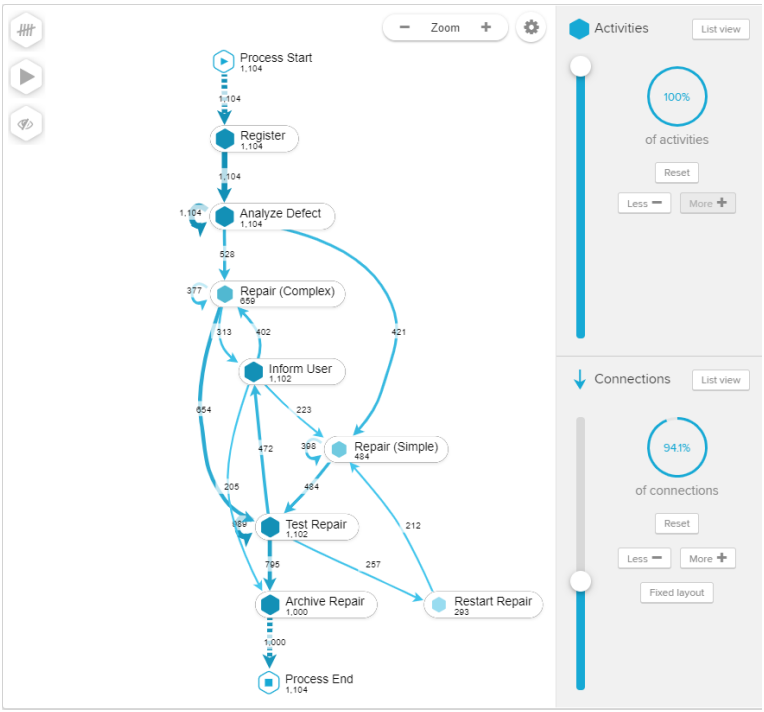
Obrázek 6.6: Model procesu reklamace mobilního telefonu, dle kterého byl vygenerován log událostí.

Na obrázku 6.6 je model procesu opravy mobilního telefonu. Na jeho základě byl vygenerován log událostí, na kterém byla porovnávána přesnost implementovaných process discovery algoritmů.



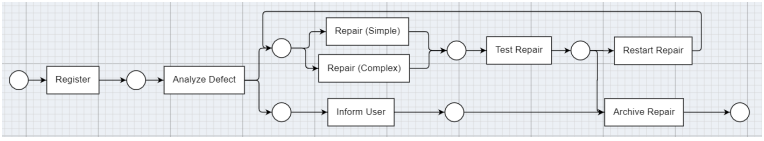
Obrázek 6.7: Model procesu opravy získaný v programu Disco

Obrázek 6.7 zobrazuje model procesu opravy získaný analýzou logu událostí `Phone repair`. Do modelu procesu doplňuje informace o době běhu jednotlivých aktivit a době prostoje mezi spuštěním dvou aktivit.



Obrázek 6.8: Model procesu opravy získaný v programu Celonis

Obrázek 6.8 zobrazuje model procesu opravy získaný analýzou logu událostí `Phone repair`. Výsledná kvalita modelu je silně závislá na nastavení atributů „Activities“ (omezuje počet zobrazených aktivit) a „Connections“ (omezuje propojení mezi aktivitami). Do modelu procesu je zde doplněna informace o počtu spuštění jednotlivých aktivit.



Obrázek 6.9: Model procesu opravy v rámci implementovaného systému.

Nyní porovnáme modely vygenerované v rámci těchto tří systémů. Všechny systémy dokázaly správně načíst informace o dostupných aktivitách. Modely vygenerované v programu Disco a Celonis obsahují menší nepřesnosti, které jsou dány použitím Fuzzy algoritmu. Na druhou stranu model vygenerovaný v implementovaném systému nejvíce odpovídá původnímu modelu.

6.5 Simulace

Cílem těchto experimentů je ověřit, jakým způsobem dokáže systém předpovídat budoucí instance procesu a na nich založené obsazení zdrojů. Veškeré testy probíhají na procesu reklamace mobilního telefonu z logu událostí `Phone repair`.

6.5.1 Základní test

Tento test generuje nové instance procesu přesně tak, jak se naučil v logu událostí. Simulace byla spuštěna s parametry:

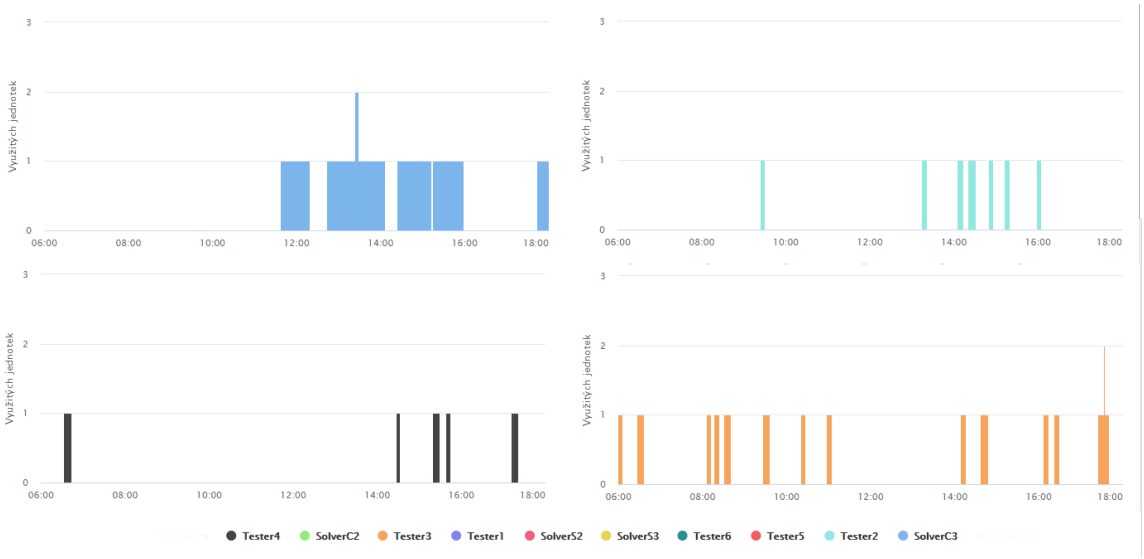
- Datum a čas první spuštěné aktivity: 2. 5. 2019 6:00:00
- Datum a čas poslední spuštěné aktivity: 2. 5. 2019 17:30:00
- Rychlost vzniku instancí procesu: 1

Datum spuštění	Datum ukončení	Průběh
02. 05. 2019 06:00:00	02. 05. 2019 06:49:00	$\langle a, b, d, e, f, h \rangle$
02. 05. 2019 06:27:17	02. 05. 2019 07:27:17	$\langle a, b, c, e, f, h \rangle$
02. 05. 2019 07:34:49	02. 05. 2019 08:29:49	$\langle a, b, d, e, f, h \rangle$
02. 05. 2019 08:06:08	02. 05. 2019 08:58:08	$\langle a, b, d, f, e, h \rangle$
02. 05. 2019 08:30:54	02. 05. 2019 09:29:54	$\langle a, b, e, d, f, h \rangle$
02. 05. 2019 09:26:41	02. 05. 2019 10:32:41	$\langle a, b, d, e, f, h \rangle$
...		
02. 05. 2019 15:01:22	02. 05. 2019 15:33:22	$\langle a, b, d, f, e, h \rangle$
02. 05. 2019 15:38:19	02. 05. 2019 16:37:19	$\langle a, b, d, f, e, h \rangle$
02. 05. 2019 15:56:04	02. 05. 2019 16:38:04	$\langle a, b, d, f, e, h \rangle$
02. 05. 2019 16:07:12	02. 05. 2019 17:22:12	$\langle a, b, d, f, g, d, e, f, h \rangle$
02. 05. 2019 17:12:00	02. 05. 2019 17:45:00	$\langle a, b, d, f, e, h \rangle$
02. 05. 2019 17:25:08	02. 05. 2019 18:45:08	$\langle a, b, d, e, f, h \rangle$

Tabulka 6.3: Instance procesů vygenerované v rámci simulace

V tabulce 6.4 je znázorněna část instancí procesu vytvořených v průběhu simulace. Celkem v průběhu simulace vzniklo 24 různých instancí využívající. Nejkratší instance trvala 11 minut a nejdelší naopak 1 hodinu a 36 minut.

V průměru se spustila nová instance procesu každých 29 minut a 47 sekund. Nejkratší rozestup mezi vznikem dvou instancí procesu bylo 11 minut a 8 sekund a naopak nejdelší rozestup mezi spuštěním dvou instancí byla 1 hodina 8 minut a 20 sekund.



Obrázek 6.10: Pohled na obsazenost určitých zdrojů dle simulace.

Na obrázku 6.10 je graf predikované alokace vybraných zdrojů využitých v rámci simulace.

6.5.2 Častější příchod nových instancí

Další experiment zkoumá, jak by se proces choval, kdyby byl spouštěn častěji.

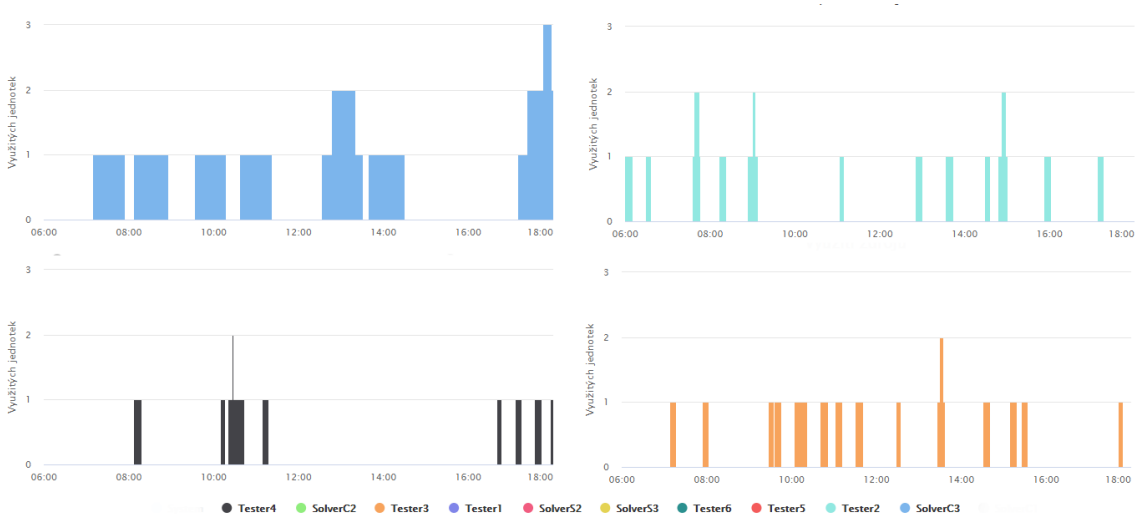
- Datum a čas první spuštěné aktivity: 2. 5. 2019 6:00:00
- Datum a čas poslední spuštěné aktivity: 2. 5. 2019 17:30:00
- Rychlost vzniku instancí procesu: 1,5

Datum spuštění	Datum ukončení	Průběh
02. 05. 2019 06:00:00	02. 05. 2019 06:41:00	$\langle A, B, D, F, E, H \rangle$
02. 05. 2019 06:09:20	02. 05. 2019 06:59:20	$\langle A, B, D, E, F, H \rangle$
02. 05. 2019 06:51:02	02. 05. 2019 08:10:02	$\langle A, B, D, E, F, H \rangle$
02. 05. 2019 07:08:35	02. 05. 2019 08:06:35	$\langle A, B, D, F, E, H \rangle$
02. 05. 2019 07:15:23	02. 05. 2019 08:22:23	$\langle A, B, C, F, E, G, C, F, H \rangle$
02. 05. 2019 07:44:47	02. 05. 2019 09:12:47	$\langle A, B, D, E, F, H \rangle$
02. 05. 2019 08:07:56	02. 05. 2019 09:09:56	$\langle A, B, D, F, E, H \rangle$
...		
02. 05. 2019 15:47:24	02. 05. 2019 16:40:24	$\langle A, B, C, F, E, H \rangle$
02. 05. 2019 15:53:11	02. 05. 2019 17:08:11	$\langle A, B, E, C, F, H \rangle$
02. 05. 2019 16:38:48	02. 05. 2019 17:48:48	$\langle A, B, E, D, F, H \rangle$
02. 05. 2019 16:41:11	02. 05. 2019 17:20:11	$\langle A, B, C, E, F, H \rangle$
02. 05. 2019 17:01:09	02. 05. 2019 18:16:09	$\langle A, B, D, E, F, H \rangle$
02. 05. 2019 17:08:43	02. 05. 2019 18:23:43	$\langle A, B, D, E, F, H \rangle$
02. 05. 2019 17:24:05	02. 05. 2019 18:52:05	$\langle A, B, D, E, F, H \rangle$

Tabulka 6.4: Instance procesů vygenerované v rámci simulace

V tabulce 6.4 je znázorněna část instancí procesu vytvořených v průběhu simulace. Celkem v průběhu simulace vzniklo 40 různých instancí využívající. Nejkratší instance trvala 11 minut a nejdelší naopak 1 hodinu a 36 minut.

V průměru se spustila nová instance procesu každých 17 minut a 22 sekund. Nejkratší rozestup mezi vznikem dvou instancí procesu bylo 25 sekund a naopak nejdelší rozestup mezi spuštěním dvou instancí bylo 45 minut a 37 sekund.



Obrázek 6.11: Pohled na obsazenost určitých zdrojů dle simulace.

Na obrázku 6.11 je graf predikované alokace vybraných zdrojů využitých v rámci této simulace. Když porovnáme tento graf s grafem na obrázku 6.10, vidíme, že zde došlo k nárůstu požadavků na zdroje. Zatímco v předchozím případě bylo v jednu chvíli požadovány maximálně 2 jednotky zdroje „SolverC3“, v této simulaci byly vyžadovány již 3 jednotky.

Kapitola 7

Závěr

Cílem diplomové práce bylo studium problematiky získání znalostí z procesních logů. K tomu byly využity metody process miningu. Byly zde popsány jeho základní kategorie a jejich cíle. Podrobně jsme se přitom zaměřili na kategorii process discovery, která je zodpovědná za sestavení modelu procesu na základě dostupného logu událostí.

Na základě teoretické části byl proveden průzkum trhu. Popsali jsme některé existující systémy. Byla navrhnuta vlastní komponenta, která řeší problematiku krátkodobého plánování podnikových procesů. K tomu využívá predikce založené na informacích získaných z logu událostí.

V rámci implementace byl vyvinut vlastní parser logu událostí, který zpracovává log událostí kratší dobu a zároveň má menší nároky na paměť než referenční knihovna OpenXES. Byly implementovány dva algoritmy, pro sestavování modelu procesu. U každého z nich byl dbán důraz za efektivitu a v průběhu implementace došlo k optimalizaci rychlosti výpočtu. Výstupy obou algoritmů pro process discovery byly porovnány s existujícími systémy.

Vytvořený systém dále umožňuje provádění simulací procesu na základě znalosti jeho modelu a statistických údajů o jeho průběhu. Díky tomu můžeme systém využít například jako podpůrný nástroj ke krátkodobému plánování.

Systém by bylo vhodné rozšířit o další algoritmy process discovery (jako je např. Fuzzy miner), které umožní získat pohledy na model dle požadované úrovně abstrakce. Díky tomu bude umožněn lepší vhled do méně strukturovaných procesů. Důležitou částí práce je simulace a na ní založené zobrazení informací o obsazenosti zdroje. Simulace v současné chvíli uvažuje, že máme každý zdroj v neomezené kapacitě. Do systému by bylo vhodné možnost rozlišení, které zdroje mají omezenou kapacitu. Při simulaci by poté byl brán ohled na tuto kapacitu a bylo by možné simulovat zpoždění. Kromě omezené kapacity zdrojů můžeme simulovat i jejich omezenou dostupnost v čase.

Literatura

[1] Say Hello to Disco! - Flux Capacitor. May 2012 (accessed March 3, 2019).
URL https://fluxicon.com/blog/2012/05/say-hello-to-disco/

[2] IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std 1849-2016*, Nov 2016: s. 1–50, doi:10.1109/IEEESTD.2016.7740858.

[3] van der Aalst, W. M.: *Process Mining in the Large: A Tutorial*. Cham: Springer International Publishing, 2014, ISBN 978-3-319-05461-2, s. 33–76, doi:10.1007/978-3-319-05461-2_2.
URL https://doi.org/10.1007/978-3-319-05461-2_2

[4] Aalst, W. M. P.: The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, ročník 8, 02 1998: s. 21–66, doi:10.1142/S0218126698000043.

[5] van der Aalst, W. M. P.: *Process Mining: Data Science in Action*. Heidelberg: Springer, druhé vydání, 2016, ISBN 978-3-662-49850-7, doi:10.1007/978-3-662-49851-4.

[6] van der Aalst, W. M. P.; van Hee, K. M.; ter Hofstede, A. H. M.; aj.: Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, ročník 23, č. 3, May 2011: s. 333–363, ISSN 1433-299X, doi:10.1007/s00165-010-0161-4.
URL https://doi.org/10.1007/s00165-010-0161-4

[7] Aalst, W. M. P.; Karla A. de Medeiros, A.; Weijters, A.: Genetic Process Mining. 06 2005, s. 48–69, doi:10.1007/11494744_5.

[8] van der Aalst, W. M. P.; de Medeiros, A. K. A.; Weijters, A. J. M. M.: Genetic Process Mining. In *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, 2005, s. 48–69, doi:10.1007/11494744_5.
URL https://doi.org/10.1007/11494744_5

[9] Aalst, W. v. d.; Hee, K. M. v.: *Workflow management: models, methods, and systems*. MIT Press, 2004.

[10] Aalst, van der, W.; Dongen, van, B.; Günther, C.; aj.: ProM : the process mining toolkit. In *Proceedings of the BPM 2009 Demonstration Track (BPMDemos 2009, Ulm, Germany, September 8, 2009)*, editace A. Alves de Medeiros; B. Weber, CEUR Workshop Proceedings, CEUR-WS.org, 2009, s. 1–4.

[11] F. van Dongen, B.; Karla A. de Medeiros, A.; M. W. Verbeek, H.; aj.: The ProM Framework: A New Era in Process Mining Tool Support. In *Applications and Theory of Petri Nets 2005*, ročník 3536, 06 2005, s. 444–454, doi:10.1007/11494744_25.

[12] van Dongen, B. F.; van der Aalst, W. M. P.: A Meta Model for Process Mining Data. In *EMOI-INTEROP*, 2005.

[13] Günther, C.: *Process mining in flexible environments*. Dizertační práce, Department of Industrial Engineering & Innovation Sciences, 2009, doi:10.6100/IR644335, proefschrift.

[14] Günther, C. W.; van der Aalst, W. M. P.: A Generic Import Framework for Process Event Logs. In *Business Process Management Workshops*, editace J. Eder; S. Dustdar, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ISBN 978-3-540-38445-8, s. 81–92.

[15] K Alves De Medeiros, A.; Weijters, A.; Aalst, W. M. P.: Using genetic algorithms to mine process models: representation, operators and results. *Systems Research and Behavioral Science - SYST RES BEHAV SCI*, 01 2005.

[16] Leemans, S. J. J.; Fahland, D.; van der Aalst, W. M. P.: Scalable process discovery and conformance checking. *Software & Systems Modeling*, ročník 17, č. 2, May 2018: s. 599–631, ISSN 1619-1374, doi:10.1007/s10270-016-0545-x.
URL https://doi.org/10.1007/s10270-016-0545-x

[17] de Medeiros, A. K. A.; van Dongen, B. F.; van der Aalst, W. M. P.; aj.: Process mining: Extending the α-algorithm to mine short loops. In *Eindhoven University of Technology, Eindhoven*, 2004.

[18] de Medeiros, A. K. A.; van Dongen, B. F.; van der Aalst, W. M. P.; aj.: Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm. In *Ubiquitous Mobile Information and Collaboration Systems*, editace L. Baresi; S. Dustdar; H. C. Gall; M. Matera, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ISBN 978-3-540-30188-2, s. 151–165.

[19] Pospíšil, M.; Mates, V.; Hruška, T.; aj.: Process Mining in a Manufacturing Company for Predictions and Planning. *International Journal on Advances in Software*, ročník 2013, č. 3, 2013: s. 283–297, ISSN 1942-2628.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=10559

[20] Rozenberg, G.; Engelfriet, J.: *Elementary net systems*. 04 2006, s. 12–121, doi:10.1007/3-540-65306-6_14.

[21] Russell, N.; ter Hofstede, A.; van der Aalst, W.; aj.: Workflow Control-Flow Patterns: A Revised View. Technická Zpráva BPM-06-22, BPM Center, 2006.
URL http://bpmcenter.org/wp-content/uploads/reports/2006/BPM-06-22.pdf

[22] Weijters, A.; Aalst, van der, W.: Rediscovering workflow models from event-based data using Little Thumb. *Integrated Computer-Aided Engineering*, ročník 10, č. 2, 2003: s. 151–162, ISSN 1069-2509.

[23] Weijters, A. J. M. M.; Ribeiro, J. T. S.: Flexible Heuristics Miner (FHM). In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, April 2011, s. 310–317, doi:10.1109/CIDM.2011.5949453.

[24] Wen, L.; Aalst, W. M. P.; Wang, J.; aj.: Mining process models with non-free-choice Constructs. *Data Min. Knowl. Discov.*, ročník 15, 10 2007: s. 145–180, doi:10.1007/s10618-007-0065-y.

[25] Wen, L.; Wang, J.; Aalst, W. M. P.; aj.: Mining Process Models with Prime Invisible Tasks. *Data & Knowledge Engineering*, ročník 69, 06 2010: s. 999–1021, doi:10.1016/j.datak.2010.06.001.

Příloha A

Obsah přiloženého paměťového média

- **event_logs**: ukázky logů událostí
- **src/client**: zdrojové soubory klientské části systému
- **src/server**: zdrojové soubory serverové části systému
- **src/thesis**: zdrojové soubory této dokumentace
- **readme.txt**: informace pro spuštění projektu
- **DP.pdf**: dokumentace diplomové práce