Trendy v modelování workflow procesů

Kristína Prochocká

Fakulta informačních technologií VUT v Brně, Božetěchova 2, Brno, Česká Republika

{iprochocka}@fit.vutbr.cz

Abstrakt. Tento článek shrnuje některé trendy v modelování workflow procesů. Zaměřuje se na grafové modely (BPMN, EPC), implementace procesů v programovacím jazyce za využití objektového programování, ale také na procesy s omezením. Cílem je zhodnotit a porovnat jednotlivé modely mezi sebou a dále jak z hlediska modelovací síly, tak z hlediska jiných vlastností – z hlediska přehlednosti a například možností změn za běhu.

Klíčová slova: Modelování procesů, workflow, BPMN, Event Process Chains, Procesy s omezením

1 Úvod

Systémy pro řízení procesů nejsou novým tématem, přesto v nich stále probíhá aktivní výzkum. Je to dáno tím, že správně namodelované a zoptimalizované procesy, přehledy a analýzy jsou pro podniky důležité, pokud chtějí uspět v náročné globální konkurenci.

Workflow je navíc velmi široká oblast, ve které se řeší spousta věcí, která spolu někdy ani plně nesouvisí. Workflow zasahuje do modelování procesů, vizualizace, databází, distribuovaných systémů, teorií managementu, analýz a dataminingu, simulací, různých optimalizací, umělé inteligence a v neposlední řadě do programovacích technik.

Tato práce bude zaměřena na modelování procesů, které je samo o sobě rozsáhlým tématem. Existuje zde spousta modelovacích přístupů, zde se zaměříme pouze na některé z nich. Práce je členěna následovně. V úvodu budou vysvětleny základní principy workflow. Základní přehled bude velmi stručný, pro podrobnější seznámení je možné využít například literaturu od Van Der Aalsta [1, 7, 8]. V další části budou popsány základní modelovací přístupy – grafové, programové a modelování pomocí pravidel a omezení. Tyto přístupy pak budou prodiskutovány z hlediska různých kritérií a nároků na WF systém. Cílem práce bude srovnání těchto přístupů a nastínit možnosti jejich různých kombinací a společných, či rozdílných prvků.

2 Prvky workflow systému

Workflow systém se skládá z několika základních prvků. Procesy mohou být ve formě petriho sítě, nebo i jiných modelů. Některé workflow systémy umožňují volné vykonávání pořadí úloh za definovaných omezení, nebo jsou zapsané přímo v programovacím jazyce.

Každý proces také potřebuje data, na jejichž základě se rozhoduje. Ty mohou být rozličného charakteru. Může se jednat o data procesních definic, data samotného workflow engine, data patřící procesní instanci, sdílená data, externí aplikační data apod. Daty se tato práce nezabývá.

Zdrojem může být myšlen člověk, který vykonává úlohu, nebo nějaký stroj, materiál, místnost. Zdroji se tato práce taktéž nezabývá.

2.1 Výhody workflow systému

Programátorský a vývojářský pohled.

Workflow jako framework.

Workflow tvoří framework pro tvorbu obchodních procesů. Obsahuje základní funkcionality, které by se jinak musely dělat ručně. Jedná se především o procesní jádro, které zajišťuje vykonávání procesů - například interpret procesního modelu, uživatelské rozhraní pro procesní model, nebo pro tvorbu formulářů a definici dat. Workflow dále může obsahovat nástroje pro validaci modelu, analýzu apod.

Rapid aplication developement.

Workflow je deklarativní, má často nástroje pro rychlou tvorbu formulářů a procesů. Umožňuje také pochopitelně formuláře programovat libovolným způsobem. Tento bod souvisí s předchozím bodem. Některé typy workflow systémů, jako je třeba Bizagi, umožňují velkou část systému nastavit v uživatelském rozhraní a prodávají se jako celý programový balík (tj. zákazník si ho může koupit a přímo používat), jiné (například Windows Workflow Foundation), jsou zase více sada knihoven, ze kterých je možné postavit workflow systém zákazníkovi více na míru.

Analýza a vývoj systému.

Díky formalizovaným procesům mimo kód (tj. uložených v nějaké deklarativní formě) můžeme tyto procesy vytvářet takovým způsobem, že máme funkční jednotky a zároveň proces ostatní pochopí díky vizualizaci procesního modelu i neprogramátor (např. formou petriho sítě). Zároveň se to hodí jako dokumentace, protože systém eviduje i změny procesů a verze.

Další oblast analýzy je získávání znalostí z procesů (Process Mining či Business Process Intelligence). Workflow systémy často mají nástroje pro reporting, analýzu výkonnosti a odhalování úzkých míst. Komerční systémy zatím často pracují s jednoduchou statistikou, ale mohou mít komunikační můstky se specializovanými programy na analýzu procesů, jako je třeba PROM či DISCO [12].

Možnost přerušení běhu a pokračování.

Protože je workflow uloženo ve formě dat a ne jako programový kód (pochopitelně až na atomické jednotky vykonávání, které jsou psané běžným kódem), může být workflow pozastaveno, uloženo (třeba do databáze) a poté znova spuštěno. Toto je důležitá vlastnost workflow systému, protože většina běžících procesů běží třeba několik týdnů, mezitím dochází k restartování serveru a celkově není možné držet všechny běžící procesy v paměti, takže workflow systém musí umět stav procesu serializovat do paměti a běžící proces z ní zase spustit. Samozřejmě nelze proces uložit ve všech bodech - musí být doběhnutá transakce vykonávání procesu - tj. nelze uložit například stav jednotlivé úlohy, kdy běží nějaká jiná transakce. Ale je možné proces odložit do paměti v případě vykonávání nějaké asynchronní úlohy (například webové služby). Pokud poté dojde k vrácení výsledku (nebo aktivní poslání zprávy webové služby procesu), proces je obnoven z paměti a požadavek je zpracován.

Možnost změnit proces za běhu (či některé instance).

Díky tomu, že je workflow definováno mimo kód (a je vlastně interpretováno WF systémem), tak je možné měnit provádění za běhu. To může být užitečné pro adaptabilní workflow. Můžeme například zastavit systém a změnit mu logiku a pak zase spustit. Toto je užitečné v několika případech.

Za prvé tak můžeme opravit chybu v již rozběhnutém procesu. Za druhé můžeme změnit logiku konkrétní procesní instance v případě, že se jedná o nějakou situaci, na kterou nebyl proces namodelován. Dále může dojít ke změně požadavků na celý procesní model a je nutné změnit definici všech běžících procesů. Poslední využití je v dynamických procesech, kdy procesní model sestavujeme dynamicky za běhu - nemusíme tak mít připraveny a namodelovány dopředu všechny možnosti, ale může existovat skript, který na základě dat a stavu procesu rozhoduje o tom, která část procesu se dynamicky změní - takovéto změny však mohou být náročné na údržbu správnosti procesu, takže je dobré je používat například na dynamické volání podprocesu, kdy si až v době běhu rozhodneme, který podproces spustit - v běžném programátorském jazyce by tomu odpovídala pozdní vazba při volání podprogramu (například v. NET pomocí reflexe).

Využití v nepodnikatelské oblasti.

Workflow má využití u složitých výpočetních systémů – například u úloh umělé inteligence a dataminingu. Zde běží nějaký kód, jehož hlavní logika je řízena workflow, může být pozastaven, pozměněn dle potřeby a zase pokračovat. Workflow

se pochopitelně nepoužije na celou logiku kódu, protože je pomalé, ale může volat atomické výpočty asynchronně a sbírat výsledky.

Workflow systém je využit například v systému RapidMiner, kde jednotlivé úlohy (Tasky) představují nějaké operace (třeba neuronovou síť, rozhodovací strom, předzpracování, čištění dat apod.). Mezi úlohami tečou data mezi metodami. Je tím pádem možné namodelovat přesné workflow dolovací úlohy a během dolování měnit proces za běhu, ukládat jeho stav se všemi daty.

Manažerský pohled.

Formalizovaný podnikový proces.

Díky tomu, že je podnikový proces formalizován, je udržován ve společnosti větší pořádek. Snižuje se riziko zapomenutí na nějaký krok a v případě odejití klíčových zaměstnanců, kteří jediní znají část procesu, je tato znalost stále k dispozici. Další důležitá vlastnost je ta, že podnik má své definice centralizované, což se vždy hodí.

To umožňuje proces například vyměňovat mezi organizacemi, nebo části podniku (například mezi pobočkami) a část procesů sdílet. Díky pokročilejší vizualizaci, kdy můžeme části procesů nezobrazit, je model přehledný pro všechny, kdo by s ním mohli pracovat. Například můžeme zobrazit pouze části, které jsou relevantní pro management, tučně vyznačit hlavní část procesu, skrýt nějaké nepotřebné části, barevně oddělit hlavní část procesu, vedlejší a část, kde se odchytávají výjimky.

Pokud je workflow správně navrženo a používáno, zvyšuje efektivitu práce. V administrativě se pak mohou používat elektronické dokumenty místo papírových a některé operace tak mohou probíhat paralelně.

Monitorování a Analýza.

Asi nejdůležitější manažerská výhoda. Díky softwarové formalizované kontrole nad procesem zde máme okamžitý přehled o tom, na jakých úkolech se pracuje, jací lidé na nich pracují a v jakém jsou stavu.

Pokročilejším monitorováním je predikce. Pomocí predikce můžeme zavčas detekovat výskyt nežádoucích událostí a stavů. Může jít například o hlášení, že daná úloha se již pravděpodobně nestihne. Predikce mohou být triviální (základní statistika), nebo může jít o pokročilé metody dolování dat [12].

S monitorováním souvisí analýza. Analýza se od monitorování liší v tom, že pracuje nad historickými daty (zatímco monitorování nad aktuálními daty). Pomocí analýzy můžeme detekovat, které problémy vznikají a kde. Analýza s monitorováním však souvisí, protože inteligentnější monitorovací systémy využívají taktéž analýzu - fáze analýzy je pak učení systému a fáze monitorování je predikce naučeného systému.

Řízení a Optimalizace.

Je-li proces dostatečně opakovatelný a predikovatelný (například výroba), můžeme ho pomocí těchto systémů i řídit – procesy mohou být plánovány s ohledem na zdroje a materiál. Toto však mnohdy nebývá součástí workflow pro administrativu. Na základě historických dat je však možné například kontrolovat některá rozhodnutí, pokud neodpovídají historické zkušenosti, je něco možná špatně.

Řízení souvisí s analýzou a monitorováním. Analýza nám může pomoci zjistit, která místa jsme v historii špatně řídili a můžeme se tím do budoucna poučit - například analýza úzkých míst zjistí kde a za jakých podmínek nastávají problémy, což může vést ke změně řízení, nebo přímo k optimalizaci procesu (zakoupení lepších strojů, změna umístění strojů a tak dále).

3 Přístupy k modelování procesů

3.1 Grafové modely toku

Grafové modely toku (dále jen grafové modely) využívají grafického popisu procesu. U těchto grafových modelů však platí, že se drží paradigmatu imperativního programování - tj. toku přesně definovaných příkazů s řízením v rozhodovacích bodech. Může se jednat o vývojový diagram, Petriho Síť, BPMN, či Event Process Chains, případně jiné modely, jako například Yawl. Přestože různé workflow systémy mají různé přístupy k modelování procesů, existuje práce, která se snaží zobecnit jednotlivé prvky a popsat, které systémy je obsahují [2]. Práce ohledně workflow patterns obsahuje poměrně rozsáhlou analýzu různých workflow systémů a jejich modelů, jednotlivé prvky poté zobecňuje. V tuto chvíli práce obsahuje přibližně 70 typů řídících prvků. V práci jsou popsané funkcionality prvků a také informace o tom, které systémy které prvky implementují.

Velké množství různých možných řídících prvků je jistý ukazatel problémů ve workflow systémech. Vzhledem ke snaze vytvořit procesní systém, který by si uživatel mohl vytvořit pomocí uživatelského rozhranní, je zde nutnost velkého množství prvků, což není zcela systémové řešení. Vždycky totiž může dojít k požadavku na speciální proces, který nepůjde rozumným způsobem namodelovat a výsledný proces bude vypadat velmi složitě, přičemž by stačil nový modelovací prvek, který v systému ovšem není k dispozici.

3.2 Workflow jako program

Jiná oblast výzkumu se snaží využít současných programovacích jazyků pro přímý popis procesu. Grafové modely mají jistě své výhody (bude prodiskutováno později), ale jedná se vlastně o interpret. Jiná možnost je využití běžného jazyka pro popis procesu a využít tak všechny jeho vlastnosti (například dědičnost a polymorfismus.). Tento přístup se dá řešit dvěma základními způsoby. Jeden využívá běžný programovací jazyk [3, 4], druhý speciální programovací jazyk, který umí uložit stav vykonávání sám o sobě – například Smalltallk [5].

Workflow ve formě programu umožňuje překonat problematiku popsanou v kapitole o grafových modelech. Pokud nastane situace, která by nešla u grafového modelu namodelovat, obvykle nebude problém to vytvořit pomocí vhodných komponent v programovacím jazyce. Nevýhoda je zřejmá - chybí explicitní procesní model, takže validace modelu, vizualizace a další věci zde nejsou k dispozici.

3.3 Omezení

Omezení jsou další možností, jak popisovat proces. Základní myšlenka omezení je taková, že vše je povoleno, dokud není řečeno jinak. To je zcela opačný přístup, než v předchozích přístupech, kde je povoleno jen to, co je součástí toku programu. Omezení jsou vhodná zejména do flexibilních procesů. Modelování pomocí omezení jsou popsána například v [6,9,10].

Nevýhoda systému s omezením je zvýšená složitost. U imperativních přístupů (graf toku a program) jsou jasně dané postupy a máme přehled o tom, jaké kombinace stavů nastávají. U systémů s omezením to vidět není, proto může docházet k problémům, kdy nastane kritická situace, která nastat neměla, protože tvůrce procesu na danou situaci zapomněl.

Přestože výše uvedené typy modelování jsme popsali jako grafické, systémy s omezeními také mohou být popsané graficky (viz. Například Declare [9, 10]).

3.4 Shrnutí

Každý z těchto přístupů má své výhody a nevýhody. V této práci budou prodiskutovány jednotlivé přístupy z hlediska vhodnosti modelování, výkonnosti, srozumitelnosti pro řízení podniku, vizualizace, simulace, ale také schopnosti adaptace na změny, což je problém, který je asi jeden z nejkomplikovanějších v celé oblasti workflow. Dále budou prodiskutovány možnosti kombinací jednotlivých přístupů.

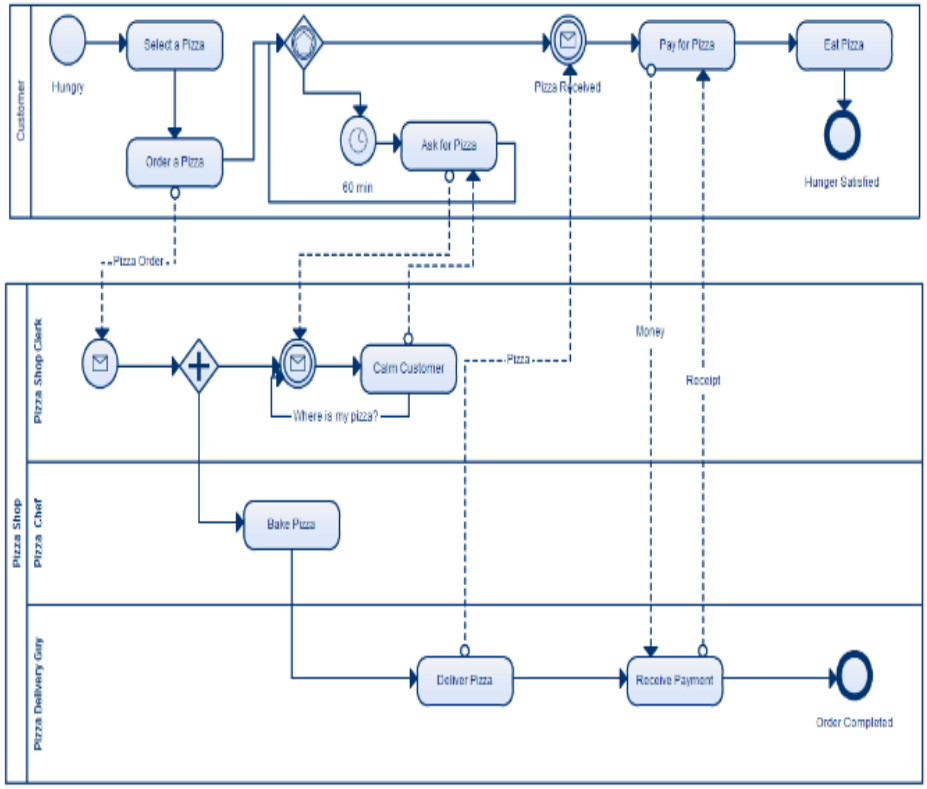
4 Grafové modely

Nejlepší bude začít příkladem. Na obrázku 1 vidíme ukázkový proces objednání Pizzy namodelovaný v BPMN notaci. Vidíme zde spousty různých prvků. V části zákazníka je nejprve Select_Pizza a pak Order_Pizza, ta posílá navíc signál tasku Pizza_Received a povolí pak tomuto tasku pokračovat. Proces se pak dělí do zákaznické a firemní části. Na straně zákazníka pokračuje do Event Or Routingu a čeká na události. Jedna z událostí je signál tasku Pizza_Received, která čekání ukončí (pokud je čekání ještě aktivní, pokud čekání není aktivní tak čeká, než aktivní zase bude).

Je tam také napojený časovač, který když dojde, tak přeruší čekání na Pizzu a přesune se do Tasku Ask For Pizza, ten pošle zprávu firmě a čeká na odpověď

(uklidnění, že pizza se dělá). Po přijetí signálu může pokračovat v čekání na pizzu (časovač se znovu aktivuje) a pokud přijde Pizza Received, tak je čekání u konce.

Na straně firmy je AND-Split, který proces zase rozdělí. V jedné části se čeká na zprávu od Ask_For_Pizza a pak se přechází do Calm_Customer, kdy se pošle zpátky zpráva (ujištění, že se na pizze dělá). Další cesta je do Tasku Bake_Pizza, který pokračuje do Deliver_Pizza a ta posílá zprávu tasku Pizza_Received. Dále si vyměňují zprávy Tasky Pay_For_Pizza a Receive_Payment. Pay_For_Pizza pošle signál a sám na signál čeká. Receive_Payment nejprve čeká na signál a pak posílá signál, že se zaplatilo.



Obr. 1. BPMN proces objednání pizzy. Převzato z http://www.bpmn.info/tutorials/bpmn-order-process-for-pizza/

Přístupů k modelování procesů pomocí grafových toků je více (např. Event Process Chains [1]), ale víceméně jsou všechny podobné. Přestože je tok programu popsán grafem, jedná se stále o imperativní programování, byť popsané graficky. Grafový model procesu je pak v systémech uložen do relační databáze, nebo do dokumentu XML, či JSON (a například do dokumentové databáze). Jádro vykonání procesu poté proces interpretuje a vykonává. Stav vykonání procesu je rovněž uložen v databázi. Je tedy možné kdykoliv proces natáhnout z úložiště do paměti a podle definice interpretovat (spustit) a poté zase uložit.

Samozřejmě, že grafem není popsáno 100% procesu, všechny tyto workflow systémy umožňují vykonávání skriptů, nebo kódů v daném jazyce. Rozdíl je v tom, že kódy jsou psané přímo v jazyce, ve kterém je workflow vytvořeno a pokud je tento kód kompilovaný (například C++, Java, .NET apod.), potom změna kódu znamená překompilování a nasazení celého workflow systému, což pro menší kódy není žádoucí. Většina workflow systémů implicitně proto podporuje i skripty (psané obvykle v JavaScriptu), které se interpretují za běhu a je možné pomocí nich modelovat menší Business pravidla, jako například rozhodovací kódy v OR routingu.

4.1 Příklady grafových modelů

Petriho sítě:

Petriho sítě jsou model, který má silný matematický základ. Díky tomu je možné Petriho sítě verifikovat a existuje mnoho prací, které se zabývají transformacemi ostatních modelů právě do petriho sítí [1]. Mimo akademii však Petriho sítě nejsou příliš používané, protože jejich schopnost popsat podnikový proces je poměrně omezená - ne ve smyslu vyčíslitelnosti, ta je silná, ale spíše ve smyslu vhodného modelování - i relativně jednoduchý podnikový proces může být těžce popsatelný Petriho sítí. Jejich hlavní výhoda je tedy v silném matematickém popisu a sadě nástrojů, které umí petriho sítě simulovat a verifikovat. Proto se často využívá právě transformací - workflow systém může umožňovat transformovat vlastní model do Petriho sítě a tu potom verifikovat. Jsou s tím samozřejmě problémy, protože se nemusí podařit převést funkcionalitu přesně, zejména v situaci, kdy proces obsahuje i řídící kódy.

BPMN

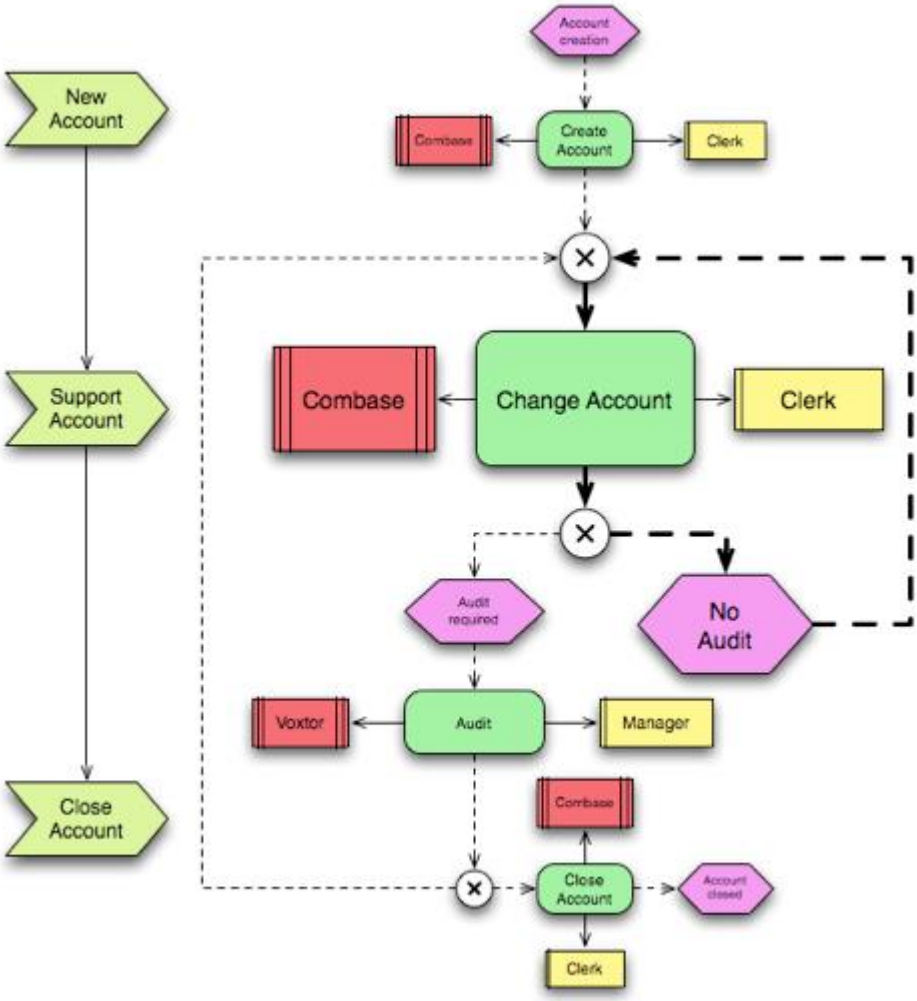
BPMN je v tuto chvíli asi nejpoužívanější způsob modelování procesu [8, 11]. Jeho vlastností je velké množství modelovacích prvků (časovače, signály, …). Nevýhodou může být nejednoznačná sémantika některých prvků. BPMN je používáno zejména jako standardizace popisu procesů v rámci společností, přestože popis není 100% jednoznačný. BPMN vzniklo v roce 2005.

Event Process Chains.

Tento způsob je starší [15], vznikl v roce 1990 na University of Saarland a tvůrcem byl August Wilhelm Sheer. Je používán ve známém procesním prostředí ARIS.

Procesní model se zásadně neliší od BPMN, má méně modelovacích prků a díky tomu, že se střídají funkční bloky a bloky aktivit, tak je méně kompaktní, než BPMN – obrázek 2. Někteří také pokládají BPMN za celkově čitelnější a přehlednější.

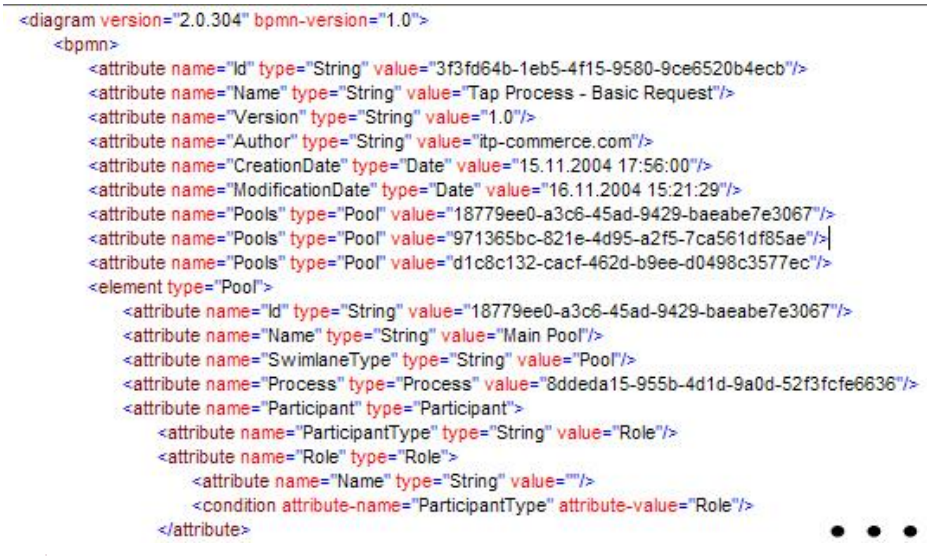
Jak vidíme, Event Process Chains se zásadně neliší od jiných jazyků pro popis – můžeme vidět úlohy a rozdělující uzly (XOR-SPLIT, XOR-JOIN), podpora je i pro AND – paralelní vykonávání.



Obr. 2. Event Process Chains, převzato z: http://frapu.de/blog/images/epc-example.png

XPDL.

XPDL je formátem zapsaným v XML pro přenos definic Business Procesů. V tuto chvíli se asi nejčastěji používá jako serializace BPMN standardu. Ukázka XPDL je na obrázku 3.



Obr. 3. ukázka XPDL, převzato z: http://www.itp-commerce.com/wp-content/uploads/2013/01/img-xml.jpg

BPEL.

Celý název zní Web Services Business Process Execution language (WS-BPEL), zkráceně BPEL.

Je to jazyk pro popis orchestrace procesů prostřednictvím webových služeb. Své místo má tedy zejména při spojování uzlů ve webových službách. Tyto uzly mohou být spojené podobným způsobem jako Tasky v BPMN – pomocí sekvenčních směrování, AND, OR apod. Rozdíl je zejména v tom, že u webových služeb je vše distribuované, jednotlivé Tasky jsou výpočetní uzly, které mohou běžet kdekoliv na světě.

Existují práce, které převádějí BPMN na BPEL. Takováto transformace není triviální, protože každý jazyk je jiný. Výsledný BPEL proces nebývá příliš čitelný, při implementaci transformací se ukázaly některé zásadní rozdíly a také nejednoznačnosti v BPMN. Navíc je téměř nemožné udržet poté synchronizaci modelu, kdy jedna modifikace v jednom modelu je propagována do druhého.

4.2 Shrnutí - Výhody

Snadné modelování.

Vytvářet si ručně vlastní procesy je poměrně důležitá vlastnost workflow systému. Zkušený uživatel tak dokáže vytvořit jednodušší proces sám. Některé systémy, jako například Bizagi [11], obsahují ještě navíc spousty 'klikacích oken' pro zadávání pravidel (například podmínky v OR routingu) apod. Je tak možné napsat některé procesy zcela bez nutnosti napsat jediný kód programového řádku. Tyto systémy totiž zároveň umožňují namodelovat graficky také data (pomocí ER diagramů) a vytvořit formuláře pro zadávání dat a vizualizaci tabulek. Díky tomu, že dodržíme určitý modelovací standard, máme navíc k dispozici validátory a verifikace, které proces ověří na živost a přítomnost dead-locků [1].

Pokud něco nejde udělat pomocí uživatelského rozhraní, je možné dopsat kódy a skripty. Ale základní procesní logika je popsána graficky. Kódy slouží zejména pro atomické operace (například Script task dle BPMN), které provádí nějaké databázové operace, pravidla (pro OR-split) apod.

Monitorování a uživatelský přehled nad procesem.

Vizuální popis není užitečný jen pro návrháře procesu, ale také pro management (a případně jiné uživatele), kteří snadno vidí posloupnost jednotlivých kroků. Tím získává management a uživatelé lepší přehled nad tím co se děje, je tak možný efektivnější monitoring, případně vizuální simulace procesu, kdy můžeme přehrát historii a uživatel přesně vidí co se kde dělo.

4.3 Nevýhody

Flexibilní proces.

Není dost dobře možné zadefinovat flexibilní proces a případné omezující podmínky. Představme si například tento proces:

- Máme 5 tasků – A, B, C, D, E. Pořadí vykonávání Tasků je libovolné, ale:
- B nesmí být vykonávání společně s C
- D nesmí být vykonáno dříve, než E za předpokladu, že datová proměnná X obsahuje True
- A musí být první vykonaná úloha, pokud X obsahuje False
- C může začít až po tom, co je vykonáno E, ale musí být mezi nimi rozdíl alespoň 4 dny

Pokud bychom chtěli tento proces namodelovat například v BPMN, zapotili bychom se. V zásadě bychom museli namodelovat všechny různé možnosti kombinací vykonání procesů, což může být velmi obtížné.

Nezvyklé pro programátory.

Pokud mají proces vytvářet programátoři, nemusí být zcela nakloněni grafickému 'klikání' procesu. Navíc při tvorbě procesu a kódu mají dva různé verzovací systémy. Verzování procesu si řeší systém sám, zatímco kódy mohou být řešeny v systémech jako je například GIT, což může způsobovat chaos.

5 Workflow v programovacím jazyce

Další slibnou cestou je modelování workflow v programovacím jazyce. Zde je však několik úskalí:

- Workflow musí umožňovat se vypnout, uložit a poté zase načíst a pokračovat ve vykonání tam, kde zůstalo (samozřejmě až na některé atomické úlohy, které například pracují s databází). Pokračování vykonávání může navíc nastat potenciálně na jiném stroji.
- Workflow musí umožňovat změny za běhu a ukládání jednotlivých verzí procesu – některé procesy běží na staré definici, jiné na nové definici a při změně může být požadavek staré procesy migrovat na novou verzi.

Naprostá většina programovacích jazyků tuto podmínku nesplňuje. Ale než se začneme zabývat, jak toto vyřešit, podívejme se na jazyk Smalltallk.

5.1 Smalltalk

Smalltallk je jazyk, který má unikátní schopnost Persistence [5]. Celý programový kód a všechny třídy jsou vlastně samy objekty a jsou uloženy v datových strukturách. Tyto datové struktury jsou poté při ukončení programu uloženy do paměti a při opětovném spuštění může program začít přesně tam, kde skončil (opět bude samozřejmě problém s atomickými operacemi, jako jsou databázové transakce apod.).

Tato vlastnost činí smalltallk vhodný například pro experimentální dlouhotrvající výpočty, kdy můžeme měnit algoritmy, či aspoň jejich parametry, aniž bychom museli nějak řešit ukládání stavu, protože Smalltallk to řeší za nás. Pro modelování workflow je smalltalk taktéž užitečný, protože v podstatě nemusí řešit persistenci stavu procesu. Horší je to pochopitelně se změnami za běhu, ale ty nejsou snadné ani u jednoho modelovacího přístupu.

Velká nevýhoda smalltallku je jeho malá rozšířenost, těžko bude možné zákazníkovi nabízet systém pro řízení celého podniku založený na jazyku, který zná málo lidí a není pro něj dost knihoven. Další nevýhoda je bezpečnost ukládání stavu celého programu. U klasického workflow systému je úložiště relační databáze, což jsou vyspělé systémy podporující spolehlivost proti výpadku a chybě při provádění transakce, ovšem totéž nelze očekávat od smalltallku. V případě poškození souboru se stavem procesu nemusí jít spustit vůbec nic, zatímco v případě poškození stavu jedné procesní instance u běžného workflow systému lze očekávat problém jen u této

konkrétní instance. Dá se očekávat, že smalltallk se toto snaží řešit, ovšem těžko to může konkurovat spolehlivosti vyspělých relačních databází.

Smalltallk není hlavním tématem tohoto článku, je zde uveden pouze pro úplnost.

5.2 Běžné programovací jazyky

Abychom mohli podobnou funkcionalitu požadovat po programovacím jazyku, je třeba to dodělat explicitně.

První možností může být **stavové programování**. Každý proces se skládá z úloh a ty jsou v různém stavu. Při každém spuštění by se zkontrolovaly všechny úlohy, a pokud by se měly vykonat, tak by se vykonaly. Řídící pravidla by pak určovala, které úlohy smí běžet a kdy. V podstatě se tento přístup myšlenkově velmi blíží grafickým modelům, jako je např. BPMN, ale bez vizualizace. Tento přístup však určitě není vhodný, protože tím ztrácíme výhody programovacího jazyka, jako využití základních řídících konstruktů (if, look apod.) a také dědičnosti či polymorfizmu [3,4].

Druhou možností může být **opakované vykonání procesu** [3, 4]. Tento princip je založen na myšlence, že proces máme napsaný v běžném programu (samozřejmě za dodržení určitých podmínek, které jsou dané workflow frameworkem). Proces se spustí a je vykonáván jako běžný program, vstupy do volání funkcí jsou však zaznamenané. Jakmile proces skončí, máme v paměti stopu o vykonávání úloh. Při opětovném spuštění pak spouštíme program znova a sledujeme historii vykonání a podle toho volíme cestu procesem, těla funkcí, která již byla vykonání, se nevykonávají znova – autor používá pojem 'Tiché vykonání'.

5.3 Výhody

Programovací jazyk.

Můžeme využít všechny konstrukty programovacího jazyka (if, loop, dědičnost). Paralelismus je poté implementován například pomocí vláken. Také vývoj takového workflow systému bude jistě levnější, než implementace interpreteru grafového modelu včetně process designéru. A pro programátory bude intuitivnější vyvíjet procesy takovýmto způsobem.

Kooperace při vývoji je řešena pomocí systémů pro správu verzí, jako je Git. Pravda je, že grafové modely jsou také popsány pomocí jazyků (XML či JSON), takže bychom mohli stejnou operaci provést i tady. To ovšem nedokáže běžný uživatel, který je zvyklý na uživatelské rozhraní.

5.4 Nevýhody

Paralelismus, časovače, zprávy a synchronizace.

Tento přístup je určitě vhodný primárně pro procesy, které nepotřebují příliš komunikovat pomocí signálů a zpráv a nepoužívají paralelismus. Pokud máme například proces jako je Pizza Process (obrázek 1), implementovat takovýto proces v běžném programovacím jazyce nemusí být snazší, než ho implementovat pomocí BPMN. Vše samozřejmě bude záležet na kvalitě workflow frameworku, který toto zajišťuje. Je snadné se ztratit v meziprocesní komunikaci a synchronizaci, vyžaduje to zkušeného programátora, zatímco podobně složitý systém není zase tak těžké vytvořit pomocí procesního modeleru.

Vizualizace.

V tomto případě chybí zcela vizualizace procesu, což nemusí být akceptovatelné managementem. Je samozřejmě možné udělat tento model v rámci dokumentace, ale zde je problém, že dokumentace a program nikdy nebude odpovídat 1:1 a nemusí být dodržená konzistence při změnách, zatímco u grafických modelů toto řeší systém sám o sobě.

Řešením může být například Process Mining [12]. Process Mining umožňuje dolovat grafické procesy (zejména petriho síť) z logů. Je tak možné vydolovat graf petriho sítě i z programu. Procesní model pak může být nasimulován a výstupy zapsány do logu. Poté může být aplikován Process Mining a jeho výstup použit ke grafické dokumentaci procesu.

Dlouho trvající procesy.

Pokud máme procesy, které se opakovaně vykonávají a trvají dlouho (například nějaké výpočetní procesy), pak jejich opakované vykonávání může být zbytečná výpočetní zátěž. Na tyto typy procesů ale může být použit Smalltallk.

6 Omezení

Nevýhoda imperativních modelů (tj. Například modelů grafových toků a programovacích) je složitý model v případě, že máme volný proces. Omezení se s tímto umí vypořádat – základní myšlenka je – je povoleno vše, co není omezeno. Plně volný proces s omezeními si můžeme představit pomocí grafového modelu takto:

Na počátku je And-split do všech úloh a z nich vede vše do And-join. V běžném grafovém modelu bychom mohli paralelně vykonat všechny úlohy, kdy se nám zachce. Pokud bychom ale přidali omezující pravidla, mohli bychom tak omezit vykonávání mnohem snazším způsobem, než grafovým popisem (například udělat něco takového v BPMN je dost problém.).

Dobrým představitelem systémů s omezením je jazyk DECLARE [9,10,16,17]. Tento jazyk se používá společně s Petriho sítí (to bude ale prodiskutováno později). Jazyk obsahuje některé speciální konstrukty. Na ty hlavní se nyní podíváme. Tyto konstrukty jsou vždy relace mezi dvěma úlohami:

A1 Response A2.

Toto omezení říká, že pokud vykonáme A1, musíme poté vykonat i A2 (ale A2 můžeme vykonat, kdy se nám zachce).

A1 Precedence A2.

Toto omezení říká pravý opak. A2 můžeme vykonat pouze tehdy, pokud vykonáme A1.

A1 Succession A2.

Spojuje dvě předchozí podmínky dohromady. A2 můžeme vykonat, pokud máme hotové A1 a pokud vykonáme A1, musíme vykonat A2. Všimněme si, že tato relace v podstatě říká totéž, co sekvenční směrování v grafových modelech (tj. A-> B). Protože to říká, že po A následuje B (vynucuje si to) a zároveň, že B nemůžeme vykonat, pokud nevykonáme prvně A.

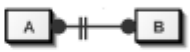
Samozřejmě, že tato omezení můžeme kombinovat a A2 může být závislé na více úlohách (tj. Třeba může následovat po A3, A4).

A1 Not Succession A2.

Toto omezení říká, že pokud nastane A1, nesmí se už vykonat A2. Nicméně A2 můžeme vykonat kdykoliv předtím. Můžeme mít samozřejmě zároveň obrácenou podmínku a tím vynutit, že kdykoliv v průběhu procesu můžeme vykonat pouze A1, nebo A2.

A1 Coexistence A2.

To říká, že pokud nastane A1, musí někdy nastat i A2 a naopak. Všechna ostatní omezení si můžete prohlédnout na obrázku 4.

Constraint	Meaning	Graphical representation
responded existence	if A occurs then B occurs before or after A	
co-existence	if A occurs then B occurs before or after A and vice versa	
response	if A occurs then eventually B occurs after A	
precedence	if B occurs then A occurs before B	
succession	for A and B both precedence and response hold	
alternate response	if A occurs then eventually B occurs after A without other occurrences of A in between	
alternate precedence	if B occurs then A occurs before B without other occurrences of B in between	
alternate succession	for A and B both alternate precedence and alternate response hold	
chain response	if A occurs then B occurs in the next position after A	
chain precedence	if B occurs then A occurs in the next position before B	
chain succession	for A and B both chain precedence and chain response hold	
not co-existence	A and B cannot occur together	
not succession	if A occurs then B cannot eventually occur after A	
not chain succession	if A occurs then B cannot occur in the next position after A	

Obr. 4. Ukázka pravidel Declare, převzato z [17]

6.1 Výhody

Vizualizace.

Declare ukazuje, že systémy s omezením, mohou být také popsány grafem, což může být důležitý parametr pro management, ale i pro přehlednost. Je snadné se ztratit v omezujících pravidlech, a proto je důležité, aby zde byla možnost vizualizace. Na druhou stranu grafy toku jsou velmi snadno pochopitelné i pro někoho, kdo nemá s workflow zkušenosti, zobrazení omezení je neintuitivní a je třeba zkušenost, aby v tom někdo viděl skutečný popis procesu, a to může být pro vedení neakceptovatelné.

Kombinace s ostatními přístupy.

Jak je vidět, je dost obtížné pomocí tohoto přístupu namodelovat pevně daný proces, takže omezení sama o sobě nejsou dost dobře použitelná. Ale Declare je sám o sobě nadstavba CPN Tools (Petriho sítě), takže je možné uvedený přístup kombinovat s imperativním přístupem – ať už programátorským, pravidly, či například BPMN. Je potom vhodné namodelovat základní kostru procesu grafovým modelem toku a poté nadefinovat omezení - třeba omezující podmínky na vykonání úloh, které se nacházejí v modelu zcela v jiných částech a pomocí BPMN bychom museli provést komplikované synchronizace.

6.2 Nevýhody

Nedostatek zkušeností.

Pro mnoho lidí bude asi intuitivnější imperativní popis (zejména pomocí grafu), ve kterém je jasné, jaké jsou následnosti kroků. Rozsáhlý proces popsaný pomocí omezení navíc může být velmi nepřehledný a komplikovaný. Navíc zde hrozí již zmíněné neočekávané stavy, které tvůrce opomněl zakázat.

Popsat pomocí tohoto přístupu pevně dané procesy není nejvhodnější, naštěstí jak již bylo řečeno, lze uvedené přístupy kombinovat.

7 Změny za běhu

Změny za běhu jsou v tuto chvíli důležité výzkumné téma [18], protože přestože jsou procesy možná na první pohled relativně jednoduché, tak nejsou. Důvodem jsou změny. Je naprosto běžné, že se procesy mění, není problém vytvořit novou definici procesu a nové instance spouštět podle ní, problém jsou změny samotných definic za běhu instancí. Článek [18] popisuje několik témat a problémů, které se týkají změn. První část se zabývá důvodem změny a další odstavec efektem změny.

Existuje několik důvodů změn. Prvním z nich je změna samotného procesu. To nastává v případě, že jsou nové požadavky na proces – například se změnil trh či situace v podniku. Dále může jít o změnu legislativy, změnu technologií - Jsou k dispozici nové přístupy a technologie a procesy na ně reagují změnou (obvykle k lepšímu). Typickým důvodem jsou chyby v návrhu, nikdo nedokáže proces navrhnout

neomylně a už vůbec ne myslet na všechny případy, které mohou nastat. Dalším důvodem jsou bezpečnostní nebo výkonnostní problém

Mezi efekty patří změna definice nějaké konkrétní procesní instance – to může být reakce na nějakou hodně specifickou výjimku, u které se rozhodlo, že nebude součástí změn celé definice. Změny mohou ale měnit přímo celou definici procesu. Aktuálně běžící procesy mohou doběhnout podle staré definice, nebo migrovat na novou definici (což není vůbec snadné). Nové procesy mohou startovat podle nové definice. Změny mohou probíhat v různých částech procesu - rok řízení, data, zdroje, organizační model, změny kódů, či integrací systému. Také můžeme provést přidání nových prvků, umazání starých, změna existujících, přeuspořádání existujících.

Změny můžeme provést okamžitě během vykonávání procesu (samozřejmě v rámci nějakých bezpečných zón jádra vykonání, ale z pohledu uživatele okamžitě). Bezpečné zóny fungují tak, že v okamžiku kdy proces přejde do určité části procesu, jsou provedeny patřičné změny (takže změny nemusí být provedeny vůbec). Okamžité změny jsou mnohem složitější na zajištění správného běhu procesu.

Existuje také několik přístupů jak změnit běžící instance. Dopředná obnova ruší staré instance a jsou provedena opatření. Zpětná obnova ruší staré instance, proběhne kompenzace (navrácení do původních stavů, například vrácení materiálů apod.), poté proběhne nový běh procesu od začátku podle nové definice. Staré procesy také mohou doběhnout podle starých definic a nové podle nových. Nejsložitějším typem je migrace, kdy staré procesy postupně převádíme na nové definice.

Změny v systému jsou poměrně složité a podle [18] jde dost těžko vytvořit nějaký obecný teoretický postup, jak je provádět, protože každá změna může být specifická a bude vyžadovat ad-hoc postup. Podle [18] je důležité, aby změny nějakým způsobem podporoval modelovací jazyk sám o sobě, to pomůže ad-hoc změnám za běhu. Například omezení jsou, co se změn týče velmi flexibilní, zatímco programový přístup a grafové modely toku jsou pro změny složitější, například u BPMN může dojít k absolutnímu rozbití celého procesu.

8 Závěr

Jak bylo vidět, každý modelovací přístup má různé výhody a také různá úskalí. Trendem začíná být spojování Grafových modelů toku (BPMN apod.) se systémy s omezením. Dokonce vznikají týmy, které řešení process mining pravidel. Tyto systémy umožňují namodelovat striktní řízení pomocí toku a v případě potřeby do něj zabudovat omezující podmínky. To umožňuje slušnou flexibilitu.

Modelování workflow pomocí programovacího jazyka má jistě své místo, odpadá nutnost použít klasický workflow systém a postačí pouze vhodné knihovny, navíc máme k dispozici běžné prvky programovacího jazyka. Nevýhodou je absence propojení s grafickým návrhem, čistý IT přístup (nemožnost najmout zaměstnance zodpovědného za modelování procesů, který není zkušeným programátorem) a také je zde horší reakce na změny za běhu.

Změny za běhu jsou jedním z těžkých aktuálních problémů výzkumu v oblasti. I když se zdá, že není možné nalézt nějaký obecný postup, výzkumníci se shodují, že je třeba problematiku změn podchytit, popsat a vyvinout modelovací prvky, které tyto změny co nejvíce zjednoduší, aby ad-hoc migrační postupy byly co nejsnazší.

9 Literatura

[1] Aalst, W., Hee, K.: Workflow Management: Models, Methods and Systems. The MIT Press, London, 2004. 384 s. ISBN 978-0262720465.

[2] Aalst, W., Workflow Patterns, dostupné na adrese: http://www.workflowpatterns.com/

[3] MÁČEL Lukáš. New approach in workflow process modelling. In: *ICMT'11: International Conference on Military Technologies 2011*. Brno: Universita Obrany v Brně, 2011, s. 525-533. ISBN 978-80-7231-787-5.

[4] MÁČEL Lukáš a HRUŠKA Tomáš. Bringing flexibility into dynamic process change:The process re-execution approach. In: *BUSTECH 2014: The Fourth International Conference on Business Intelligence and Technology*. Benátky: The International Academy, Research and Industry Association, 2014, s. 31-38. ISBN 978-1-61208-345-2.

[5] DRAGOS-ANTON MANOLESCU , Micro-Workflow: A Workflow architecture supporting compositional object-oriented software developement, Disertační práce, dostupná na: http://micro-workflow.com/PDF/phdthesis.pdf

[6] J.Wainer and F. de Lima Bezerra. Groupware: Design, Implementation, and Use, volume 2806, chapter Constraint-Based Flexible Workows, pages 151-158. Springer Berlin / Heidelberg, 2003.

[7] Workflow Management Coalition: *The Workflow Reference Model*, 1995, dostupný na URL: <http://www.wfmc.org/standards/docs/tc003v11.pdf>.

[8] OMG: *Business Process Model and Notation (BPMN): ver. 1.2.* 2009, dostupný na URL: <http://www.omg.org/spec/BPMN/1.2/>.

[9] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development, 23(2):99–113, May 2009.

[10] M. Pesic, M.H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), pages 287–298, Wash- ington, DC, USA, 2007. IEEE Computer Society.

[11] Bizagi, dokumentace dostupná na adrese: http://download.bizagi.com/docs/modeler/2408/en/Modeler_user_Guide_2408.pdf

[12] W. M. P. van der Aalst, "Process Mining", Berlin, Heidelberg 2011, ISBN 978-3-642-19344-6

[13] Využití Business pravidel ve workflow, dostupné na adrese: http://www.w3.org/2004/12/rules-ws/paper/105/

[14] Využití Business pravidel ve workflow, dostupné na adrese: http://www.bptrends.com/publicationfiles/09-05%20WP%20Workflow%20and%20Business%20Rules%20%20Lienhard%20-%20Kunzi.pdf

[15] Formalizace a verifikace Event Process Chains, dostupné na adrese: http://wwwis.win.tue.nl/~wvdaalst/publications/p74.pdf

[16] Declare, dostupné na adrese:
https://westergaard.eu/2013/08/cpn-tools-4-declare-constraints/
[17] A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps
Dostupné na adrese:
http://wwwis.win.tue.nl/~wvdaalst/publications/p733.pdf
[18] Dealing with workflow change: identification of issues and solutions
Dostupné na adrese:
http://wwwis.win.tue.nl/~wvdaalst/publications/p112.pdf