Teorie programovacích jazyků

Ing. Zita Němečková

August 28, 2017

Abstract

V rámci této práce bude probrána problematika workflow a vysvětleny klíčové pojmy a principy. Bude se jednat především o organizaci workflow, modely, Petriho sítě, management workflow, analýzu, funkce a architekturu. Tato práce je shrnutím jediné knihy [1].

1 Cíl práce

Tato práce má za cíl uvést čtenáře do problematiky workflow a seznámit jej se základnímy pojmy z této oblasti. Práce vychází z knihy Workflow Management [1] nebude-li řečeno jinak.
V první kapitole bude čtenář seznámen s workflow a důležitými pojmy s ním souvisejícími a to s důrazem na Petriho sítě. V následující kapitole bude probrán management workflow a jeho specifika. Jedná se o zdroje, které workflow může využívat a jejich vlastnosti a omezení. Budou probrány o alokační principy a analýza úzkých míst. Další kapitola se bude zabývat analýzou workflow a rozdílem mezi kvalitativní a kvantitativní analýzou. Pozornost bude věnování dostupnosti, zvučnosti procesu a metodám bez podpory počítače. Poslední kapitola bude zaměřena na funkce a architekturu workflow systému. Probrány budou referenční model, typy uživatelů, data a jejich viditelnost. Dále budou ukázány trendy ve vývoji systémů a popsán adaptivní workflow systém.

2 Organizace Workflow

V této kapitole budou popsány základní pojmy pro modelování workflow. Nejdříve bude důkladně probrána terminologie v dané oblasti. Dále bude kladen důraz na Petriho sítě, protože jsou často využívány pro formální popis workflow procesů.

2.1 Modely workflow

Správný model workflow systému je základem pro jeho úspěch. Proto je důležité se důkladně seznámit s terminologií, která popisuje modelování, aby nedocházelo k chybám z neporozumnění popisu modelů. Z tohoto důvodu si zde důkladněji rozebereme základní terminologii.

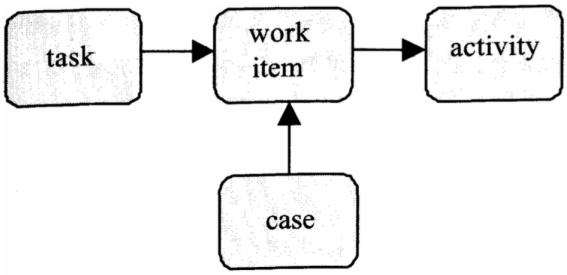
2.1.1 Případ (Case)

Případ je základním pojmem ve workflow, protože celý workflow systém je uzpůsobený práci s případy. Případ reprezentuje jednotku nějakého většího celku práce, který je zapotřebí vykonat - například zpracování objednávacího procesu, procesu vrácení zboží, procesu vyřízení přijímacího řízení apod.
Každý případ může být nějakého typu (typ může být modelován například procesní definicí). Typ tak reprezentuje stejné, či podobné případy. Každý případ má navíc identitu, můžeme ho jednoznačně identifikovat, takže v terminologii objektově orientovaného programování bychom mohli přirovnat konkrétní případ k instanci třídy a typ případu k třídě.
Každý případ má omezenou dobu životnosti. V nějakém okamžiku vzniká (například zadání objednávky) a zaniká (v případě vyřízení objednávky). V době svého života je identifikován stavem. Za prvé je to seznam atributů, podmínky a samotný obsah případu. Seznam atributů slouží jako uložiště řídících dat, která nějakým způsobem charakterizují samotný případ. U procesu objednání se může jednat o typ produktu, množství a další věci. Chování případu se pak může lišit podle těchto atributů. Podmínky (někdy se také používá pojem fáze) určují jakým způsobem případ probíhá, co vše je už hotovo, co není hotovo, nebo co ani nemůže být hotovo - toto je základ pro řízení toku případů. Rozdíl mezi atributy a podmínkami je v tom, že podmínky úzce souvisí s vykonáváním procesů a s jádrem systému, zatímco atributy spíše udávají doplňkové informace k případu. Obsah případu je poslední součást stavu. Zatímco atributy a podmínky jsou zpravovány workflow systémem a řídí se podle toho celé vykonávání procesu, obsahem případu se myslí externí datová uložiště - například soubory (externí datová uložiště, dokumenty přiložené k řešení případu, či jiná multimedia, se kterýma workflow systém nepracuje přímo).

2.1.2 Úloha (Task)

Úloha je součást případu. Jedná se o malou část operace, která bývá obvykle atomická. Bohužel v realitě tomu tak být vůbec nemusí, protože úlohou může být celý podproces, který však nemusí být modelován systémem. Úloha je z hlediska systému definována většinou jako atomická (tj. připomíná transakci), buď je provedena celá, nebo vůbec a v případě problému se provádí roolback. Transakce v databázi bývají krátké operace, kdežto úloha může probíhat klidně i několik dní. Úlohu si například můžeme představit jako vyplnění formuláře, kdy může systém umožňovat uložit rozdělanou práci. Nicméně z pohledu vykonání je úloha finálně hotova až v okamžiku potvrzení vyplněného formuláře (takže transakce úlohy proběhne pouze v okamžiku odeslání dat). V případě, kdy úloha tvoří podproces (například i na straně zcela jiné organizace), je dosažení atomičnosti velmi obtížné ne-li nemožné.
Typy úloh lze rozdělit do několika kategorií - manuální, automatická a poloautomatická. Manuální úloha probíhá zcela mimo vědomí workflow systémů, pouze se do ní vloží informace o tom, že byla vykonána, případně se doplní dodatečné atributy jako záznam. Příkladem může být výrobní operace v továrně. Poloautomatická úloha již vyžaduje spolupráci člověka a stroje, typickým příkladem je vyplnění formuláře, provádění operací pomocí uživatelského rozhraní apod. Automatická úloha je zcela vykonávána bez vědomí uživatele, jedná se tedy například o automatické skripty.
Úloha označuje nějakou obecnou definici práce, nikoliv konkrétní práci, která se má udělat. Tu označuje až Work Item (pracovní položka). Pracovní položka a úloha se rozlišuje tak, že úloha se používá jako definice úlohy v nějaké definici případu a pracovní položka už je konkrétní úloha v

konkrétním případu, která se musí vykonat. Pracovní položka je teda vytvořena v okamžiku, kdy je danou úlohu možné vykonat (vzhledem k podmínkám úlohy). Jakmile je k pracovní položce přiřazen pracovník, může začít být skutečně vykonávána a nazývá se aktivitou. Obrázek 1 to ukazuje graficky.



Obrázek 1: Souvislost mezi pojmy úloha, pracovní položka a aktivita

2.1.3 Proces (Process)

Proces definuje typ případu - tj předpis podle kterého se vykonává. Jak jsme si již řekli - v terminologii OOP by případ odpovídal instanci třídy a proces třídě. Proces může být definován různými způsoby, například Petriho sítí, ale může se jednat i o jiné modely. Proces se obecně skládá z úloh a podmínek. Úlohy definují práci, která se má v rámci procesu vykonat a podmínky určují řízení - které úlohy v jaké fázi procesu můžeme vykonat.
Proces může být znovupoužitelný a může být součástí jiného procesu. Tímto způsobem můžeme hierarchicky skládat procesy podobně jako podprogramy v běžném programovacím jazyce.

2.1.4 Směrování (Routing)

Protože většina workflow systémů vychází z modelu Petriho sítí, popíšeme si základní typy směrování (podmínek procesu).
Nejjednodužší podmínka je sekvenční směrování. To znamená, že pokud máme definovanou sekvenci mezi úlohami A a B (po A následuje B), tak úloha B může být provedena až teprve po tom, co je provedena úloha A. Paralelní směrování znamená, že po jedné úloze může být spuštěno až N různých úloh (které jsou na sobě nezávislé). Operace AND-Split dělí tok a operace AND-Join zase spojuje několik paralelních toků do jednoho. Výběrové směrování (Selective routing) funguje podobně jako paralelní směrování, ale řízení může přejít pouze do jedné větve - toto se může provádět automaticky na základě nějakého skriptu a datových atributů případu. Rozdělení se nazývá OR-Split a spojení OR-Join.

2.1.5 Spuštění (Enactment)

Pracovní položka může být spuštěna několika různými způsoby. První z nich je spuštění aktivitou uživatele (uživatel si sám vyžádá vykonání položky), dalším způsobem je externí událost (například zpráva EDI od jiného procesu) a poslední možností je časovač (chceme úlohu spustit v určitý čas). Případ samotný může být spuštěn zcela stejným způsobem - může ho založit uživatel (například založení procesu objednávky), může ho spustit externí událost, nebo časovač (spuštění samotné - inicializaci atributů - pak provede nějaký skript).

2.2 Petriho sítě

Petriho síť je jedním z mnoha na matematice založených modelovacích jazyků určených k popisu distribuovaných systémů. V roce 1962 Carl Adam Petri popsal Petriho síť jako nástroj pro modelování a analýzu procesů (především chemických). Její hlavní výhodou je spojení grafické reprezentace procesu s matematickým základem. Toto je odlišuje od jiných grafických metod, protože těm chybí formální definice.
Její použití pro popis workflow systému přináší mnohé výhody spojené s formálním zápisem. V první řadě se jedná o zamezení nejasností, nepřesností a protichůdných jevů, čehož je dosaženo

použitím přesně definovaného konceptu. Dále umožňuje využití analytickych nástrojů například pro verifikaci, která bude popsána později.

2.2.1 Formální definice

Petriho síť je definovaná jako pětice $PN = (P, T, F, W, M_0)$. Kde:

$P = \{p_1, p_2, ..., p_m\}$ je konečná množina míst
$T = \{t_1, t_2, ..., t_n\}$ je konečná množina přechodů
$F \subseteq (P \times T) \cup (T \times P)$ je množina hran
$W = F \to \{1, 2, 3, 4, ...\}$ je hodnotová funkce.
$M_0 = P \to \{0, 1, 2, ...\}$ je počáteční rozložení.

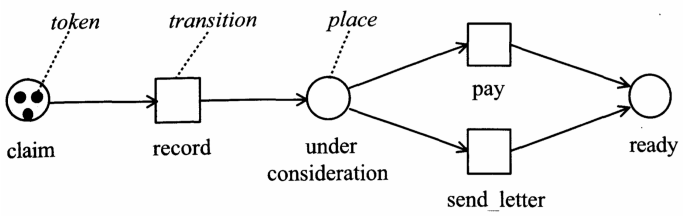
Petriho síť má dvě důležitá pravidla, která říkají, že množiny P a T jsou navzájem disjunktivní a obě množiny mají dohromady alespoň jeden prvek:

$$P \cap T = \emptyset \text{ a } P \cup T \neq \emptyset$$

Pokud neexistuje počáteční nastavení Petriho sítě, tak je zapsána jako $N = (P, T, F, W)$ a nazvána struktura Petriho sítě. Petriho síť s počátečním nastavením je možno zapsat jako (N, W_0).

2.2.2 Grafická reprezentace

Grafické zobrazení Petriho sítě je velmi podobné orientovanému grafu, jak je možné vidět na obrázku 2. Prvním rozdílem je fakt, že uzly reprezentují dva druhy částí Petriho sítě. Prvním je místo (place), které je zobrazeno jako kruh. Druhým je přechod (transition), které je reprezentováno čtvercem. Dále jsou mezi nimi orientované hrany, které značí další kroky v síti. U hran je důležité, aby vedly z místa do přechodu nebo z přechodu do místa. Není přípustné, aby existovala hrana, která spojuje dvě místa nebo dva přechody. A nakonec máme černé tečky, které zobrazují tokeny, které reprezentují aktuální stav systému. Přechod, který nemá žádné vstupní místo, je nazývaný zdrojový přechod (source transition). V opačném případě, kdy přechod nemá žádné výstupní místo je nazývaný odpadní přechod (sink transition).



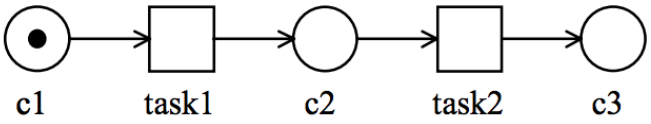
Obrázek 2: Jednoduchý příklad Petriho sítě.

2.2.3 Druhy směrování

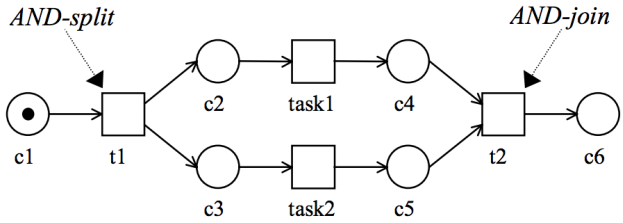
Jednotlivé úlohy mohou být nepovinné nebo podmíněnné jinou úlohou. Případně musí být úloha vykonána opakovaně. Posloupnost úloh může být také proměnlivá. Pomocí směrování můžeme určit povinnost a pořadí úloh. Existují čtyři základní směřování, které zde budou popsány.

Sekvenční směrování předpokládá, že budou úlohy vykonány postupně. Tedy úloha může být vykonána až poté, co předcházející úloha byla dokončena. Lze předpokládat, že takto vykonávané úlohy mají mezi sebou závislosti a jejich posloupnost je pevně daná. Toto směřování lze vidět na obrázku 3.

Paralelní směrování popisuje situaci, kdy v daném čase může probíhat více než jedna úloha. Pro formalní zápis se využije AND-split, který umožní průběh více úloh v daném čase. Je-li zapotřebí synchronizace pro další tok procesu, je využit AND-join jako podmínka pro spuštění další úlohy. Jednoduchý příklad lze vidět na obrázku 4.

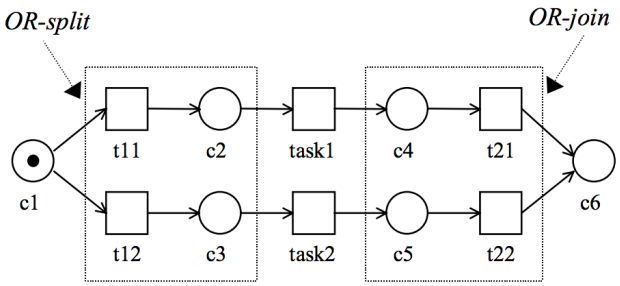


Obrázek 3: Jednoduchý příklad sekvenčního směrování



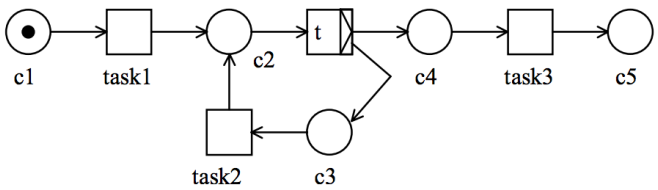
Obrázek 4: Jednoduchý příklad paralelního směrování

Selektivní směřování se používá v případech, kdy proces může mít různý průběh na základě parametrů. Může se jednat například o schvalování hypotéky, kdy zájemce může být odmítnut nebo schválen na základě svého příjmu. Pokud potom proces je rozdílný jedná se o selektivní směřování. V bodě rozhodutí o dalším směřování se používá OR-spit a pokud mají rozdělené větve procesu později společnou část, tak jsou spojeny do jedné větve pomocí OR-join. Demonstraci tohoto směřováí lze vidět na obrázku 5.



Obrázek 5: Jednoduchý příklad selektivního směřování

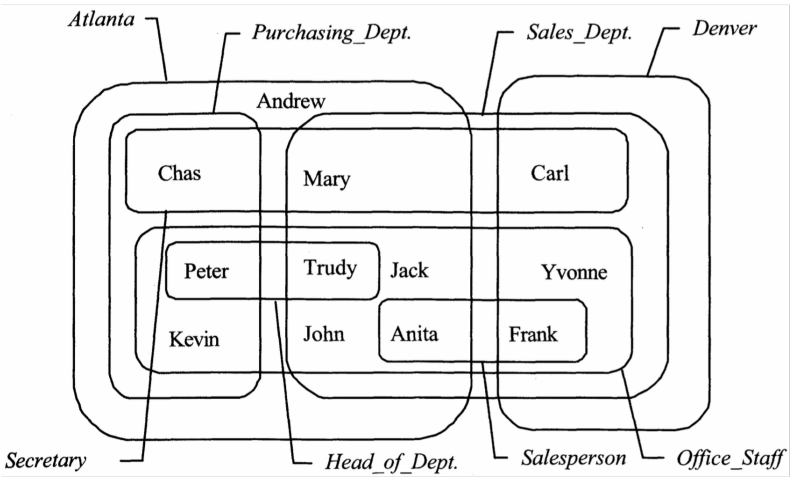
Iterativní směřování je posledním druhem směřování, který popisuje opakované vykonání úlohy nebo úloh. Příkladem může být opakování dané úlohy dokud se nedostaví požadovaný výsledek. Pomocí OR-splitu je určeno, jestli bude proces pokračovat nebo se znovu spustí předchozí úloha. Na obrázku 2 je tato iterace zobrazena.



Obrázek 6: Jednoduchý příklad iterativního směřování

3 Management workflow

V předchozí kapitole byly probrány modely procesů. Tato kapitola se zaměří na zdroje. Zdrojem je subjekt schopný vykonávat úlohy. Zdroje jsou ve workflow obvykle lidé, ale nemusí to tak být vždy. Například v továrních procesech se může jednat o stroje. Zdroj může být také nějaká výpočetní služba s omezenou výpočetní kapacitou. Zdroje mají několik charakteristik.
Kapacita určuje na kolika úlohách může současně zdroj pracovat. Dostupnost určuje časy, kdy je zdroj dostupný pro práci. V procesním modelu jsou potom definována pravidla pro alokování zdrojů. Například u úlohy může být definované, které zdroje ji mohou vykonávat (statická pravidla). Dynamická pravidla už závisí na kontextu vykonání procesu. Například z bezpečnostních důvodů platí, že pokud jeden zdroj vykonával úlohu A, nesmí vykonat úlohu B - například nesmí zároveň provádět práci a audit téže práce. Tato omezení jsou obvykle z bezpečnostních důvodů. Další pravidla se týkají strategií přidělování práce a plánování, ta si popíšeme později. Zdroje dále mohou obsahovat atributy. Atributy mohou být historie (statistika) výkonnosti pracovníka, jeho zkušenosti a seznam dovedností. Tyto atributy také mohou ovlivňovat rozhodování o alokaci.
Cena zdroje bývá mnohdy v literatuře opomíjená, ale je důležitá jednak pro kontrolu finančních toků a také pro alokační pravidla. Drahý zdroj může být zodpovědný, rychlý, spolehlivý a mít širokou škálu dovedností. Na druhou stranu je drahý a workflow systém toto může brát v potaz při alokaci.
Protože by bylo krajně nevhodné definovat pro každý zdroj pravidla, tak zdroje klasifikujeme do dvou základních tříd. První se nazývá role. Ta určuje jakou funkci daný zdroj vykonává - například účetní, programátor, sekretářka. Jedná se funkcionální klasifikaci zdrojů. Druhá se nazývá Organizační jednotka a určuje kam je zdroj zařazen z hlediska organizační hierarchie. Protože procesní řízení jde napříč organizačními jednotkami, role a organizace se mohou překrývat. Obě třídy mohou také tvořit hierarchii - role může být součástí jiné obecnější role, stejně tak oddělení bývají součástí větších oddělení. Na obrázku 7 vidíme ukázku překrývajících se rolí a organizačních jednotek, vidíme také, že některé tvoří hierarchie.
Alokační pravidla pak mohou být založena i na rolích a organizačních jednotkách. Workflow systém se může snažit nabízet úlohy zaměstnanci, který má s těmito úlohami nejvíc zkušeností, nebo se naopak může snažit vytížit nejméně zkušené zaměstnance a ty zkušené si nechat do zálohy. Lokalita zdroje také může hrát roli - můžeme se snažit alokovat ty, které jsou nejlépe fyzicky dostupné. Dynamická pravidla navíc mohou zohledňovat i lokalitu vykonání v rámci procesu, jestliže daný zdroj vykonává daný proces, je vhodné aby byly další úlohy v procesu nabídnuté jemu (není-li to v rozporu s bezpečnostní politikou).
Jak vidíme, alokační pravidla mohou být velmi komplikovaná. Workflow systémy nabízejí uživatelská rozhraní pro nastavení pouze pro základní alokační pravidla, zbytek bývá řešen pomocí skriptů.

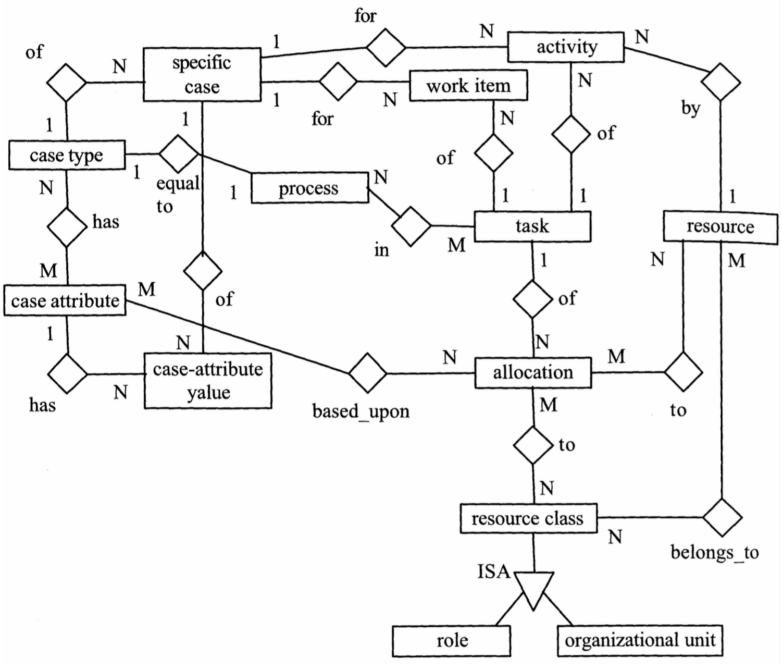


Obrázek 7: Ukázka zdrojů a jejich klasifikace.

3.1 Diagram entit

Nyní si popíšeme vztahy jednotlivých popsaných entit ve formě E-R diagramu na obrázku 8. Pro přehlednost si to vše vysvětlíme.

Nalevo vidíme Case Type (typ případu). To je ekvivalentní (1:1) s pojmem proces. Typ případu



Obrázek 8: ER diagram entit ve workflow.

má seznam typů atributů a každý atribut může patřit do více typů případu (tj. nadefinujeme-li si například atribut počet zboží na skladě, může být využíván ve více případech). Specific Case (případ) může patřít do právě jednoho typu případu a jeden typ případu může mít N případů (instancí). Konkrétní případ má seznam atributů a každý atribut patří právě do jednoho případu a je jednoho typu atributu. Hodnoty atributů tedy patří přímo případům, zatímco typy atributů mohou být sdíleny.

Úloha může patřit do více procesů (typů případu) a jeden typ případu má více úloh. Tato M:N vazba je zde proto, že většina workflow systémů umožňuje definovat úlohy buď v rámci platnosti daného procesu, nebo globálně. Globální úloha může být znovupoužita ve více procesech. Work item (pracovní položka) patří právě jedné úloze jednomu případu, ale jedna úloha může mít více nastartovaných pracovních položek (například v případě nějakého opakovaného vykonávání, nebo u speciálních úloh, které pouští více pracovních položek paralelně - ty se nazývají Multiple Instance Tasks). Aktivita je práce nad pracovní položkou (má tedy stejné vazby jako pracovní položka). Vsuvka autora - není zřejmé, proč v knize nedali vazbu mezi aktivitou a pracovní položkou.

Zdroj (Resource) může vykonávat práci na N aktivitách, ale za vykonání jedné aktivity je zodpovědný jeden zdroj. Zdroj může patřit to Třídy zdroje. Vazba M:N zaručuje volnost při definování zdrojů a tříd - jeden zdroj může patřit do vícero tříd a jedna třída může mít vícero zdrojů. V grafu je také vidět, že organizační jednotka a role jsou modelovány stejně (význam je pouze sémantický). Alokační pravidlo (Allocation) může záviset na typu zdroj,úloze, atributech typu případu a na konkrétním zdroji. (poznámka autora - není zřejmé, proč neuvedli závislost na hodnotě atributu pro konkrétní případ).

3.2 Alokační principy

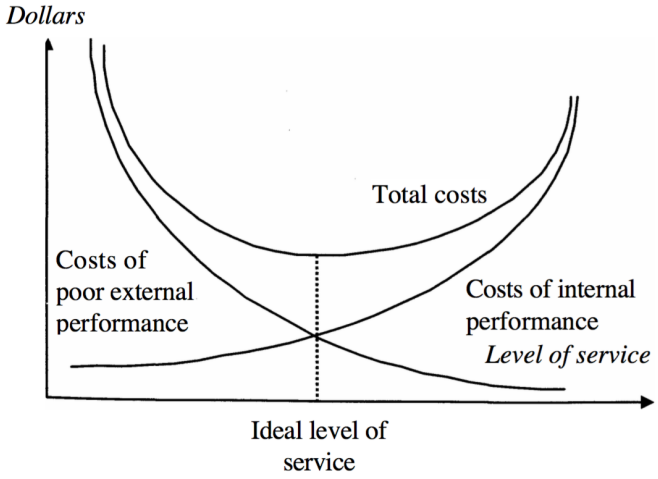
Kromě alokačních pravidel popsaných výše shrneme stručně různé používané heuristiky. Ty se netýkají pouze rozhodnutí, který zdroj má úlohu vykonat, ale jak určíme priority vykonávání úloh, pokud je zdrojů málo. V takovém případě vznikají totiž fronty.

FIFO (First in, first out) je jednoduchá strategie, která je často používaná a říká, že čím dříve úloha příjde, tím dříve bude vykonána. FIFO zaručuje nízké čekací doby, pokud nejsou příliš velké

rozdíly v čase vykonání úloh (například by se mohlo stát, že jedna dlouhá úloha bude blokovat vykonání 10ti krátkých místo aby se nejdříve vykonalo 10 krátkých a čekala pouze tato jedna úloha). FIFO také nebere v potaz prioritu úlohy (někteří zákazníci jsou důležitější, některé úlohy jsou kritické).
LIFO (Last in, first out). Tato strategie je pravým opakem FIFO a je zřídka využívaná, protože způsobuje velké čekací doby starších úloh. Jediné užitečné situace jsou tam, kde zpoždění vykonání úloh přináší firmě výhody - například zboží na skladě se může zúročovat apod.
Shortest Processing Time řeší problém uvedený ve FIFO - prioritně se vykonávají úlohy, které jsou krátké na čas vykonání. Opakem je Logest Processing Time, kdy se nám může vyplatit pravý opak, což nebývá tak časté.
Earliest Due Date je metoda, která je v praxi taktéž velmi používaná. Přednostně jsou řešeny ty úlohy, které nejvíc spěchají (mají nejbližší termín, kdy mají být hotovy). Samozřejmě nemá smysl u úloh, kde jsou termíny volné a jde nám o celkovou rychlost služeb.
V praxi jsou potom často využívány kombinace těchto metod. Navíc se přidává důležitost úlohy, protože zpoždění důležitých úloh může znamenat pro firmu velké penále. Shortest Processing Time je metoda, která sice dává dobré výsledky, avšak vyžaduje odhad pracnosti vykonání, který se může lišit podle typu případu (a jeho atributů), zkušenosti a výkonnosti alokovaného zdroje.
Další pravidla už jsou určena podle zdrojů. Jedno pravidlo říká, že je dobré využívat specializací zdroje - to znamená, že by zdroje měly být alokovány na úlohy, se kterýma mají největší zkušenosti. Pravidlo omezení tzv. Setup Times říká, že by lidé neměli příliš měnit typy úloh na kterých dělají za sebou a úlohy by měly být vykonávány v dávkách. Toto pravidlo se běžně používá v továrnách, ale platí i u lidí. Pravidlo generalizace zase říká, že prvně se alokují zdroje s nízkou obecnou použitelností, abychom měli široce použitelné lidi v záloze pro neočekávané případy.
Poslední pravidlo rozlišuje Push a Pull princip přidělování prací. V principu tlaku (push) je úloha přidělena zdroji, v principu tahu (pull) si naopak zdroj vybírá úlohy. V praxi bývají kombinace obou přístupů.

3.3 Analýza úzkých míst

V této kapitole si probereme "symptomy" špatné výkonnosti. Není třeba vždy sledovat procesy do úplných detailů, mnohdy stačí sledovat obecné charakteristiky, kterýma se právě špatná výkonnost projevuje. Typické symptomy jsou:
Příliš mnoho rozpracovaných případů může signalizovat, že máme nedostatek zdrojů, nebo dochází k vysokým čekacím časům například díky špatné komunikaci. Na druhou stranu to může znamenat, že je mnoho procesů ve frontě z důvodu vhodné optimalizace vzhledem ke zdrojům a setup time. Pokud je čas skutečné práce na procesu příliš malý v porovnání s celkovým časem (od začátku do konce procesu), pak to signalizuje velké synchronizační a čekací prodlevy. V zásadě to znamená skoro totéž, co předchozí bod. Nedodržování lhůt dokončení je opět signálem, že je něco v nepořádku, může to být ale způsobeno pouze špatným časovým odhadem pracnosti.
Externí výkonností jsou nazývány globální pohledy na proces - tj. z pohledu zákazníka - jak rychlá bude odezva objednávky, či procesu reklamace. Interní výkonností jsou zase parametry jednotlivých úloh, čekací doby a strategie alokací zdrojů. Je jasné, že externí a interní výkonnosti jdou proti sobě - dobrá externí výkonnost přináší lepší zisky, ale vyžaduje vysokou interní výkonnost a to stojí náklady. Typicky to ukazuje křivka na obrázku 9. Cílem je najít optimální rovnováhu mezi výkonností a cenou procesu. V praxi ovšem křivka nebývá tak jednoduchá, bývá vícedimenzionální a má více lokálních optim. Hledání optimální konfigurace může být provedeno například simulací v době návrhu. Pokud známe relativně přesné parametry procesu, můžeme si nasimulovat různé varianty a vybrat tu nejoptimálnější.



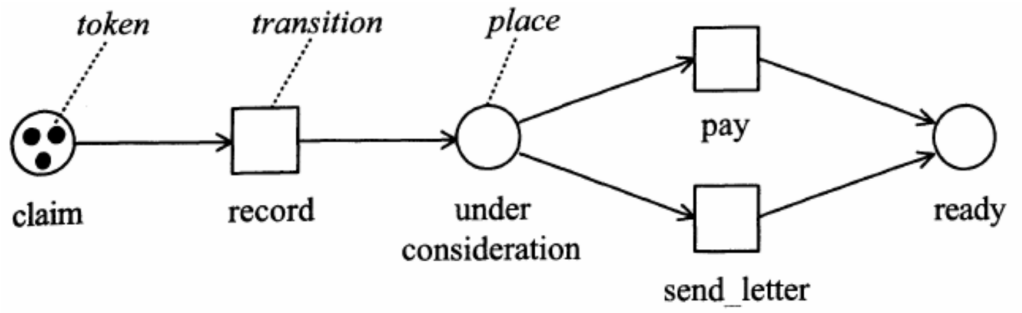
Obrázek 9: Křivky ceny a výkonnosti procesu. Hledáme optimální celkovou cenu.

4 Analýzy workflow

Workflow systémy umožňují zavést do procesu paralelní zpracování, které by jinak nebylo dost dobře možné pomocí starých postupů (například oběh papírů). To na druhou stranu klade nároky na kvalitu procesu - nesmí v něm vznikat například deadlocky (uváznutí - proces nemůže pokračovat a být ukončen), nebo livelocky (proces sice pokračuje, ale uvíznul v nekonečné smyčce a nemůže být ukončen). Rozlišujeme dva typy analýz. Kvalitativní analýza se zaměřuje na logickou strukturu procesu, zda nedochází k uváznutím, či nějakým dalším nežádoucím efektům v toku procesu. Kvantitativní analýza se na druhou stranu zaměřuje na výkonnostní pohled na proces - sleduje fronty, doby vykonání úloh a celých případů, sleduje chybovost apod.

4.1 Analýza dostupnosti (Reachability analysis)

V minulé kapitole jsme si popsali Petriho sítě, které jsou definovány stavem (pozice tokenů v místech). Existuje velké množství kombinací pozice tokenů, které mohou nastat v průběhu vykonávání procesů. Tyto kombinace mohou být popsány grafem dostupnosti. Uzly v grafu představují stav procesu (tj. rozmístění tokenů v místech) a hrany změnu ve stavu (přesun tokenu z jednoho místa do druhého).
Graf je tedy definován vektorem, kde délka vektoru je rovna počtu míst (každý prvek vektoru reprezentuje jedno místo) a hodnota prvku vektoru určuje počet tokenů v daném místě. Například z příkladu na obrázku 10. Aktuální stav tohoto procesu je reprezentován vektorem (3, 0, 0) - tj.

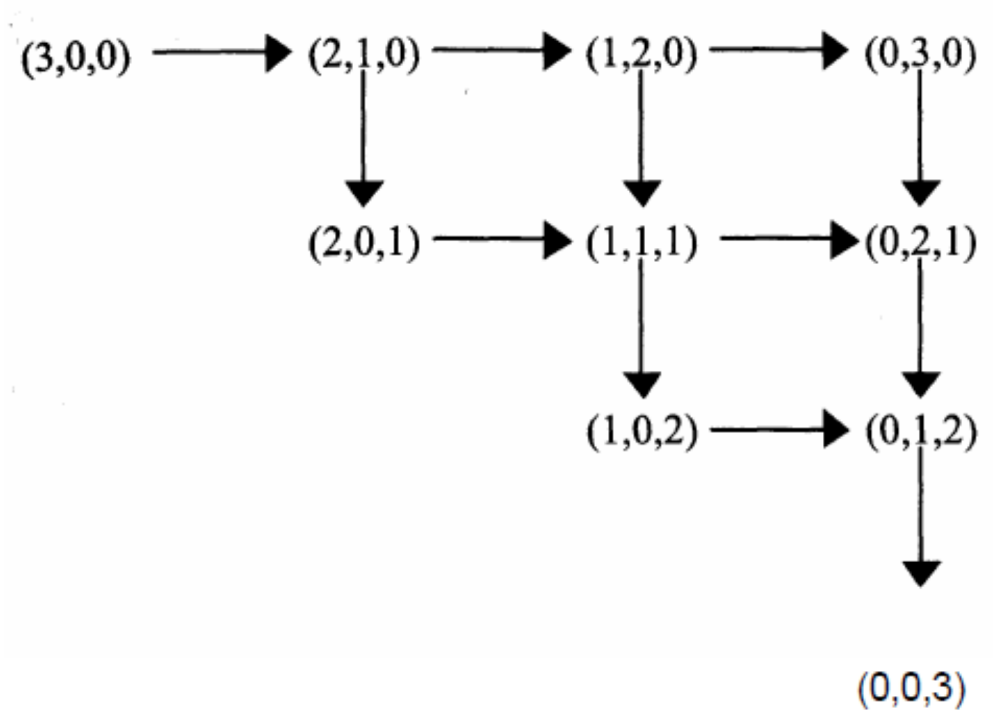


Obrázek 10: Jednoduchý proces se třemi místy

na pozici claim jsou 3 tokeny, under consideration a ready je bez tokenů (0). Na obrázku 11 vidíme graf průchodu stavy procesu. Finální stav je (0, 0, 3), což značí tři tokeny ve finálním uzlu ready. V prvním kroku je možná pouze jedna varianta a to případ, kdy první token přejde do stavu under

consideration. Pak jsou již dvě možnosti, za prvé může tentýž token pokračovat do stavu ready. V druhém případě se posune druhý token ze stavu claim do stavu under consideration. Vidíme, že graf neobsahuje cykly a všechny možnosti vedou do finálního stavu.

Počet hran vedoucích z grafu nám udává míru nepředvídatelnosti procesu - všechny možnosti



Obrázek 11: Graf dostupnosti

mohou nastat. Pokud existuje pouze jedna hrana, je stav plně předvídatelný (ale samozřejmě v realitě zde máme ještě čas a výjimky, které nikdo nepředpokládal). Pokud z uzlu nevede žádná hrana, jedná se o koncový uzel. Tento typ analýzy může být velmi užitečný, protože díky tomu, že jde o průzkum všech možností do kterých se může proces dostat, můžeme odhalit chybu v návrhu (například, že se proces dostane do nežádoucího stavu). Na základě předchozího příkladu si můžeme popsat několik typických problémů, které v návrhu nastávají (12):

Úlohy bez vstupní/výstupní podmínky

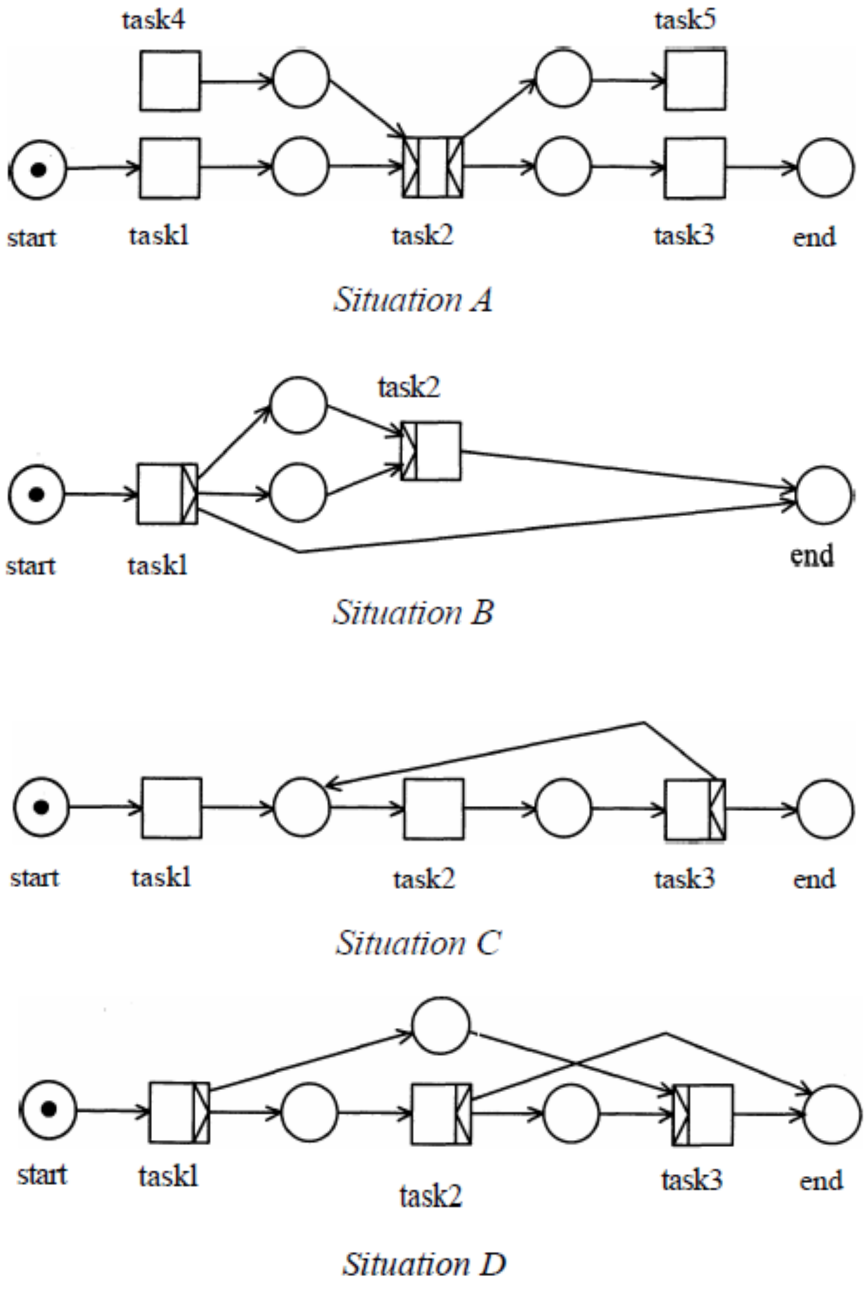
Pokud úloha nemá vstupní podmínku, není jasné, kdy by měla být vykonána. Pokud nemá výstupní podmínku, může proces doběhnout i bez vykonání této úlohy a není zřejmé čím daná úloha procesu přispívá. Na obrázku (?) tomu odpovídá Situace A - úloha 4 nemá vstupní podmínky a úloha 5 nemá výstupný podmínku.

Nedostupné úlohy (Dead tasks)

Úlohy, které není možné úspěšně vykonat se nazývají nedostupné, nebo také mrtvé úlohy. V situaci B se jedná o úlohu 2. Úloha jedna provádí OR-Split, takže na jedno z následujících míst umístí token, ovšem úloha 2 obsahuje AND-Join, takže čeká na 2 tokeny z obou míst, tyto tokeny zde ovšem díky předchozímu OR-Split nikdy nebudou. Takže daná úloha nemůže být ukončena. To samé potom platí pro úlohu 3 v situaci D.

Uváznutí (deadlock)

Uváznutí nastává v Situaci B za předpokladu, že úloha 1 doručí token na libovolné místo před úlohu 2, která pak čeká na dva tokeny na obou místech, což se nikdy nestane a proces nepřejde do finálního stavu. V situaci D máme totéž v úloze 3. I když se může z následujícího popisu zdát, že uváznutí a nedostupné úlohy jsou přesně totéž, není to pravda. V našem příkladě se sice



Obrázek 12: Chybně navržené procesy

překrývají, ale můžeme mít nedostupné úlohy, které nejsou nutné k ukončení (například navíc bez výstupní podmínky). Nedostupné úlohy jsou však dobrým indikátorem potenciálního uváznutí.

Nekonečný cyklus (livelock)

Nekonečný cyklus vzniká v situaci C, kdy úloha 3 současně umisťuje token do konce a zároveň před úlohu 2, tím pádem nikdy nemůže dojít k ukončení procesu. Protože workflow sítě jsou definované tak, že každý proces má mít možnost být ukončen (tj. ukončení nějakého případu), je to nežádoucí. Jinak samozřejmě můžeme mít obecný proces, u kterého je nekonečný obslužný cyklus žádoucí a nežádoucí je naopak možnost ukončení.

Existující aktivity a tokeny po dosažení finálního stavu

Pokud se ocitne token ve finálním stavu a v procesu stále existují jiné tokeny a tím pádem můžou být vykonané úlohy. Správný workflow proces by neměl umožnit vykonání úloh po tom, co dojde k přechodu do finálního stavu. Toto nastává v situaci C, kdy se nám hromadí tokeny ve finálním stavu a proces stále pokračuje.

Hladovění

Jestliže existuje úloha, která může být vykonána, nemůže být její vykonání oddalováno do nekonečna. Toto může nastávat například v případě, že ostatní úlohy dostávají při výběru z více možností prioritu a toto se děje cyklicky.

4.2 Zvučnost procesu (Soundness) a Workflow sítě

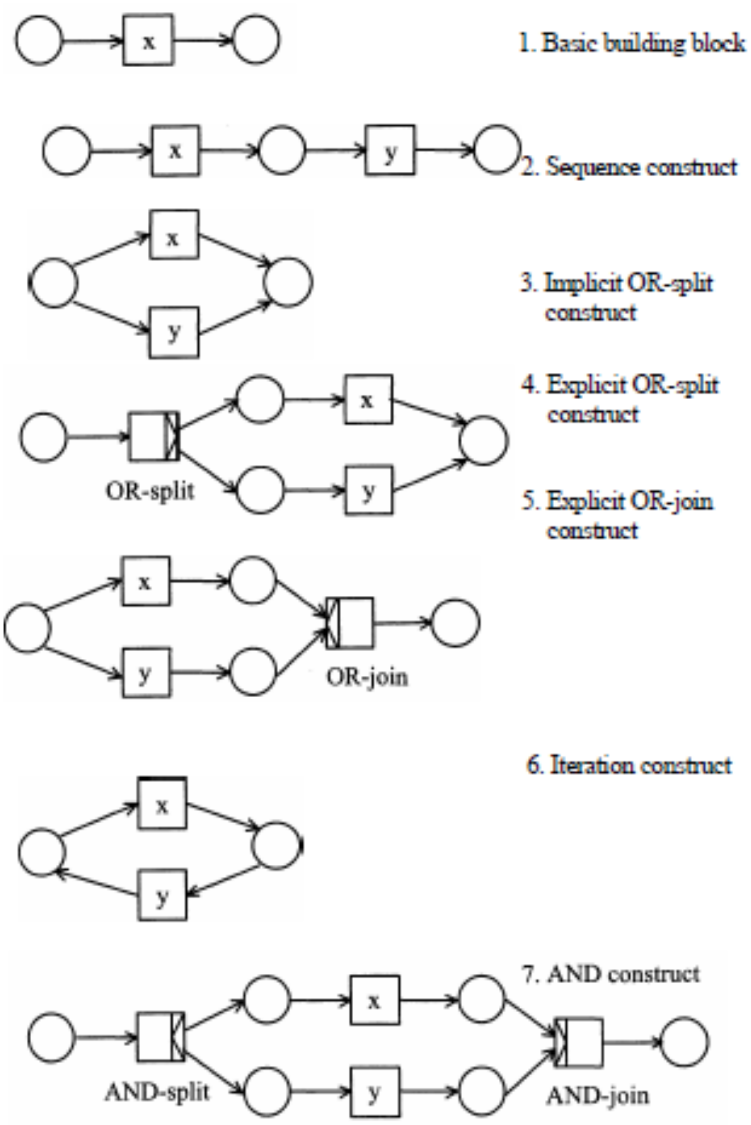
Na základě předchozích informací si shrneme, jak se definují workflow sítě a jejich zvučnost.

- Proces nesmí obsahovat nedostupné úlohy.
- Proces nesmí obsahovat úlohy bez vstupní a výstupní podmínky.
- Nesmí existovat tokeny po dosažení finálního stavu.
- Pro každý token umístěný na startovní pozici musí existovat právě jeden token umístěný ve finálním stavu.
- Nesmí docházet k hladovění.

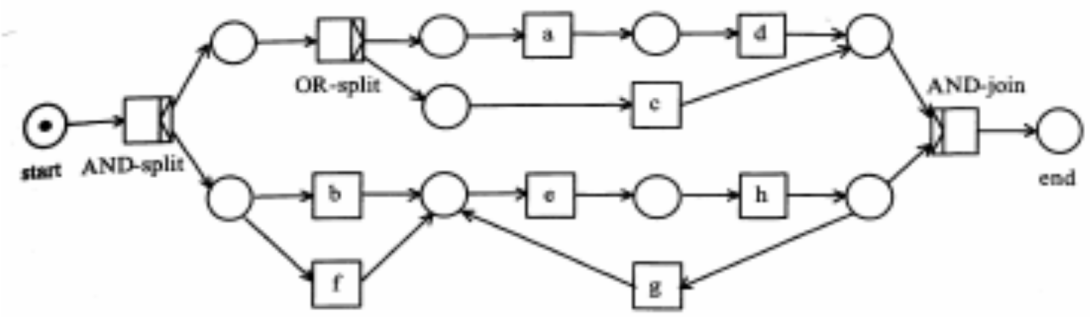
Analýza nekonečných cyklů je však problémem. Pokud proces obsahuje cykly, graf dostupnosti nemusí být možné vygenerovat donekonečna. V případě, že nevznikají nové tokeny, dojde sice k zdánlivě nekonečnému generování možností přechodů, ale nakonec zjistíme, že stavy se začínají opakovat - toto jsou nekonečné opakující se cykly v konečném grafu a graf dostupnosti lze použít. V jiném případě může docházet ke generování nových tokenů a tím postupnému zvětšování stavového prostoru, který tak roste do nekonečně. Na tyto typy procesů je nutné používat jiné typy analýz, ale to je zcela mimo tuto práci. Na základě této premisy definujeme ohraničené (bounded) procesy. Ohraničený proces je definován tak, že v žádném místě se nemůže naakumulovat více tokenů, než kolik jich bylo ve startovním stavu při spuštění procesu. Je tedy možné aby v rámci procesu vzniklo více tokenů než na začátku (například pomocí AND-Split), nesmí se ale stát, aby se začaly kumulovat v jedném místě. V případě AND-Split se tokeny nachází v jiných částí procesu a ve správném procesu je to ukončeno úlohou AND-Join, která je spojí zase do jednoho.

4.3 Metody analýzy bez podpory počítače

Aby bylo možné procesy analyzovat bez podpory automatických verifikací, musíme si nadefinovat další pojem. Proces je bezpečný (safe), pokud každé místo může obsahovat maximálně jeden token. Kdyby totiž mohlo obsahovat více tokenů, byl by daný proces příliš komplikovaný na vizuální analýzu člověkem.
Díky tomuto omezení můžeme nahradit úlohu v bezpečném a zvučném procesu jiným bezpečným a zvučným procesem a celý nový proces je opět bezpečný a zvučný. Této vlastnosti se využívá při stavbě hierarchických procesů, které samy o sobě umožňují lepší intuitivní analýzu. Toto je důležité, protože v praxi se mnohdy nechceme spoléhat pouze na automatické verifikační nástroje, ale chceme aby proces byl přehledný a v rozumné míře vizuálně ověřitelný i pro návrháře, případně manažery.
Na obrázku 13 vidíme základní stavební bloky. Jsou bezpečné a znělé. Pokud se budeme držet výše uvedených pravidel a tyto bloky mezi sebou kombinovat, budou nám vznikat pouze bezpečné a znělé kombinace. Obrázek (?) potom popisuje hotový proces, který byl složen právě na základě těchto pravidel a bloků. Můžeme se podívat, že máme pouze jeden začátek a jeden konec. Dále máme AND-split a AND-join konstrukt. V horní větvi máme vnořenou OR-split konstrukci se sekvencemi uvnitř. V dolní větvi máme implicitní OR a dále sekvenci a iteraci. Pohledem se můžeme přesvědčit, že proces splňuje všechny výše uvedené podmínky.



Obrázek 13: Základní stavební bloky



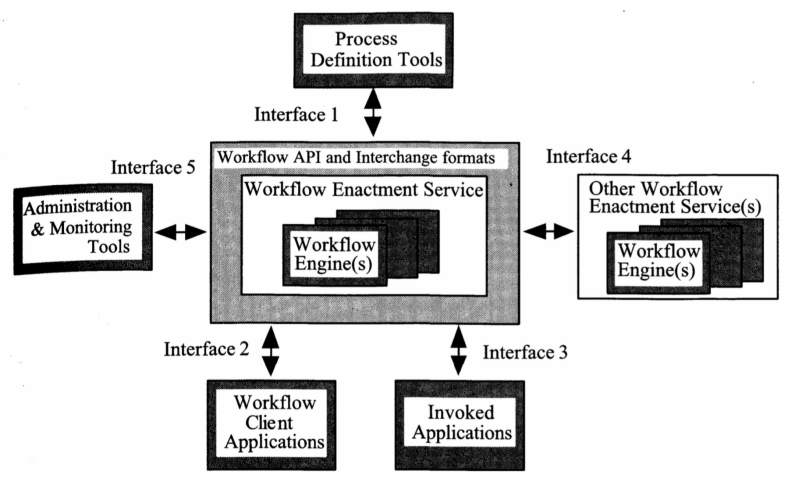
Obrázek 14: Proces vytvořený pouze ze základních stavebních bloků

5 Funkce a architektury

V dřívějších dobách byly datové operace součástí programů,pak vznikly databázové systémy, které oddělily data od programů. Workflow systémy mají za cíl oddělit obchodní procesy od programů. Stejně jako databáze přišly s unifikovaným datovým uložištěm, dotazovacími jazyky a analytickými nástroji, formalizace procesů umožňuje podobné věci. Workflow tedy slouží jako jakýsi procesní framework pro aplikace. Procesy mohou být samozřejmě přímo součástí programovacího jazyka, ale workflow nám dává mnohé výhody.
Protože proces je u workflow systémů definován modelem, který je interpretovaný, získáváme tím významnou kontrolu nad tokem programu. Procesní model můžeme namodelovat graficky (například pomocí standardu BPMN) a tím získat přehlednost oproti programovému kódu. Toto docení zejména manažeři. Workflow systémy také automaticky logují veškeré události, což je užitečné pro analýzy a monitorování. Díky formalizaci je proto možné mít k dispozici analytické a monitorovací systémy, které jsou znovupoužitelné pro všechny procesy. Protože procesní model je interpretován, lze ho v omezené míře měnit za běhu - můžeme měnit běžící proces, stejně tak můžeme použít verzování - v průběhu vykonávání nám mohou běžet jak staré typy procesů, tak nové typy procesů.

5.1 Referenční model workflow

Zatímco databázové systémy (respektive relační databáze) jsou dnes vyspělé a standardizované, u Workflow systémů tomu tak není. Mnohdy není jasné, jaké části a komponenty by měl workflow systém obsahovat. Proto vznikla iniciativa WFMC (Workflow management coalition), která vydala dokument s popisem, jak by měl workflow systém vypadat. Avšak i skrze snahu zatím nejsou procesy dostatečně standardizované a možná ani nikdy nebudou. Relační databáze mají tu výhodu, že pracují nad relativně jednoduchým modelem. Ale stejně jako se doposud nepodařilo dostatečně standardizovat složitější modely, jako jsou objektové a dokumentové databáze, nedaří se standardizovat ani workflow systémy. I když existují některé respektované standardy, mají spousty nedostatků a celá oblast se neustále vyvíjí. Je to dáno tím, že různé typy procesů mají různé požadavky a je velmi těžké to nějakým způsobem obecně formalizovat.
Na obrázku 15 jsou zobrazené komponenty workflow systému, které si nyní popíšeme.



Obrázek 15: Referenční model workflow od WFMC.

5.1.1 Workflow Enactment Service

Zaváděcí služba je jádro celého workflow systému. Může být složena z více workflow jader (Workflow engine) z důvodu distrubovatelnosti. Můžeme distrubuovat celé případy (case) do různých výpočetních serverů dle lokality procesu, můžeme ovšem také distrubuovat vykonávání samotných úloh - tj v některých případech může proces běžet i na několika serverech zároveň.
Procesní jádro je zodpovědné za veškeré řízení vykonávání procesů - interpretace procesního modelu, logování, kontroly konzistence, alokování práce uživatelům, řízení časovačů, komunikace s os-

tatními aplikacemi, persistenci vykonávání procesu (uložení do databáze), spouštění a ukončování případů a úloh.

5.1.2 Process Definition Tools

Jedná se o vývojové prostředí pro veškeré modelování procesů. Procesní modely mohou být znázorněny nějakým jazykem (například XML), takže tento modul může fungovat i nezávisle na procesním jádře a definice procesů tak mohou být přenášeny mimo systém.
Skládá se z různých částí. Typicky obsahuje prostředí pro definici procesu (například kreslítko procesního modelu). To kromě informací důležitých pro vykonání procesu může obsahovat i grafické informace (rozmístění úloh, definici směrování, pravidel, popisky, barvy, formáty...). Kreslítko může navíc obsahovat nástroje pro formální verifikaci modelu. Kreslítko také obsahuje nástroje pro tvorbu rozhodovacích pravidel (pravidla pro směrování v XOR). Další částí je nástroj pro modelování zdrojů, jejich kategorizaci, tvorba rolí, případně definicí práv nad procesy a případně daty. Některé workflow systémy také definují data a case atributy v rámci workflow definice (například formou stavění ER diagramů), jiné s němi vůbec nepracují a data nechávají čistě v režii skriptů. Pokud workflow systém umožňuje vykonávání dynamických jazyků (skriptů), pak může obsahovat i nějaké pomocné vývojové prostředí s intellisense, ladícími nástroji a podporou pro propojení s workflow.

5.1.3 Administrative and Monitoring Tools

Administrativní nástroje slouží k údržbě uživatelů, definic a nastavení režimů vykonávání, zkrátka se to nijak zásadně neliší od administrativních aplikací u databází. Umí ale i některé věci navíc - umožňují například "tvrdé" zásahy do vykonávání procesů, například v případě kdy se musí řešit nečekané uváznutí procesu, násilné ukončení procesu, změna za běhu, kterou proces nepodporuje, ruční migrace procesu na novou verzi, nebo ruční obsluhou nějaké nečekané výjimky, či Monitorovací nástroje jsou dvojího typu. První typ monitoruje výkonnost systému, vytížení, poslané zprávy a jejich latence, zkrátka technické parametry systému. Druhý typ monitoruje logické vykonávání procesů. Sem patří různé analytické a monitorovací nástroje, simulační nástroje, OLAP a datamining či process mining. Toto je důležitou součástí workflow systému, protože kromě samotné automatizace je důležitá i zpětná vazba, která umožňuje procesy vylepšovat. Monitoring zase může hlásit problémy, které se dějí a není to ještě zřejmé, nebo predikovat, že by se mohly brzy pravděpodobně stát.

5.1.4 Invoked Applications

Vykonávání úlohy může vést ke spuštění nějaké aplikace. Přestože tyto aplikace mohou být nezávislé na workflow systému, částečně do něj patří. Workflow systém obvykle iniciuje spouštění těchto aplikací a posílá jim vstupní data (například atributy případu), na tomto základě aplikace provedou nějaké výpočty (třeba se připojí do databáze apod) a dají zpět vědět workflow systému jak dopadl výsledek (mohou poslat zmodifikované atributy případu, které workflow systém může a nebo nemusí - z důvodu bezpečnosti - uložit). Vykonané aplikace však nijak nemohou ovlivnit řízení toku ve workflow systému přímo, maximálně na základě toho, že pošlou zpět nějaká data na základě kterých se workflow proces modifikuje.
Vykonané úlohy mohou být automatické a interaktivní. Automatický je například nějaký program, skript, či webová služba. Interaktivní může být například microsoft office, do kterého je veprogramováno programové rozšíření, které komunikuje s workflow systémem.

5.1.5 Workflow Client Applications

Toto je klientská část systému, kterou používají běžní uživatelé. Obsahuje například seznam úloh, na kterých může uživatel pracovat. Seznam může být standardní, tj. vestavěný do workflow, nebo integrovaný přímo s aplikací - takovýto seznam je upravený dle požadavků konkrétních procesů, tj. může pracovat s daty, provádět speciální úpravy, které standardní seznam neumí apod.

5.1.6 Other Workflow Enactment Services

Řekli jsme si, že workflow jádro může být distribuované mezi několik výpočetních uzlů. Tento případ je trochu jiný. V předchozím případě šlo o centrální řízení několika výpočetních uzlů, v

tomto případě jde o různé workflow systémy, které jsou mezi sebou propojeny můstky tak, aby komunikovaly (v případě, že se jedná o stejné workflow systémy, můstky nemusí být potřeba, avšak vždy je třeba dbát na bezpečnost a ne vše jde nastavit pravidly). Může se jednat o propojení systémů různých firem.

5.2 Typy uživatelů

V této části si popíšeme, kteří uživatelé s workflow systémem pracují, jaká je jejich role a které části výše popsaných rozhraní používají. Jeden člověk pochopitelně může plnit více rolí naráz.

5.2.1 Workflow Designer

Návrhář procesů má za úkol navrhnout a namodelovat proces. Používá k tomu Process Definition Tools. Jeho úkolem je namodelovat toky, nastavit pravidla, určit atributy případů. Návrhář procesů nemusí být nutně programátor, ale může se jednat o analytika či projektového manažera, protože většina uživatelských rozhraní umožňuje modelovat procesy bez hlubší znalosti programovacích jazyků.
Vývojář má na starosti psaní obslužných kódů, které nesouvisí přímo s procesním modelem. Ať už se jedná o dynamické interpretované skripty, kompilované kódy nebo webové služby, které provádějí business logiku aplikace. Může také psát rozšíření do workflow systému, které slouží k lepší adaptaci programu pro dané potřeby organizace.
Administrátor má na starosti instalace, nasazení, správu uživatelů, práv, monitorování výkonu, sledování logů a chyb.
Procesní analytik používá reportovací a monitorovací nástroje. Sleduje výkonnost procesů, provádí analýzy ať už pomocí standardních statistických nástrojů, grafů, nebo pomocí pokročilých analytických nástrojů a simulací.
Zaměstnanec má na starosti vykonávání úloh. Manažer je zodpovědnou osobou za celý běh procesů a řídí ostatní pracovníky. Mnohdy bývá zároveň procesním analytikem, nebo designerem.

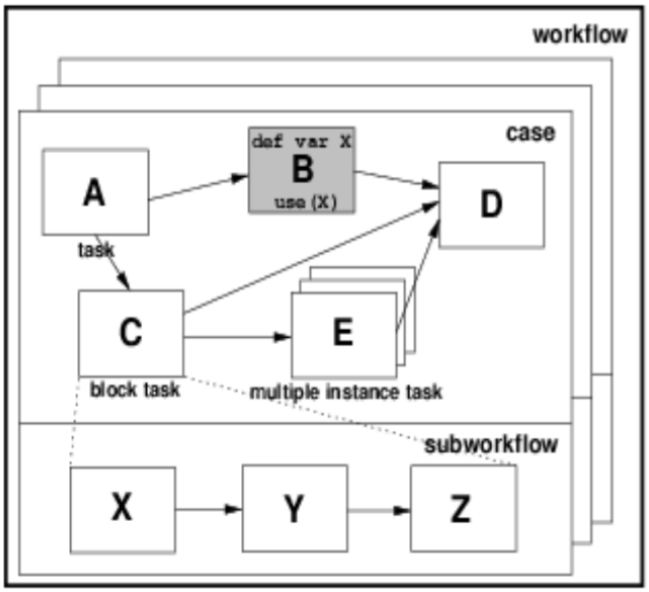
5.3 Data ve workflow systému

Zde popíšeme různé typy dat, které workflow systém obsahuje. Tato data mohou být uložena v jedné databázi, ale liší se svým významem.
Procesní definice obsahuje veškeré informace ke spuštění procesů - procesní model, jeho grafické informace, názvy, popisky, definice pravidel, alokační pravidla pro vykonávání úloh, skritpy ve formě dynamického interpretovaného jazyka apod. Dále máme informace o zdrojích, jejich klasifikaci a oprávnění. Technická data obsahují nastavení pro administrátory. Historická data slouží jednak k analýzám a jednak k případným revizím a auditům. Analytická data jsou uložené výstupy analýz z historických dat. Procesní data se vztahují k případům (atributy případů), stavům procesu, nebo k celé definici procesu, či workflow aplikaci. Procesní data tedy určují celý stav vykonání procesu. Interní data jsou data, kam si ukládá workflow systém své vnitřní běhové informace, které ovšem nesouvisí přímo s uložištěm pro jednotlivé případy (Case). Aplikační data jsou taková data, které nespravuje přímo workflow systém.

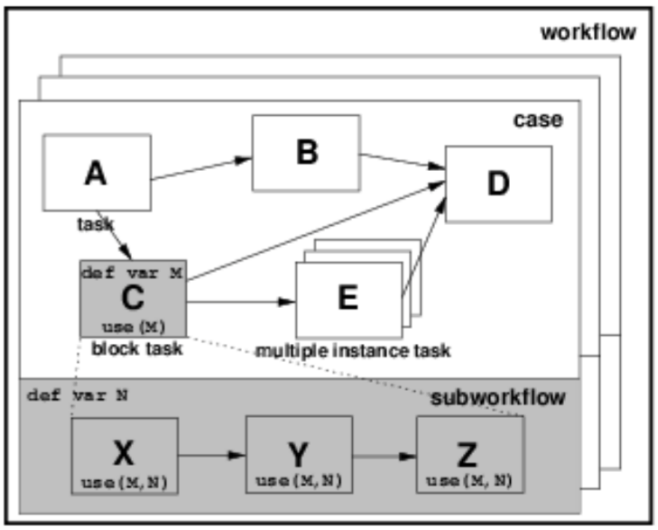
5.3.1 Vzory procesních dat

Iniciativa Workflow Patterns (dostupná na www.workflowpatterns.org) definuje různé typy pohledů na data, která se vyskytují ve workflow systému. Jedním z pohledů je viditelnost. Data mohou být viditelná v rámci úlohy (lokální data úlohy), celého workflow systému (globální data), typu procesu (v teorii OOP tomu odpovídají statická class data), celého konkrétního případu (Case data), či konkrétního označeného bloku v rámci případu, nebo data v rámci jedné procesní hiearchie. Viditelnost má podobný význam jako u programovacích jazyků.
Viditelnost je také důležitá například při implementaci rekurzivních podprocesů, kdy se musíme odkazovat na data v jedné procesní hierarchii (data příslušící danému zanoření), ale zároveň se můžeme odkazovat na Case data, která obsahují kořenová data k případu (zde si například můžeme uložit sdílená data ke všem zanoření).

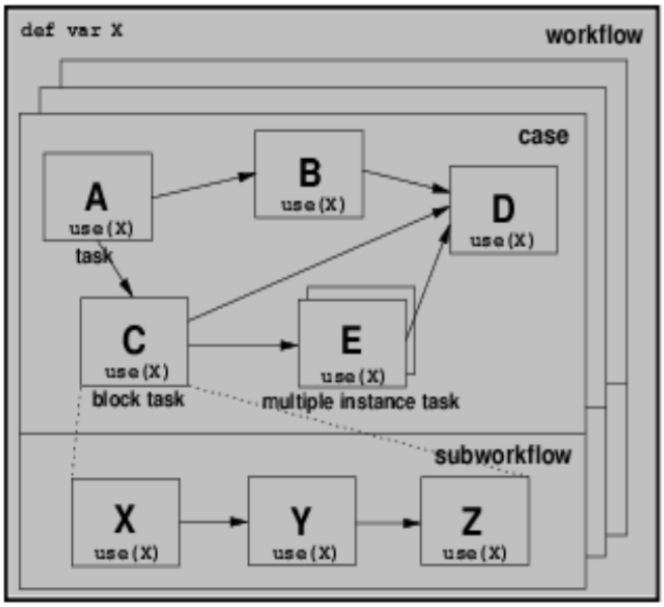
Data mohou být sdílená (tj. odkazujeme se na konkrétní uložiště - uložiště může být v úloze, případu, podprocesu - viz. odstavec výše), nebo můžeme data posílat ve formě toku (zpráv) mezi úlohama a procesama - v programovacím jazyce tomu odpovídají data alokovaná v dynamické paměti a data předávaná mezi funkcemi. Data také mohou být předávána ve formě hodnoty, nebo odkazu. V případě meziprocesní komunikace se obvykle používá předávání zpráv.



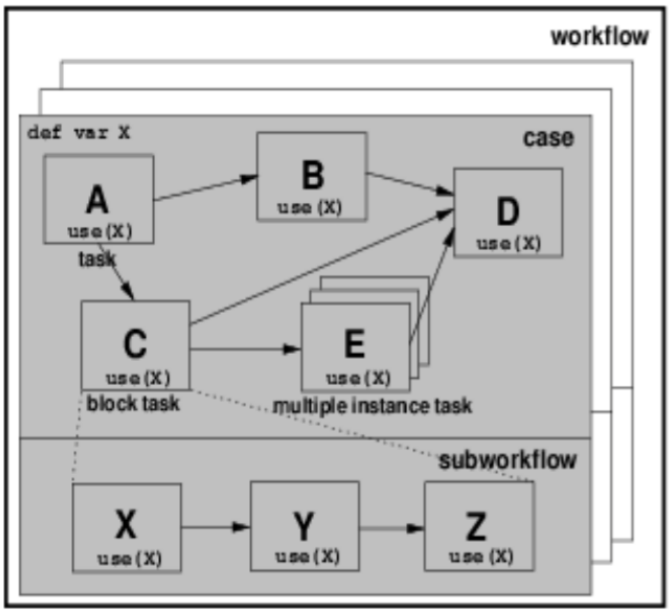
Obrázek 16: Viditelnost dat pro úlohu. [2]



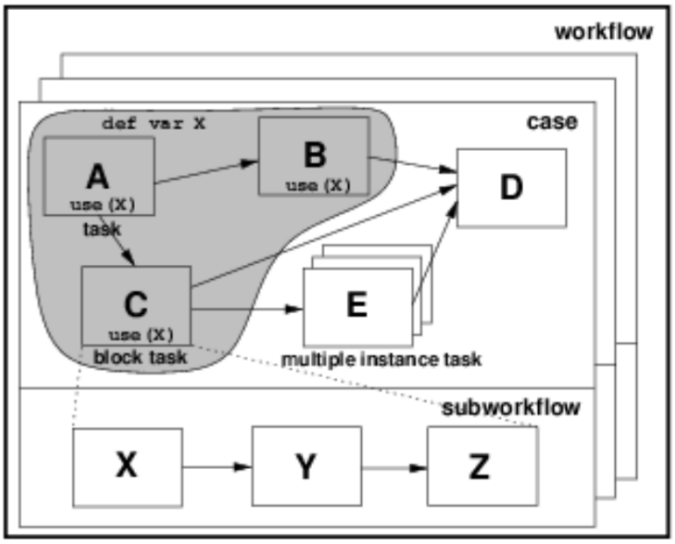
Obrázek 17: Viditelnost dat pro podproces. [2]



Obrázek 18: Viditelnost dat pro celý systém. [2]



Obrázek 19: Viditelnost dat pro celý případ. [2]



Obrázek 20: Viditelnost dat pro část procesu. [2]

5.4 Vývojové trendy

Zde si popíšeme jednotlivé oblasti problematiky workflow, jejich současný stav, problémy a budoucí pravděpodobné trendy vývoje.

5.4.1 Modelování

Procesní modely jsou stále ve výzkumu, přestože již vzniklo několik uznávaných standardů (jako například BPMN). Potíže jsou v tom, že se stále hledají vhodné modelovací prvky, které by dokázaly vhodně modelovat i nestandardní situace (výjimky, synchronizace, definice různých omezení pro systémy s vysokou volností). Typické workflow systémy jsou totiž navrhované primárně pro produkční procesy, což jsou opakovatelné, dobře definované procesy. Opačným případem jsou ad-hoc procesy, kdy máme pouze několik málo případů daného typu. Další problém jsou změny za běhu. Ukazuje se, že toto je velmi náročná problematika a nelze jí dost dobře řešit obecně, mnohdy se to řeší ručními zásahy, či převodními skripty, které se liší proces od procesu. Vhodný procesní metamodel toto ale může usnadnit.

5.4.2 Verifikace a ověřování

Formální verifikace procesů samozřejmě není snadnou záležitosti a lepší metody se neustále vyvíjí. Problémem může být převod procesních modelů do petriho sítě, verifikace výjimek a dalších těžce vefifikovatelných věcí (skriptů, obcházení pravidel, ručních zásahů do systému). Díky možnosti ručních zásahů do systému je však možné případná uváznutí a problémy rozumně řešit.
Ladící nástroje mohou obsahovat debugger skriptů a procesů, pohodlnou navigaci logů, nástroje pro migraci případů na nové verze (řešení změn za běhu). Zajímavá také může být podpora pro simulaci, kdy si můžeme proces odsimulovat ještě než ho pustíme do produkce.

5.4.3 Analýzy, Simulace, Monitorování a predikce

Díky rozvoji Big Data a umělé inteligence je tato oblast taktéž důležitou součástí výzkumu. Výkonné počítače s velkými datovými uložišti a rozvoj snímačů umožňuje logovat velké množství dat. Historická data mohou sloužit k tvorbě simulačních modelů, k analýzám, monitorováním a predikcím. Mohou být určena jak k budoucím vylepšením procesů, tak jako podpora řízení.
Rozvoj získávání znalostí z databází, umělé inteligence a neuronových sítí (rekurentních neuronových sítí, hlubokého učení a zpětnovazebního učení) ukazuje v dnešní době posun, takže se dá očekávat, že tyto metody budou stále častější i v implementaci.

5.4.4 Plánování

Protože existující workflow systémy se zaměřují spíše na administrativní procesy, nevěnují zatím dostatečnou pozornost plánování. Oblast plánování využívá poznatky z teorií prohledávání stavového prostoru (horolezecký algoritmus, simulovaného žíhání), teorii omezení nebo evolučních algoritmů. Samostatnou oblast pak tvoří agentní a multiagentní systémy, které fungují na bázi distribuovaného plánování.

5.4.5 Transakce

Spolehlivost vykonávání transakcí je jedna z nejobtížnějších oblastí, protože workflow bývá z principu distribuované - komunikuje s externími aplikacemi, jinými workflow systémy a vyměňuje EDI zprávy. Úloha samotná taktéž nemusí být zcela atomická - jednak může obsahovat podproces a jednak se zde řeší například problém, když skript uvnitř úlohy doběhne a zavolá commit a potom systém spadne, workflow neví zda skript doběhl v pořádku nebo ne a úlohu nastartuje podruhé. Proto je důležité například, aby všechny volání úlohy měly přiřazené jednoznačné identifikace, podle kterých skript pozná, že již toto volání jednou vykonal. Roolback taktéž není tak jednoznačný jako u databázových transakcí, protože workflow transakce může trvat dny, týdny i měsíce. Vrácení transakce tak může znamenat spuštění celého nového procesu navrácení, který může obsahovat vrácení fyzického materiálu, převody financí, účtování penále, či dokonce vyjednávání podmínek, které nebyly přesně definované předem.

5.5 Vývoj systémů

V této kapitole budou diskutovány postřehy k praktickému vývoji systémů jak z hlediska návrhu procesů, tak z hlediska implementace a obecné teorie programování a informačních systémů.

5.5.1 Business Process Reengineering

K maximalizaci výhod workflow systému nestačí pouze automatizovat existující procesy, protože ty bývají nastaveny s využitím starších přístupů. Často je nutné zcela změnit přístup a procesy definovat znovu, těmito postupy se zabývá oblast Business Process Reengineering. Probíhá v několika fázích. V první fázi jsou existující procesy diagnostikovány, jsou hledány nedostatky a navrhována vylepšení, která může workflow systém přinést. Může se například jednat o paralelizaci některých úkolů, které dříve musely být z důvodu omezení vykonávány sekvenčně, využití oběhu elektronických dokumentů apod. V druhé vzniká zbrusu nový proces. Ve třetí fázi dochází k převedení starých procesů do nového systému. Ve čtvrté fázi je celý proces měřen, analyzován a následně opět optimalizován - zde se obvykle už neprovádí reengineering, ale postupné zlepšování procesu.

5.5.2 Rapid Application Developement (RAD)

Tato oblast úzce souvisí s workflow. RAD v sobě zahrnuje rychlý, postupný vývoj systému založený na prototypování a rychlém nasazení postupně vyvíjejích se částí. Obojí workflow systémy podporují. K prototypování může přímo sloužit kreslítko procesního modelu, kdy je možné spousty věcí navrhnout přímo se zákazníkem a on už vidí část výsledku, může se k němu vyjádřit a dodat případné doplňkové informace včas. Kromě procesního modeleru workflow systémy také umožňují ruční editaci datových modelů (ER diagramy) a poloautomatickou tvorbu formulářů (vygenerování na základě ER a metadat, nebo ruční postavení formuláře v uživatelském rozhraní). Obojí je užitečné pro prototypování. Protože procesní model je interpretován, není třeba provádět při změně procesu nasazení nové verze systému, které by u běžných IS muselo nastat. Taktéž podpora dynamických interpretovaných skriptů (které se používají pro menší obslužné rutiny) je možné měnit za běhu bez potřeby nasazení.

5.6 Adaptivní workflow

Tahle kapitola je v knize na straně 192, takže je třeba jí správně zařadit do už hotového textu, nepatří to k verifikaci, bude to někde na konci kapitoly funkce a architektury.
V tuto chvíli současné workflow systémy nemají dostatečné nástroje na změny za běhu. Je to

také jedna z nejobtížnějších oblastí ve workflow. Přitom je to užitečná vlastnost. Workflow proces modeluje skutečné obchodní procesy podniků a ty se během života mění - může jít o změny v zákonech, nové technologické možnosti, nové ekonomické příležitosti, nebo zkrátka může jít o výskyt chyby (či nějakého nedostatku) v procesu, na který se přišlo až během produkce - této situaci se říká výjimky. Takováto změna není jednoduchá a potýkáme se s několika následujícími problémy. První je správnost (corectness). Zde se řeší jaké změny můžeme provést, aby byla nová verze procesu stále korektní. Rozlišujeme syntaktickou správnost (zda je nový procesní model korektní) a sémantickou správnost (zda budou aktuálně běžící procesy schopné doběhnout). Dynamická změna (dynamic change) zase řeší problém, co udělat s běžícími procesy v rámci migrace na novou verzi. Je také třeba dát vědět managementu nějaké agregované informace o změnách v dobře pochopitelné podobě.

5.6.1 Klasifikace změn

Existuje několik pohledů na změny. Procesní perspektiva sleduje proces z pohledu změny zapojení úloh, úlohy mohou být přidány, smazány, nebo jejich pozice může být změněna. Perspektiva zdrojů se zabývá tím, jak byly modifikovány zdroje a jejich klasifikace (třídy). Alokační perspektiva řeší změny alokačních pravidel - tj. jak budou zdroje přidělovány úlohám. Perspektiva úloh se zabývá změnou jedné konkrétní úlohy a její funkcionality. A konečně systémová perspektiva se zabývá změnou konfigurace celého systému.
Pro workflow systémy je dominantní procesní perspektiva a ta je také nejobtížnější. Můžeme opět definovat dva typy. Individuální změna nastává v případě, že chceme změnit pouze jeden běžící případ. Změna může nastat buď za běhu případu - například se objeví nečekaná výjimka a proces se jí musí přizpůsobit - tento způsob může být lepší, než definování obrovského množství výjímek, které mohou proces velmi znepřehlednit. Druhá varianta je, když změna nastane před spuštěním případu. Tato varianta je velmi užitečná pokud si chceme proces sestavovat dynamicky na základě parametrů případu. Strukturální změna nastává v případě, že změníme celou definici procesu všech případů. Toto je typické pro globální změny například kvůli změnám zákonů, či prostého vylepšení procesů. Může se však jednat i o opravu chyby, která byla zanesena špatným návrhem. Další důvod může být i špatné technické parametry procesu - například výkonnost.
Problém je v tom jak změnit běžící instance u procesní perspektivy. To nastává u individuální i strukturální změny, ale pouze pro instance, které již běží. Instance které ještě neběží mohou být spuštěny podle nové definice. Staré instance ale běží podle původní definice a je třeba je nějak převést na novou definici. To může být provedeno okamžitě nebo ve speciálních zónách. Zóny mají tu výhodu, že máme kontrolu nad tím jaké změny a kdy budou provedeny. Při okamžité změně může dojít totiž k velkým problémům s konzistencí procesu. Můžeme se také rozhodnout staré instance nechat běžet podle starých pravidel, pokud nesplní podmínky (například jsou již téměř u cíle). Zóny nám tak vytváří jakési milníky procesu, které rozhodují o tom, jak a zda proces migrovat na novou verzi. Dále můžeme provést několik typů obnov při migraci. Dopředná obnova zruší všechny staré procesy a jsou provedena ruční opatření. Zpětná obnova zruší staré procesy a proběhne kompenzace (navrácení do původních stavů, například vrácení materiálů), poté proběhne nový běh procesu od začátku podle nové definice. Pokračování je způsob kdy staré procesy necháme doběhnout podle staré definice. Migrace je způsob, kdy staré procesy převedeme na nové.

References

[1] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems.* MIT Press, 0-262-01189-1, 2002.

[2] Workflow Patterns Initiative
http:www.workflowpatterns.com