Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

METODY BYZNYS MODELOVÁNÍ

pro kombinované a distanční studium

Prof. Ing. Ivo Vondrák, CSc.

Ostrava 2004

Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

POKYNY KE STUDIU

Metody byznys modelování

Pro modul Metody byznys modelování studijního programu „Kvantitativní a informační podpora managementu“ jste obdrželi studijní balík obsahující

- integrované skriptum pro distanční studium obsahující i pokyny ke studiu

Prerekvizity

Pro studium tohoto předmětu se nepředpokládají žádné speciální předměty. Za výhodné lze považovat absolvování předmětů z oblasti Infromačních technologií zabývajících se specifikací softwarových systémů cestou objektového či datového modelování.

Cílem modulu

je seznámení se základními pojmy používanými při specifikaci byznys (podnikových) procesů s využitím různých paradigmat a metod. Celý kurz je rozdělen do dvou částí, přičemž první z nich je věnována neformálním, příp. semi-formálním, metodám postavených na třech základních druzích abstrakcí – funkčním modelování, modelování chování a strukturálním modelování. K tomuto účelu budou představeny tři metody implementující tyto přístupy, a to metoda IDEF (Integrated DEFinition), EPC (Event-driven Process Chains) a UML (Unified Modeling Language). V druhé části budou studující obeznámeni s formálními metodami, konkrétně s aplikací teorie Petriho síti pro účely jednoznačné a přesné specifikace byznys procesů včetně jejich analýzy a verifikace. Po prostudování modulu by měl student být schopen sestavení modelů podnikových procesů nezávisle na použitých principech a softwarových nástrojích.

Pro koho je modul určen

Modul je zařazen do magisterského studia oboru “ Kvantitativní a informační podpora managementu“.

Při studiu každé kapitoly doporučujeme následující postup:

Skriptum se dělí na části, kapitoly, které odpovídají logickému dělení studované látky, ale nejsou stejně obsáhlé. Předpokládaná doba ke studiu kapitoly se může výrazně lišit, proto jsou velké kapitoly děleny dále na číslované podkapitoly a těm odpovídá níže popsaná struktura.

Čas ke studiu: 1 hodina

Na úvod kapitoly je uveden **čas** potřebný k prostudování látky. Čas je orientační a může vám sloužit jako hrubé vodítko pro rozvržení studia celého předmětu či kapitoly. Někomu se čas může zdát příliš dlouhý, někomu naopak. Jsou studenti, kteří se s problematikou databází ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.

Cíl Po prostudování tohoto odstavce budete umět

- popsat …
- uvést příklady z praxe, kdy …

Ihned potom jsou uvedeny cíle, kterých máte dosáhnout po prostudování této kapitoly – konkrétní dovednosti, znalosti.

Výklad

Následuje vlastní výklad studované látky, zavedení nových pojmů, jejich vysvětlení, vše doprovázeno řešenými příklady z praxe. Při výkladu je základní text doplňován tučně vyznačenými důležitými a novými pojmy. Kurzivou jsou psány příklady z praxe, buď v rámci textu nebo jako ucelené řešené příklady samostatně označené.

Shrnutí pojmů

Na závěr kapitoly jsou zopakovány hlavní pojmy, které si v ní máte osvojit. Pokud některému z nich ještě nerozumíte, vraťte se k nim ještě jednou.

Otázky

Pro ověření, že jste dobře a úplně látku kapitoly zvládli, máte k dispozici několik teoretických otázek.

Úlohy k řešení

Protože většina teoretických pojmů tohoto předmětu má bezprostřední význam a využití v praxi softwarového inženýra, jsou Vám nakonec předkládány i praktické úlohy k řešení. V nich je hlavní význam kurzu a schopnost aplikovat čerstvě nabyté znalosti při řešení reálných situací hlavním cílem kurzu.

Klíč k řešení

Výsledky zadaných příkladů – stejně jako teoretických otázek výše jsou uvedeny v závěru učebnice v klíči k řešení. Používejte je až po vlastním vyřešení úloh, jen tak si sebekontrolou ověříte, že jste obsah kapitoly skutečně úplně zvládli.

Příprava na tutoriál

Souhrn znalostí nebo vypracovaných úloh, se kterými máte přijít na tutoriál. Mohou to být náměty k diskusím, otázky k promýšlení. Studující se tak mohou připravit na společná setkání a výsledkem je omezení okamžitých improvizací. .

Průvodce studiem

V tomto rámečku budou občas napsány pokyny o tom, co je důležité umět, co stačí jen přečíst informativně apod.

Úspěšné a příjemné studium s touto učebnicí Vám přeje autor kurzu Ivo Vondrák.

OBSAH

1. ÚVOD DO PROBLEMATIKY

Čas ke studiu kapitoly: 2 hodiny

1.1. Úloha byznys modelování

Cíl Po prostudování tohoto odstavce budete znát

- jaký mají vliv informačních technologie na problematiku byznys modelování,
- co je účelem byznys modelování.

Výklad

Několik slov úvodem

Informační a znalostní společnost, ve které se pohybujeme, je postavena na vědomostech a znalostech uplatňovaných v sítích spolupráce. Informační technologie se staly mechanismem realizace takové společnosti. Obsahem studijního materiálu, který nese název Metody byznys modelování, je prezentace metod a přístupů používaných ke specifikaci a analýze podnikových procesů. Informační technologie tak kromě návrhu a implementace informačních systémů a z pohledu realizace efektivní komunikace sehrávají důležitou roli při specifikaci chování organizace jako takového. Je důležité si uvědomit, že efektivní nasazení informačních systémů je podmíněno právě korektním návrhem procesů organizace nebo podniku. Důsledkem požadavku korektního popisu těchto procesů je vznik celé řady metod a nástrojů, které lze podle účelu použití rozčlenit do následujících tří kategorií:

1. BPR (Business Process Re-engineering) nástroje určené k modelování a analýze byznys procesů. Cílem je umožnit radikálně, nebo postupně procesy vylepšovat a umožnit podle nich vlastní řízení organizace či podniku.
2. ERP (Enterprise Resource Planning) systémy jako SAP, BAAN, Oracle apod. umožňující automatizovat výrobní procesy, finanční toky a řídit lidské zdroje, právě na základě explicitně popsaných procesů. Byznys modelování se tak stává počáteční fází softwarového procesu, na jehož konci je v podniku či organizaci implementovaný informační systém.
3. WFM (Workflow Management) systémy reprezentující generické softwarové nástroje pro definici, správu, realizaci a vlastní řízení podnikových procesů.

BPR nástroje slouží primárně k účelům analýzy a vylepšování procesů, zatímco ERP a WFM jsou softwarové nástroje určené k jejich realizaci. Rozdíl mezi posledními dvěmi uvedenými spočívá v míře explicitního vyjádření těchto procesů a v možnostech jak je dynamicky měnit. WFM svým pojetím přímé podpory byznys procesů tak poskytují lepší možnosti než systémy ERP, které mají tyto procesy implicitně zakódovány v rámci jejich implementace.

❑ Účel byznys modelování

Nezávisle na výše uvedeném způsobu použití modelů můžeme definovat hlavní účel byznys modelování jako *vytvoření korektní specifikace byznys procesů a analýzu jejich vlastností.*

Na závěr bychom ještě měli zdůraznit, že byznys modelování a modelování byznys procesů jsou pojmy, které se bežně zaměňují a jsou svým významem považovány za shodné.

Shrnutí pojmů 1.1.

Byznys modelování a informační technologie.

BPR, ERP a WFM systémy.

1.2. Základní pojmy

Cíl Po prostudování tohoto odstavce budete znát

- základní pojmy z oblasti modelování byznys procesů.

Výklad

Základní definice

Smyslem byznys modelování je vytvořit takovou abstrakci procesu, která umožňuje pochopení všech jeho aktivit, souvislostí mezi těmito aktivitami a rolemi reprezentovaných schopnostmi lidí a zařízení zapojených do daného procesu. Pokusme se tedy v dalším textu o definici jednotlivých pojmů, které jsme již intuitivně používali nebo těch, které použity teprve budou.

Definice 1.1 (*Byznys proces*): Byznys proces je po částech uspořádaná množina procedur a aktivit, které společně realizují podnikatelský nebo strategický cíl, obvykle v kontextu organizační struktury definující funkce rolí a jejich vztahy.

Pojmem procedura rozumíme podproces obsažený v daném procesu. Pojmem *po částech uspořádaná množina* pak vyjadřujeme fakt, že ne všechny aktivity a procedury lze seřadit do jediné posloupnosti. Jinými slovy řečeno, takových posloupností může být více a mohou být řazeny vedle sebe – mohou být souběžně, paralelně uskutečnitelné.

Definice 1.2 (*Model byznys procesu*): Model byznys procesu je abstraktní reprezentace byznys procesu obvykle umožňující jeho další zpracování automatizovaným způsobem.

Někdy se také hovoří o specifikaci nebo definici procesu. Tato specifikace může být neformální nebo formální. *Neformální specifikace* používá přirozený jazyk, případně obrázky, tabulky a ostatní nástroje umožňující pochopit popisovaný proces. *Formální specifikace* je technika umožňující jednoznačně specifikovat proces díky precizně definované syntaxi a sémantice použité metody.

Jinými slovy vyjádřeno, jde nám o takové modely, které lze vytvářet v počítači, tam je ukládat, analyzovat a případně dále využívat v ERP či WFM systémech.

Definice 1.3 (*Workflow*): Workflow je automatizovaný byznys proces.

Byznys proces a workflow jsou zaměnitelné pojmy, protože jejich význam je totožný. Jediným rozdílem je, že workflow spravuje a řídí k tomu určený software – ERP nebo WFM systém. Je však důležité si uvědomit, že právě workflow, díky svému počítačovému zpracování, klade vysoké nároky na specifikaci procesu, na jeho přesnost a jednoznačnost.

Pozn.: Workflow je pojem, který zatím v českém slovníku nemá ekvivalent na rozdíl od slova byznys, které je obsaženo v pravidlech českého pravopisu. Proto budeme v daším textu namísto nepohodlného *tok prací*, který je doslovným překladem, používat původní anglický výraz.

Definice 1.4 (*Aktivita*): Aktivita je popis činnosti, která reprezentuje jeden atomický (dále nedělitelný) krok ve vykonání procesu.

Aktivita může být *manuální*, která nevyžaduje počítačovou podporu, nebo může být automatizována (*workflow aktivita*). Workflow aktivita vyžaduje kromě lidských i strojově orientované zdroje. Speciálním případem workflow aktivity je *automatická* aktivita, která probíhá zcela bez lidské účasti.

Definice 1.5 (*Instance procesu*): Instancí procesu rozumíme jednotlivý případ vykonání procesu.

Z výše uvedeného tedy vyplývá, že proces můžeme chápat jako popis toho, jak jeho jednotlivé případy (instance) mají být vykonány. Každá instance má svůj začátek a konec a na jejím výstupu je konkrétní produkt (věc, služba apod.). Obvykle každý proces má celou řadu instancí (případů), které jsou podle takového předpisu realizovány. Systém řízení jakosti je postaven právě na tomto principu opakovatelnosti, která zaručuje stabilitu kvality vytvářeného produktu. Existují však situace, kdy proces má pouze svou jedinou instanci. V takovém případě hovoříme o tom, že tato instance má svůj vlastní *projekt*. Slovo projekt je tak synonymem pro slovo proces. Speciálním případem jsou také kontinuální procesy (např. výrobní linky), které nemají přesně definovaný počátek a konec. Přesto i zde je možné identifikovat periodu, kterou můžeme považovat za jednotlivý případ – instanci procesu.

Definice 1.6 (*Instance aktivity*): Instance aktivity reprezentuje činnost prováděnou při vykonávání procesu, t.j. v rámci dané instance procesu.

Definice 1.7 (*Role*): Role je soubor vzájemně se doplňujících dovedností.

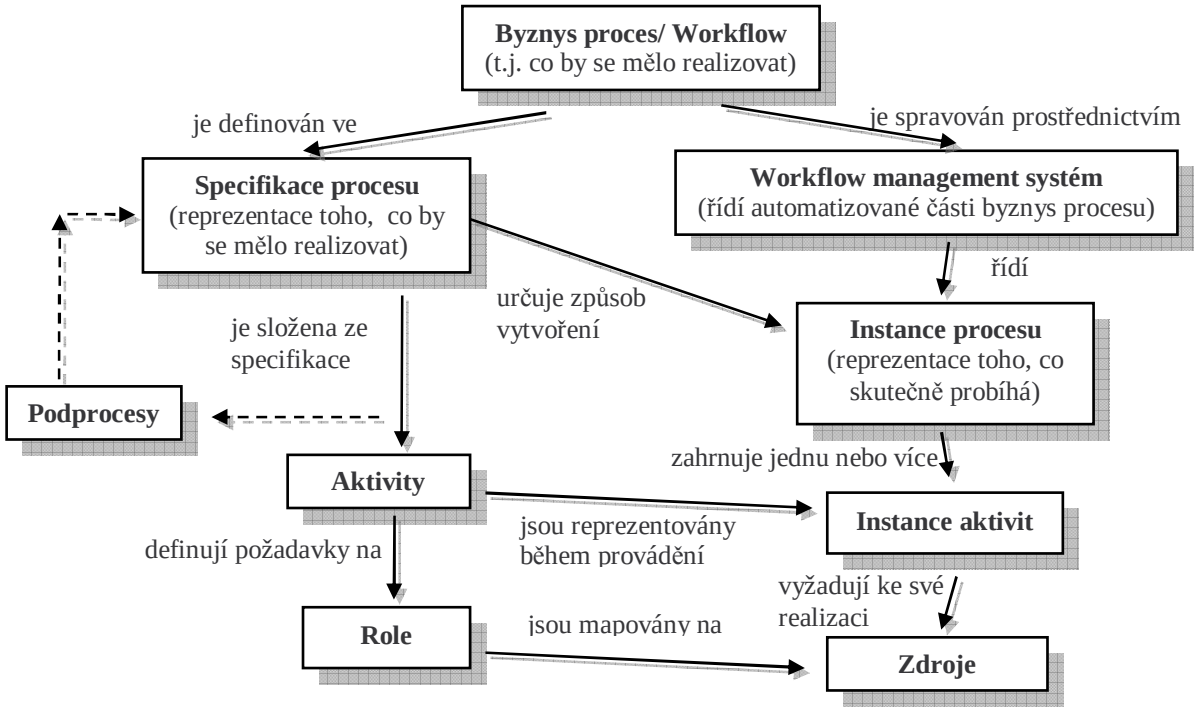
Role jsou přiřazovány k jednotlivým aktivitám s cílem umožnit jejich plnění v rámci vykonání procesu.

Definice 1.8 (*Zdroj*): Zdroj je prostředek nebo skupina prostředků nutných k vykonání aktivity. Zdroje mohou být lidské nebo strojově orientované.

Z výše uvedeného mimo jiné vyplývá, že role definují chování, kompetence a zodpovědnosti jednotlivých osob nebo skupiny osob spolupracujících v týmech. Jednotlivé osoby (lidské zdroje) jsou mapovány na požadované role dle toho, jak jsou požadované kompetence slučitelné se schopnostmi těchto osob.

❑ Ontologie procesního inženýrství

Ontologií rozumíme systém jasně definovaných pojmů a jejich vzájemných vztahů popisující danou oblast znalostí. V našem případě tedy oblast znalostí týkající se procesního inženýrství. V minulé části jsme zavedli definice jednotlivých pojmů. Nyní se pokusíme o vytvoření sémantického grafu, který k uvedeným pojmům přiřadí i vzájemné vztahy (obr. 1.1).



Obr. 1.1: Ontologie procesního inženýrství

Našim cílem je popsat v rámci dané ontologie znalostní oblast týkající se metod specifikace procesů. V dalším textu budou tedy popsány základní přístupy, které lze použít pro sestavení modelů byznys procesů včetně možností jak tyto modely analyzovat a ověřit správnost jejich navržení.

Shrnutí pojmů 1.2.

Byznys proces, workflow, aktivita, role a zdroje.

Ontologie procesního inženýrství.

Otázky 1.2.

1. Jaký je rozdíl mezi byznys procesem a workflow?
2. Co je to instance (případ) procesu?

2. ZÁKLADNÍ PŘÍSTUPY K BYZNYS MODELOVÁNÍ

Čas ke studiu kapitoly: 12 hodin

Cíl Po prostudování tohoto odstavce budete

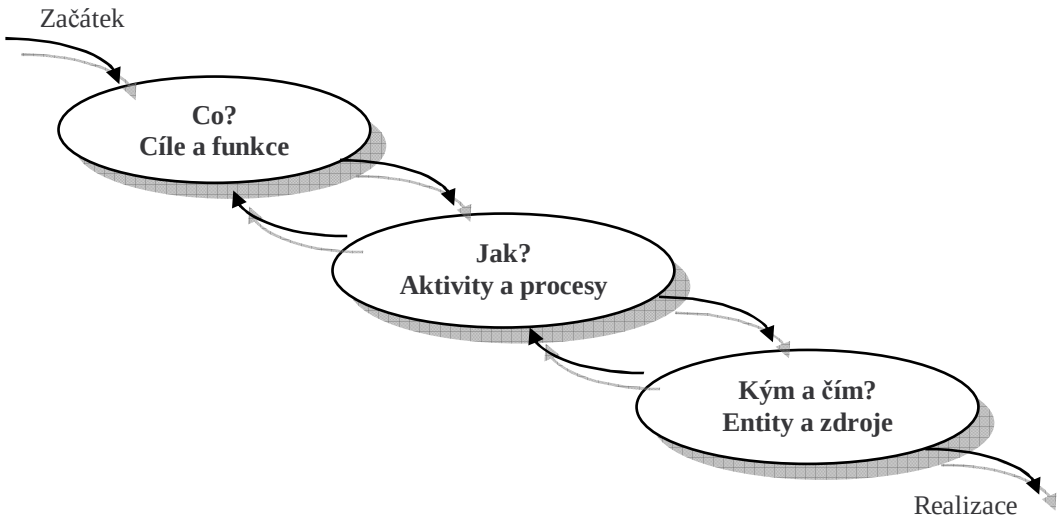
- znát základní přístupy k byznys modelování,
- znát a umět použít metodu IDEF pro účely funkční specifikace,
- znát a umět použít metodu EPC pro specifikaci událostmi řízených procesů,
- znát a umět použít profil jazyka UML pro účely byznys modelování,
- seznámeni s meta-modelem obecného byznys modelu.

2.1. Abstraktní rámec specifikace modelu

Výklad

Postup návrhu byznys procesu

Účelem modelování je vytvoření takové abstrakce procesu, která umožňuje pochopení všech jeho aktivit, souvislostí mezi těmito aktivitami a rolemi reprezentovaných schopnostmi lidí a zařízení zapojených do daného procesu. V současné době lze nalézt celou řadu metod postavených na různých technologiích, které jsou používány k sestavování modelů podnikových procesů. Tyto metody však mají společný abstraktní rámec , který vyplývá z postupu návrhu byznys procesu (obr. 2.1).



Obr. 2.1: Postup návrhu byznys procesu

Nejprve je nutné identifikovat, jaké funkce daná organizace či podnik má plnit a za jakým účelem. Hledáme co je vstupem a výstupem těchto funkcí a jak jsou tyto funkce strukturovány. Následuje další krok popisující jak tyto funkce budou zajišťovat transformaci vstupů na výstupy pomocí k tomu určených aktivit a procesů. Nakonec je nutné definovat, čím konkrétně jsou v předchozích krocích definované toky dané a kdo a co bude realizovat specifikované aktivity.

V podstatě tedy existují tři základní přístupy, které se využívají k modelování procesů, a které vychází ze tří základních typů použité abstrakce:

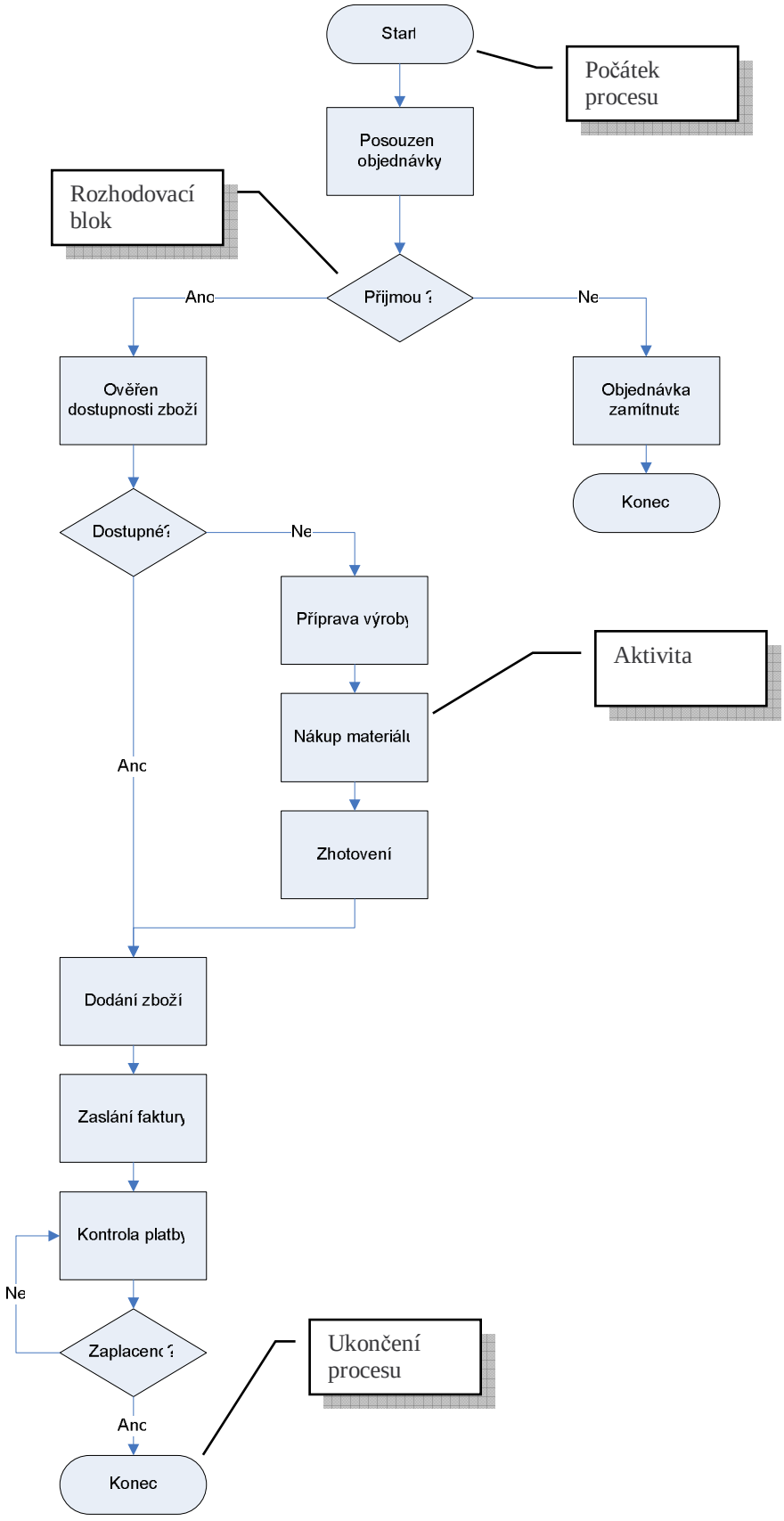
1. *Funkční přístup* zaměřený především na funkce, jejich strukturování, vstupy a výstupy.
2. *Přístup specifikací chování* je zaměřen na řídící aspekt vykonávání procesu cestou stanovení událostí a podmínek, za kterých mohou být jednotlivé aktivity prováděny.
3. *Strukturální přístup* je zaměřen na statický aspekt procesu. Cílem je postihnout entity a zdroje vystupující v procesu včetně jejich atributů, činností (služeb) a vzájemných vazeb.

Všechny moderní metody modelování byznys procesů používají všech těchto tří základních abstrakcí. Liší se pouze v akcentu té či oné stránky modelu. Našim úkolem bude ukazát základní reprezentanty těchto metod a demonstrovat na nich, jakým způsobem jsou jednotlivé abstrakce v dané metodě implementovány. Nejprve si ukážeme způsob, jakým metoda IDEF (Integration DEFinition) definuje funkční model. Následovat bude popis způsobu koordinace aktivit s použitím metody EPC (Event-driven Process Chain). Strukurální přístup budeme demonstrovat na diagramu jazyka UML (Unified Modeling Language). Poněvadž jazyk UML je dnes považován za de facto standard pro modelování nejen softwarových systémů, ukážeme si nakonec kompletní řešení sestavení byznys modelu z pohledu všech tří abstrakcí, a to právě v tomto jazyku.

K účelům demonstrace způsobu použití jednotlivých metod si však nejprve definujme jednoduchý příklad realizace zakázky, který bude sloužit k porovnání jednotlivých přístupů. K jeho specifikaci nejprve použijeme neformální přístup textového popisu. Následně stejný příklad vyjádříme pomocí vývojového diagramu (obr. 2.2).

Příklad 2.1 (*Realizace zakázky*): Mějme dánu firmu, která po obdržení objednávky nejprve prostřednictvím svého obchodního oddělení rozhodne o tom, zda-li je schopna obdržené zboží dodat. Pokud ne, proces je ukončen zamítnutím objednávky. V opačném případě pokračuje proces ověřením, zda-li je zboží k dispozici ve skladu, či je nutné je vyrobit. K účelu vyrobení je nutné zakoupit materiál, připravit výrobu a nakonec požadované zboží zhotovit ve výrobním úseku firmy. Následně je expedicí zboží odesláno objednateli, vystavena a zaslána faktura účetním oddělením, které také kontroluje její zaplacení .

Z textu je patrné, co rozumíme pod pojmem neformální přístup. Je to chybějící přesné definování syntaxe (pravidel jak je proces vyjádřen) a ne zcela jasně daná sémantika (význam použitých pojmů). Z toho vyplývá nejednoznačnost v intuitivním chápání specifikovaného problému. Vývojový diagram, který již používá určitou, všem dobře známou, grafickou notaci (syntaxi), ale může být sestaven různými lidmi různým způsobem a navíc zanedbává celou řadu věcí, které by měly být modelovanému procesu vlastní. Jinými slovy řečeno, vývojový diagram stojí z hlediska formalizace na vyšší úrovni než přirozený jazyk, ale stále však není dostatečně přesný. Vývojový diagram tedy spadá spíše do oblasti semi-formálních popisů. V našem případě budeme i takové prostředky nazývat neformálními.



Obr. 2.2: Realizace zakázky

2.2. Funkční specifikace pomocí IDEF

Výklad

Podstata a historie metody

Metoda IDEF (Integration DEFinition), konkrétně IDEF0, poskytuje modelovací jazyk s danou syntaxí a sémantikou, umožňující vytvořit strukturovanou grafickou reprezentaci systému nebo organizace. Jejím použitím je možné sestavit konsistentní model tvořený popisem funkcí systému, jejich vzájemných vztahů a dat umožňujících tyto funkce integrovat.

Metoda IDEF byla odvozena z graficky orientovaného jazyka SADT (Structured Analysis and Design Technique) na základě požadavků U.S. Air Force. Účelem bylo nalézt prostředek pro analýzu a komunikaci mezi lidmi zaměřenými na zvyšování produktivity výroby. Výsledkem bylo vytvoření celé řady technik, které byly v budoucnu ještě dále rozšířeny. Základem byly následující metody:

1. IDEF0 určená pro účely sestavení funkčního modelu, který strukturovaným způsobem popisuje funkce modelované doménové oblasti.
2. IDEF1 sloužící k sestavení informačního modelu, který reprezentuje strukturu a sémantiku informací.
3. IDEF2 popisující dynamiku systému, tedy jeho chování.

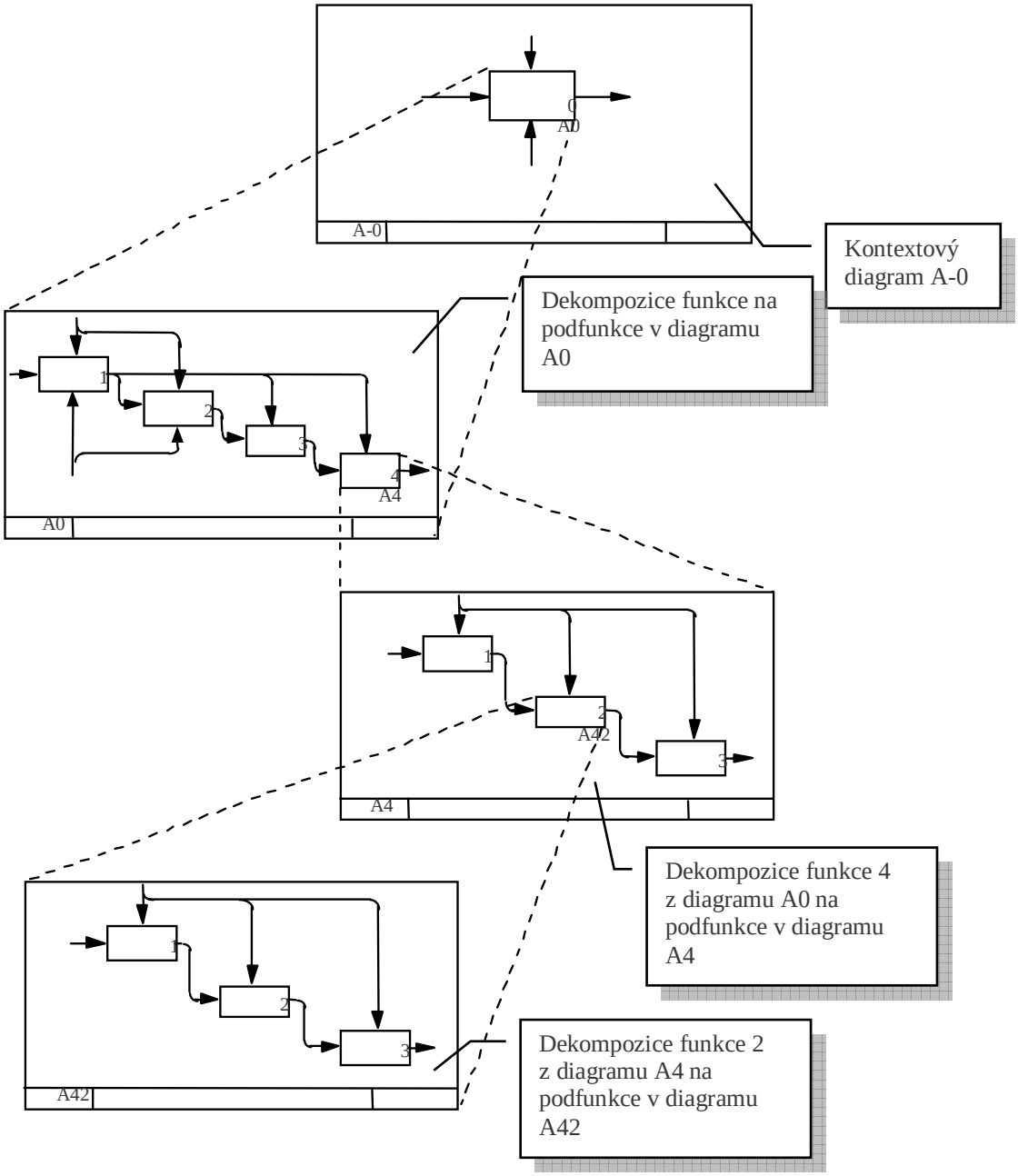
V našem případě se omezíme právě na metodu IDEF0 jako na typického reprezentanta systematického přístupu k analýze funkcí.

Funkční analýza

Účelem použití IDEF0 je vytvoření modelu, který se sestává z hierarchicky uspořádané sady diagramů a textů s přesně vytvořeným systémem vzájemných odkazů popisujícími funkce organizace či podniku. Primárními modelovacími komponentami jsou funkce a data/objekty, které vzájemně tyto funkce propojují. Konkrétně se jedná o následující syntaktické prvky (obr. 2.4):

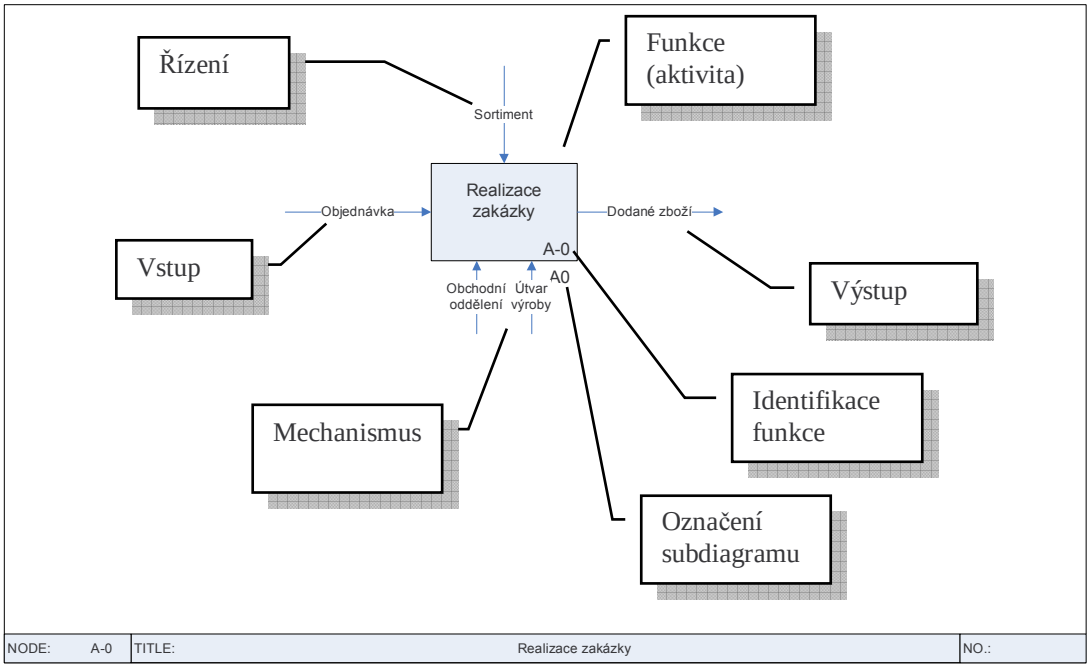
1. Funkce (*Function*) popisující činnost transformující vstup na požadovaný výstup.
2. Vstupem (*Input*) jsou data nebo objekty, které budou funkcí transformovány na výstup.
3. Výstupem (*Output*) rozumíme data nebo objekty produkované funkcí.
4. Řízení (*Control*) je dáno pravidly potřebnými k vytvoření požadovaného výstupu.
5. Mechanismus (*Mechanism*) definuje prostředky nutné k realizaci funkce.

Z anglických názvů jednotlivých toků se pak hovoří o tzv. ICOM (Inputs, Controls, Outputs, Mechanisms), které je vyžadováno každou funkcí. Kromě toho, každá funkce nese své číselné označení (ID) a případně také označení diagramu, ve kterém je funkce pak dále rozpracována do svých dalších podfunkcí (obr. 2.4). Díky tomu je možné vytvářet hierarchii diagramů odpovídající dekompozici funkcí na své podfunkce (strukturovaný přístup). Vrchol této hierarchie je definován tzv. kontextovým diagramem označeným písmenem a číslem 0 (obr. 2.3). Při sestavování diagramů jsou dodržovány zásady jejich řazení ve směru diagonály a diagram by neměl mít méně než tři a více než šest funkcí. Platí zde také důležitá vlastnost těchto diagramů, kdy výstupy dané funkce mohou být vstupem, řízením či mechanismem jiných funkcí. Tímto způsobem jsou definovány vzájemné závislosti mezi funkcemi.



Obr. 2.3: Hierarchie funkcí a jejich diagramů

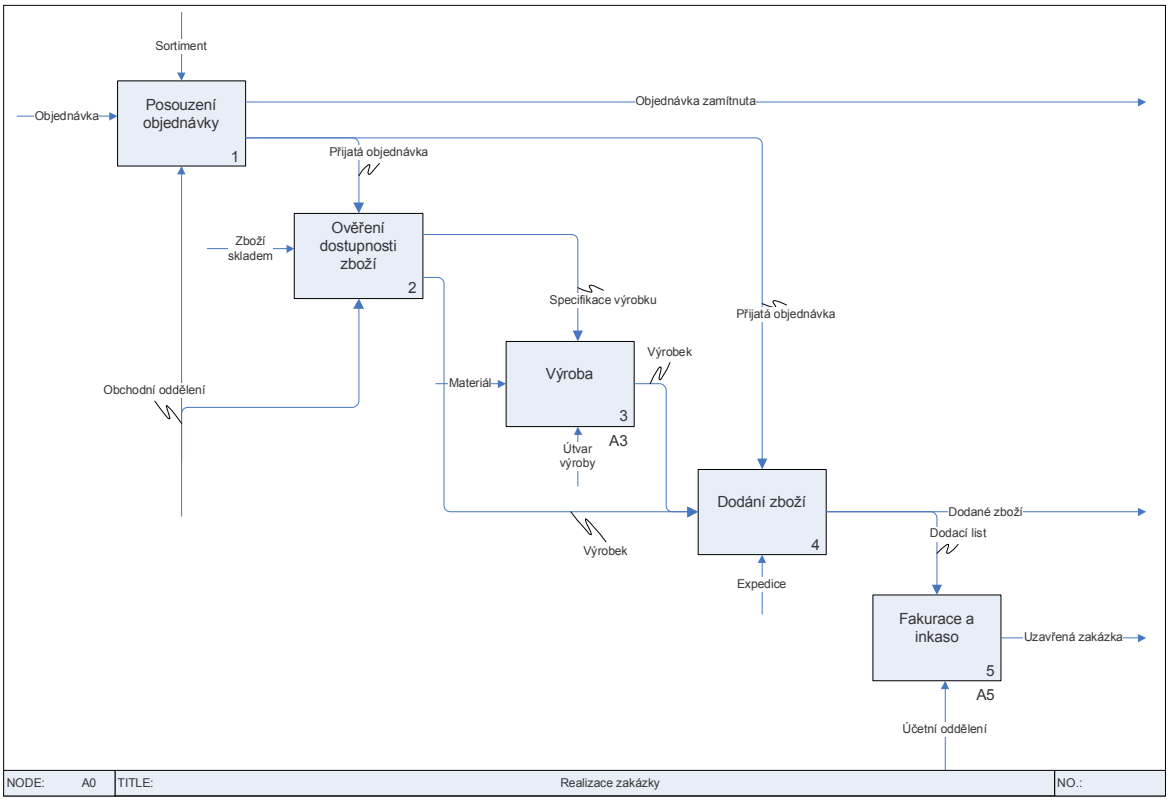
Pokusme se nyní o aplikaci metody IDEF0 na **příklad 2.1** (*Realizace zakázky*) (obr. 2.4). Funkce *Realizace zakázky* má na vstupu objekt *Objednávka*, který bude transformován na objekt(y) *Dodané zboží*. Regulátorem jsou pravidla, kterými se přijetí a následná realizace objednávky řídí. V našem případě je to *Sortiment* zboží poskytovaný firmou. Mechanismem uskutečnění funkce jsou zdroje reprezentované *Obchodním oddělením* a *Útvarem výroby*. Kontextový diagram popisující výše uvedené je vyjádřen pomocí diagramu A-0. Diagram označený jako A0 pak popisuje dekompozici funkce *Realizace zakázky* na další podfunkce (obr. 2.5).



Obr. 2.4: Kontextový diagram realizace zakázky

Realizace zakázky je tvořena pěti dalšími funkcemi. Funkce *Posouzení objednávky* má na svém vstupu objekt *Objednávka* definovaný již v kontextovém diagramu. Regulátorem je zmíněný *Sortiment*, který slouží *Obchodnímu oddělení* k rozhodnutí o tom, zda-li výstupem bude *Objednávka zamítnuta* či *Přijatá objednávka*. Druhý ze zmiňovaných výstupů se pak stává řízením pro funkce *Ověření dostupnosti zboží* a funkce *Dodání zboží*. *Ověření dostupnosti zboží* má na svém vstupu *Zboží skladem*, které je v případě jeho dostupnosti vyskladněno a požadovaný *Výrobek* je předán jako vstup do zmíněné funkce *Dodání zboží*. *Expedice* pak v rámci této funkce a na základě *Objednávky* odešle zboží zákazníkovi (výstup *Dodané zboží*) spolu s *Dodacím listem*. Pokud zboží není k dispozici, je nutné je vyrobit v rámci funkce *Výroba*, která na svém vstupu požaduje *Materiál* a způsob zhotovení zajišťuje *Útvar výroby* dle *Specifikace výrobku*. Výstupem je opět *Výrobek*, který není v tomto případně vyskladněn, ale zhotoven v rámci výroby. Poslední funkce *Fakturace a inkaso* na základě dodacího listu v *Účetním oddělení* zajistí finanční záležitosti a výstup *Uzavřená zakázka* celý proces uzavírá. Důležité je si také uvědomit, že všechny prvky ICOM kontextového diagramu jsou explicitně vyjádřeny také v diagramu A0. *Objednávka* je vstupem, *Sortiment* řízením a *Obchodní oddělení* mechanismem funkce *Posouzení objednávky*. *Útvar výroby* je mechanismem funkce *Výroba* a nakonec výstup *Dodané zboží* je výstupem funkce *Dodání zboží*. To, že ne všechny vstupy, řízení, výstupy a mechanismy jsou znázorněny v kontextovém diagramu vyplývá z faktu, že až díky dekompozici byly tyto nové informace týkající se ICOM identifikovány.

Další strukturování jednotlivých funkcí je znázorněno v diagramu u funkcí *Výroba* (číselná identifikace funkce je 3) a *Fakurace a inkaso* (identifikace 5). Diagramy, které budou popisovat dekompozici těchto funkcí budou nést označení A3 a A5.



Obr. 2.5: Funkční specifikace realizace zakázky

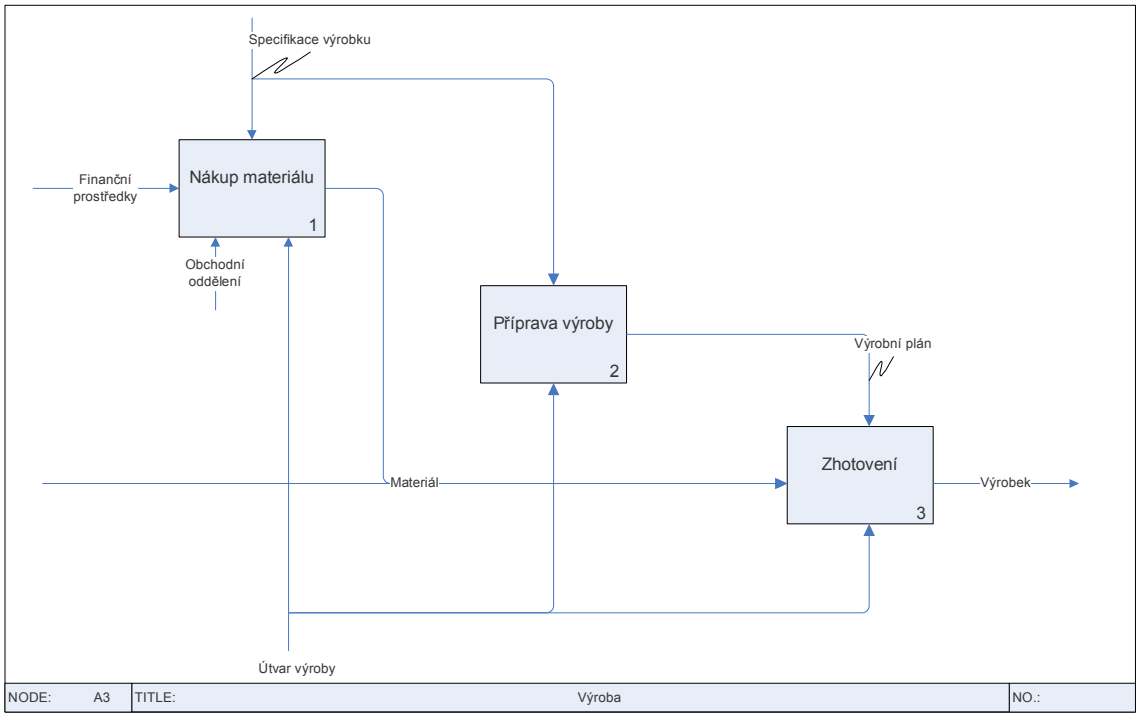
Ukažme si tedy ještě na diagramu A3 rozpracování funkce *Výroba* (obr. 2.6). Tato se sestává z funkcí *Nákup materiálu*, *Příprava výroby* a *Zhotovení*. *Nákup materiálu* ke své realizaci vyžaduje na vstupu *Finanční prostředky* potřebné pro tento nákup a to, co se má zakoupit je řízeno *Specifikací materiálu*. Výstupem je *Materiál*, který vstupuje do funkce *Zhotovení*. Vlastní *Zhotovení* je řízeno *Výrobním plánem*, který je výstupem funkce *Příprava výroby*. Mechanismem uskutečnění všech tří funkcí je *Útvar výroby*. U funkce *Nákup materiálu* pak společně s *Obchodním oddělením*.

Pokud bychom chtěli dále některou z funkcí rozpracovat, pak by musel být založen další diagram s označením začínajícím na A3 a další číslo by označovalo identifikaci rozpracovávané funkce. V případě funkce *Zhotovení* by se tedy jednalo o diagram s označením A33.

❑ Hodnocení metody

Uvedená metoda má celou řadu výhod, ale také své nevýhody, které si na tomto místě rozebereme. K hlavním výhodám patří:

- Metoda je dobře formalizována. Má jednoznačně danou syntaxi a i sémantika je dobře specifikována. Přesto lze u sémantiky nalézt některé neurčitosti v interpretaci, co je skutečně vstupem, mechanismem a řízením. V našem příkladě jsme např. považovali *Specifikaci výrobku* za řízení, ale někteří autoři modelů považují takovou specifikaci za vstup funkce.
- Funkční analýza obsahuje všechny důležité aspekty procesu a díky rozpracovanému způsobu strukturování a přijatým konvencím lze pomocí IDEF0 popisovat i velmi složité (komplexní) procesy.
- Metoda je standardizována na úrovni National Institute of Standards and Technology (USA).



Obr. 2.6: Výrobní proces

Z hlediska toho, co je na této metodě nevýhodné je:

- IDEF0 popisuje pouze funkce a zanedbává skutečný průběh procesu, tedy jak jsou jednotlivé aktivity za sebou řazeny z hlediska času. I když je možné tuto posloupnost z diagramů IDEF0 částečně odvodit, není tento náhled jejich prioritou.
- Komplexní popis procesu lze realizovat s využitím dalších technik (např. již zmíněný IDEF1, IDEF2, IDEF3 atd.), pak se však celá metoda stává náročnou z hlediska jejího zvládnutí a ztrácí se názornost, která je primárním požadavkem modelování obecně.

Závěrem ještě jednou zdůrazněme, že primárním účelem funkční analýzy (v našem případě popsané pomocí IDEF0) je při návrhu byznys procesu (obr. 2.1) nalézt odpověď na otázku „Co?“ v daném podniku či organizaci vlastně chceme popsat.

Shrnutí pojmů 2.2.

Funkční analýza, hierarchická dekompozice funkcí.

Funkce, její vstupy, výstupy, řízení a mechanismy uskutečnění (ICOM).

Otázky 2.2.

1. Jaký je rozdíl mezi vstupem funkce a jejím řízením?
2. Jak jsou označovány diagramy popisující dekompozici vybrané funkce?

Úlohy k řešení 2.2.

Vytvořte pro funkci *Fakturace a inkaso* z procesu *Realizace zakázky* (obr. 2.5) diagram, který specifikuje dekompozici této funkce. Pro zjednodušení této úlohy předpokládejte, že funkce *Fakturace a inkaso* je složena pouze z funkcí *Zaslání faktury* a *Kontrola platby*.

2.3. Specifikace procesu s využitím EPC

Výklad

Podstata a historie metody

V minulé kapitole jsme pomocí IDEF0 specifikovali funkce, které by měl proces plnit. Nyní nastává okamžik, kdy je nutné definovat, jak budou tyto funkce realizovány pomocí aktivit a především jakým způsobem budou tyto aktivity koordinovány z časového pohledu. Jinými slovy vyjádřeno, jaká bude posloupnost těchto aktivit, případně, které z aktivit budou moci být realizovány souběžně (paralelismus).

Metoda EPC (*Event-driven Process Chain*) patří k jedné z nejrozšířenějším především proto, že se stala součástí systémů jako SAP R/3 (ERP/WFM) a ARIS (BPR). Podstata metody, jak vyplývá i z jejího názvu, spočívá v řetězení událostí a aktivit do posloupnosti realizující požadovaný cíl. Z obecného pohledu vykonávání procesu událost definuje vstupní podmínku (*precondition*) uskutečnění aktivity. Ukončení aktivity pak definuje další událost – výstupní podmínku (*postcondition*), na kterou mohou navazovat další aktivity. Z toho vyplývá, že každá aktivita je vymezena dvěmi událostmi a tak je i jednoznačně definován její začátek a konec.

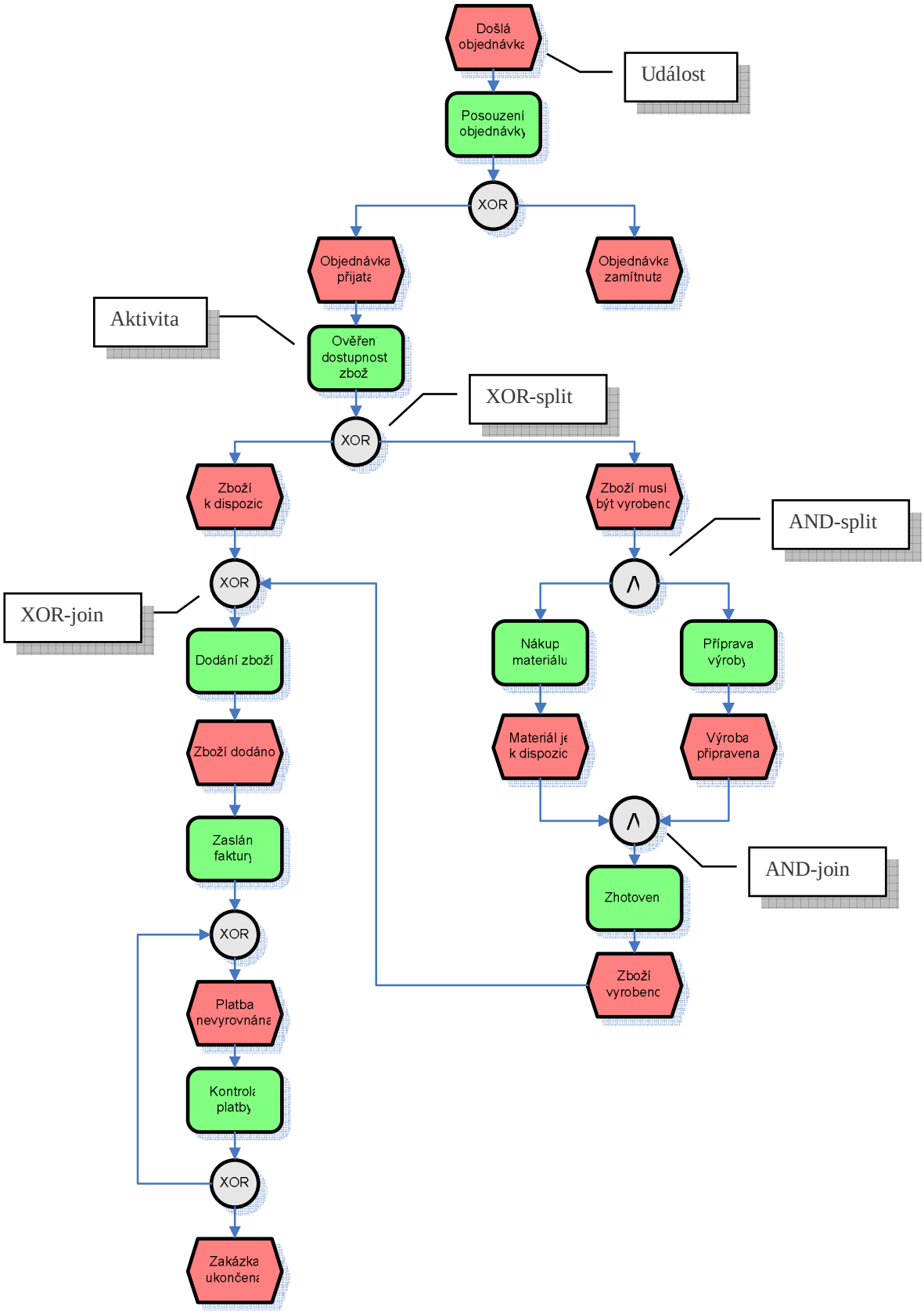
Specifikace řídícího aspektu procesu

Princip událostí a aktivit umožňuje velmi efektivně a nutno dodat i elegantně, srozumitelným způsobem popsat proces. To bylo také primárním cílem autorů (Keller, Nüttgens a Scheer) grafického jazyka, který je v EPC diagramech používán. Popsat procesy na úrovni byznys logiky tak, aby mohl být zvládnut širokou komunitou zabývající se touto problematikou. Byznys proces specifikovaný pomocí EPC diagramu využívá následujících elementů (obr. 2.7):

1. Aktivity (*Activities*), které jsou základními stavebními bloky určují, co má být v rámci procesu vykonáno.
2. Události (*Events*) popisují situace před a/nebo po vykonání aktivity. Aktivity jsou vzájemně propojeny pomocí událostí. Jinak řečeno, nějaká událost může vyjadřovat výstupní podmínku jedné aktivity a současně vstupní podmínku jiné aktivity.
3. Logické spojky (Connectors) se používají ke spojování aktivit a událostí. Tímto způsobem je popsán řídící tok procesu. EPC používá tři typy spojek: $\wedge$ (*AND* – a současně), $\vee$ (*OR* – nebo) a XOR (*exclusive OR* – vzájemně se vylučující nebo).

Pozn.: Poněvadž použití anglických označení spojek se v oblasti modelování byznys procesů stalo faktickým standardem, budeme i my v dalším textu také používat anglických výrazů *AND*, *OR* a *exclusive OR (XOR)*.

Logické spojky mají v popisu procesu dvojí význam. Mohou sloužit k rozdělení toku činností (anglicky *split*) nebo tyto toky naopak slučují (anglicky *join*). V prvém případě má spojka jeden vstup a minimálně dva výstupy, v druhém případě má spojka nejméně dva vstupy a právě jeden výstup. Odtud tedy vyplývá, že budeme používat tzv. *AND-split* pro potřeby vytvoření souběžných toků činností a *AND-join* pro účel synchronizovaného sloučení souběžných toků činností. Význam synchronizované sloučení spočívá v tom, že proces může pokračovat, pokud se oba souběžné toky dostaly až k bodu jejich sloučení. *XOR-split* rozpojuje tok procesu do jedné z možných cest a analogicky *XOR-join* tyto vzájemně se vylučující toky spojuje zpět do jediného. *OR-split* resp. *OR-join* rozpojuje resp. spojuje tok řízení procesu v proměnlivém počtu, tedy je vybrána jedna, druhá nebo taky obě cesty. Pokusme se nyní o aplikaci jazyka EPC na **příklad 2.1** (*Realizace zakázky*) (obr. 2.7).

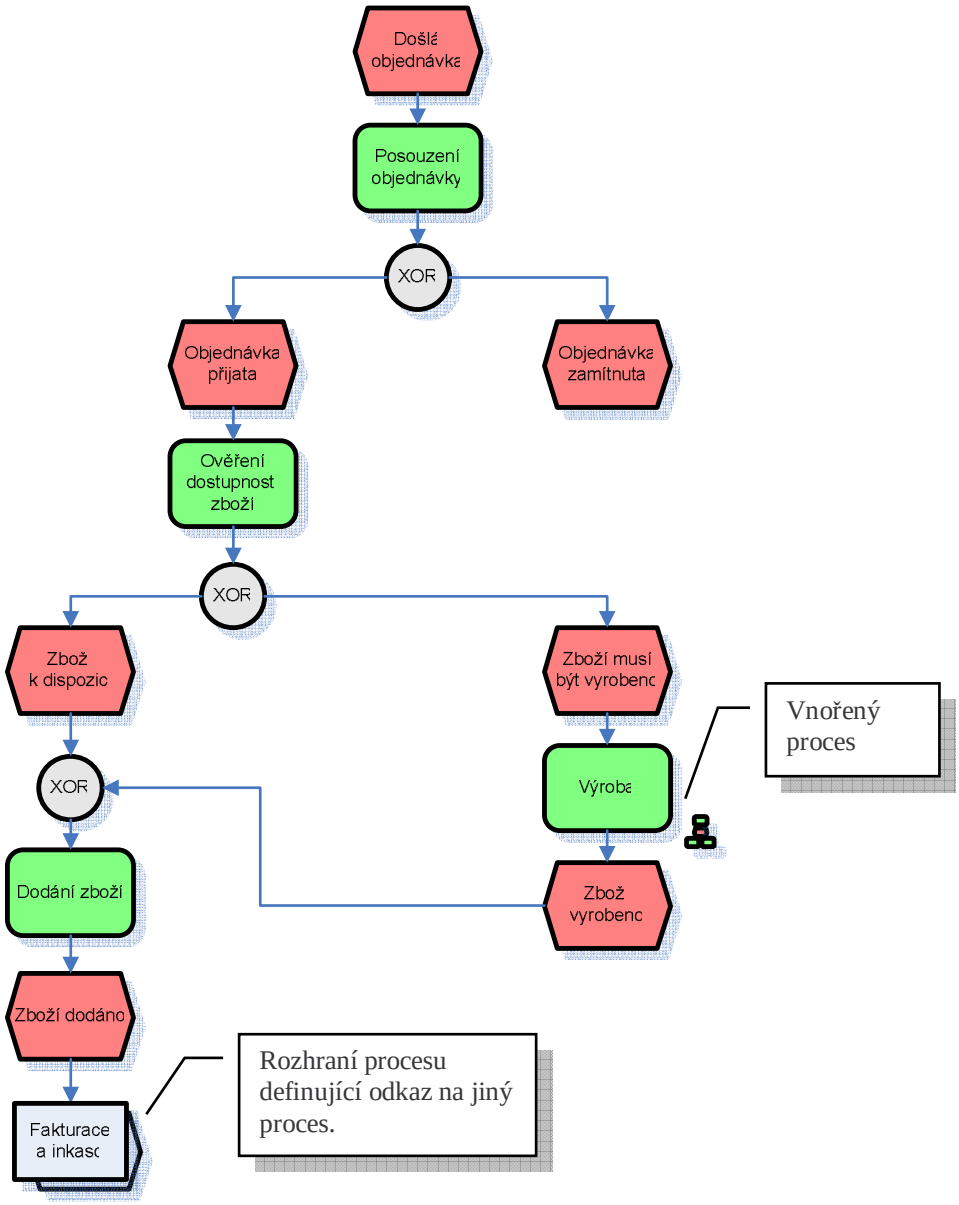


Obr. 2.7: Realizace zakázky pomocí událostmi řízeného procesu

Událost *Došlá objednávka* je vstupní podmínkou aktivity (a spouštěcí událostí celého procesu) *Posouzení objednávky*. Pokud požadované zboží je v nabídce firmy, aktivita generuje událost *Objednávka přijata*. V opačném případě je výstupní událostí *Objednávka zamítnuta*. V případě přijetí objednávky je provedena činnost *Ověření dostupnosti zboží*, která vede na dvě vzájemně se vylučující alternativní výstupní podmínky – *Zboží k dispozici* nebo *Zboží musí být vyrobeno*. V případě uplatnění druhé z uvedených dochází k paralelizaci běhu procesu, kdy jedna větev procesu vede na aktivitu *Nákup materiálu* a druhá na *Přípravu výroby*. První aktivita generuje událost *Materiál je k dispozici*, zatímco druhá vede k události *Výroba připravena*. V okamžiku, kdy jsou obě podmínky splněny dochází ke sloučení toku do činnosti *Zhotovení*. Jinými slovy řečeno, aktivita *Zhotovení* nemůže být zahájena dříve, než je materiál zakoupen a výroba připravena (synchronizace). Událost *Zboží vyrobeno* je spolu se *Zboží k dispozici* alternativní vstupní podmínkou pro aktivitu *Dodání zboží*. Výstupní podmínkou *Dodání zboží* je *Zboží dodáno* reprezentující vstupní podmínku pro aktivitu *Zaslání faktury*. Po aktivitě *Zaslání faktury* dojde v účetnictví firmy k vytvoření situace *Platba nevyrovnána*, která je pak v cyklu ověřována pomocí aktivity *Kontrola platby*. Výstupem této činnosti jsou opět dvě alternativy reprezentované událostmi *Platba nevyrovnána* nebo *Zakázka ukončena*, která je i poslední událostí celého procesu.

Obdobně jako v případě funkční analýzy a metody IDEF je i v případě EPC možné proces strukturovat. Lze libovolnou atomickou aktivitu nahradit podprocesem, který je pak symbolizován malou ikonou EPC diagramu u dané aktivity (obr. 2.8). Je také zřejmé, že procesů může být ve firmě celá řada a pak je možné jejich vzájemné návaznosti vyjádřit pomocí, speciálně k tomuto účelu definovanému, symbolu rozhraní procesu. Hovoří se o tzv. navigaci mezi procesy (obr. 2.8).

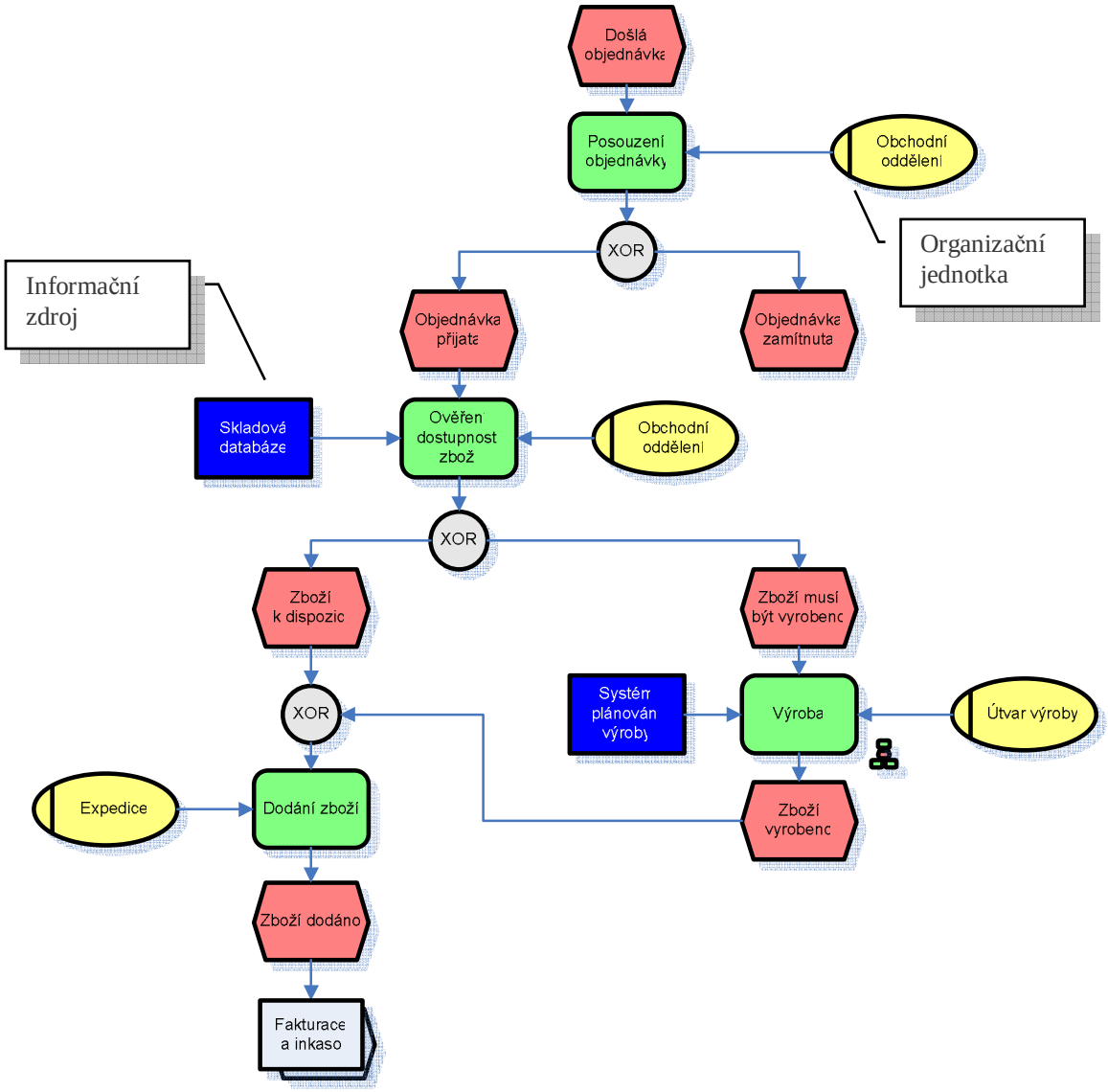
Kromě tohoto standardního EPC diagramu je dnes využíván i tzv. eEPC (extendedEPC) umožňující doplnit do diagramu vykonávání procesu i další informace jako např., která organizační jednotka či role vykonává danou aktivitu, jakou informaci či materiál aktivita vyžaduje a pod. (obr. 2.9).



Obr. 2.8: Strukurovaný přístup u EPC diagramů

V našem případě *Realizace zakázky* jsme určili, že za *Přijetí zakázky* a *Ověření dostupnosti zboží* zodpovídá organizační jednotka *Obchodní oddělení*. Podproces *Výroba* je realizován *Útvarem výroby*, zatímco *Dodání zboží* má na starost *Expedice*. Kromě toho aktivita *Ověření dostupnosti zboží* používá ke své realizaci informační zdroj *Skladová databáze* a *Výroba* používá *Systém plánování výroby*.

Tímto způsobem lze dosáhnout komplexního a úplného popisu procesu. Nevýhodou však je, že dochází k „přetížení“ znázornění procesu velkým množstvím symbolů, které pak zřejmě snižují čitelnost takového diagramu.



Obr. 2.9: Rozšíření EPC diagramu

Hodnocení metody

Stejně jako v případě předchozí metody i v tomto případě se pokusme o základní zhodnocení EPC diagramů. Nesporné výhody spočívají v následujícím:

- Metoda poskytuje jednoduchý princip spojení událostí a aktivit usnadňující vytváření i velmi složitých procesů.
- EPC diagramy jsou základním nástrojem popisu procesů u celé řady komerčně úspěšných a v masovém měřítku nasazovaných softwarových systémů jako jsou SAP R/3 (SAP AG), ARIS (IDS Scheer), LiveModel/Analyst (Intellicorp Inc.) a Microsoft Visio (Microsoft).

Na druhou stranu má tato metoda i své nedostatky:

- Jazyk, který je v EPC používán není formálně definovaný. Syntaxe ani sémantika není důsledně dána (např. *OR-join* nemá jasně dáno, zda-li je či není synchronizovaný apod.), což může vést k nejednoznačnosti ve specifikaci procesů. Situace může být o to horší, že se předpokládá softwarová implementace těchto procesů v podobě workflow v rámci ERP a WFM systémů. Nejednoznačnosti či chybné specifikace mohou pak vést k problémům při

jejich vykonávání. Nemusí být zaručeno dosažení požadovaného koncového stavu procesu z důvodu jeho uvíznutí čekáním na nesplnitelnou podmínku nebo nekonečném opakování nějakého z cyklů obsaženém v procesu.

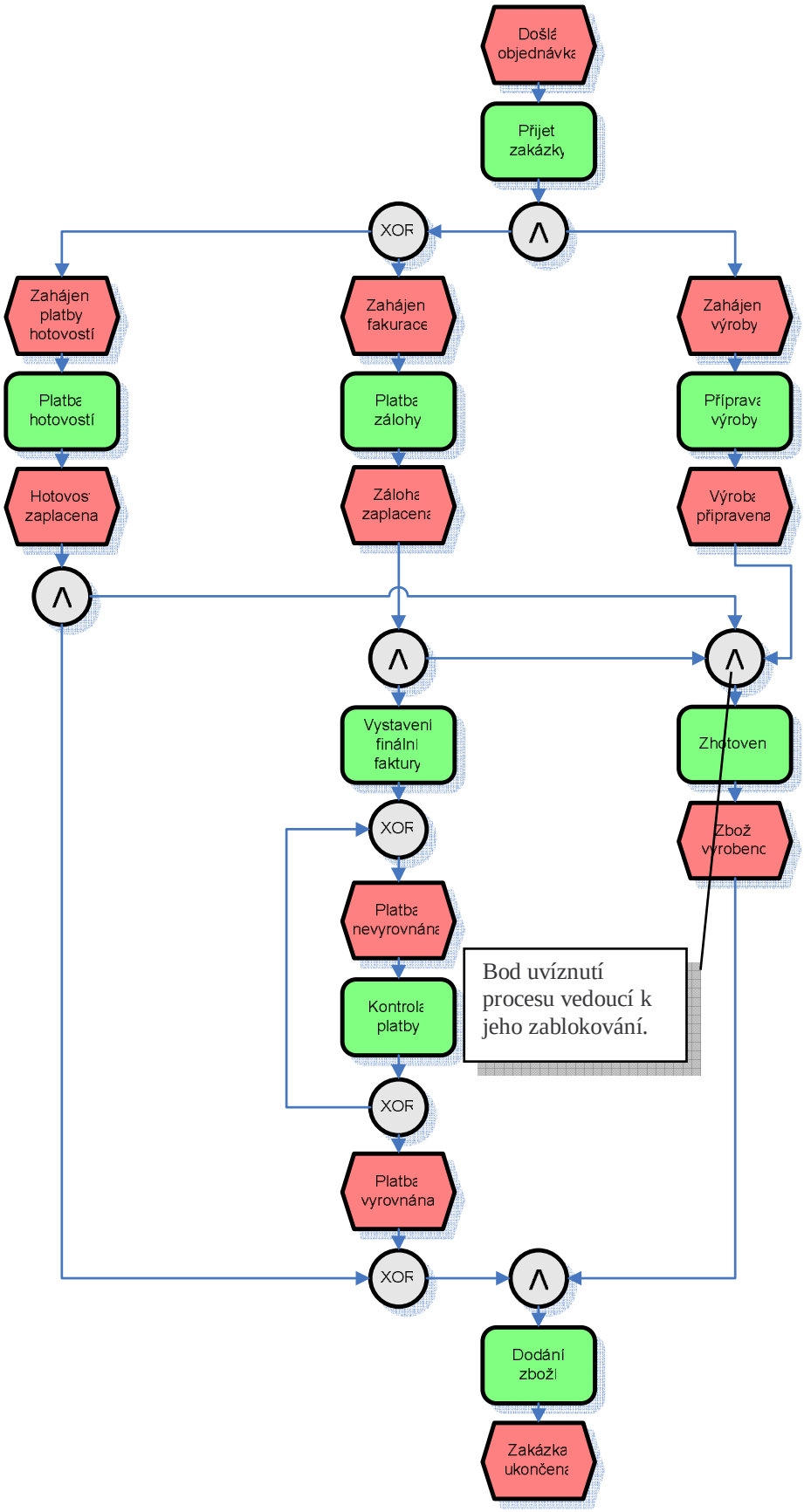
- Chybějící formální specifikace omezuje možnosti použití procesů specifikovaných v EPC v jiných produktech, než výše uvedených. Je tak omezena přenositelnost mezi jednotlivými produkty [5].

Vraťme se v závěru této kapitoly ke zmíněné nízké úrovni formalizace procesu definovaného v EPC diagramech. Snahou tvůrců modelů procesů je navrhnout tyto procesy korektním způsobem. Zatím tento požadavek formulujme v podobě, že je požadováno, aby se každý proces dostal ze svého počátečního stavu do koncového a aby tímto byl i ukončen. Na našem příkladu *Realizace zakázky* to znamená, že proces skončí buď událostí *Objednávka zamítnuta* nebo *Zakázka ukončena*. Je také nepřípustné, aby byla *Zakázka ukončena*, ale bude probíhat např. aktivita *Příprava výroby* v rámci téže instance procesu (**definice 1.5**). Pokusme se o příklad procesu, který nemusí být korektní.

Příklad 2.2 (*Modifikovaná realizace zakázky*). Změňme realizaci zakázky z **příkladu 2.1** takovým způsobem, že vždy uskutečňovaná výroba nemůže být zahájena, pokud nebyla zakázka zaplacena celá hotově nebo nebyla zaplacena alespoň požadovaná záloha. Samotné zboží bude dodáno zákazníkovi až po uhrazení veškerých pohledávek.

Pomocí metody EPC sestavíme diagram způsobem, který je znázorněn na obr. 2.10. Proces je po přijetí objednávky rozdělen do dvou souběžných částí, kde pravá větev popisuje problematiku výroby požadovaného zboží a levá část popisuje vzájemně se vylučující alternativy platby celé zakázky hotovostí nebo cestou zálohy a následné fakturace. *Výroba* pak před aktivitou *Zhotovení* obsahuje *AND-join*, který vyžaduje splnění platby zálohou i hotovostí. A v tomto bodě dochází k porušení požadavku korektní specifikace procesu, neboť jedna z těchto dvou větví určitě nenastane. Zákazník buď platí celou zakázku hotově nebo zálohou a tudíž podmínka daná spojkou AND-join, že všechny vstupující toky musí být ukončeny není splněna. Dochází k uvíznutí procesu.

Při podrobnějším zkoumání procesu lze tuto chybu nalézt, ale ne všechny procesy jsou takto jednoduché a ne vždy jsme upozorněni na to, že ve specifikaci procesu je chyba. Pak nastává okamžik, že správnost navrženého procesu musíme nějakým způsobem ověřit. Nejjednodušší cesta je proces ověřit simulací, ale to znamená projít důsledně všechny možné větve procesu a hlavně mít k dipozici nástroje, které to umožní. Další možností je použití analytických metod, které pomocí určitých algoritmů dokáží taková nekorektní místa najít. K tomuto účelu však musí být použitý jazyk pro modelování procesů formálně definovaný – jednoznačný z hlediska syntaxe i sémantiky. Třetí možností je na základě formální definice modelovacího jazyka používat procesní komponenty s ověřenými vlastnostmi a při jejich sestavování dodržovat pravidla, která zajistí, že výsledný proces bude mít požadované vlastnosti. Ke všem těmto definovaným postupům se dostaneme ve 3. kapitole tohoto studijního materiálu. Zatím se musíme spokojit s tím, že metoda EPC v této podobě neposkytuje uvedené možnosti ověření a je pouze možné v odpovídajícím softwarovém prostředí provést simulaci vytvořených procesů.



Obr. 2.10: Chybně definovaný proces

Metoda EPC díky svému principu řazení aktivit a událostí umožňuje odpovědět na druhou z otázek „Jak?“ bude vlastní proces vykonáván. Zbývá tedy modelovat proces z pohledu „Kým a čím?“ bude zajištěn.

Shrnutí pojmů 2.3.

Události a aktivity.

Logické spojky, rozdělení a spojení toků.

Ověřování správnosti procesu.

Uvíznutí a dosažitelnost cílového stavu.

Otázky 2.3.

1. Jak jsou popsány vstupní a výstupní podmínky aktivity?
2. Co jsou to logické spojky a jaké typy logických spojek používá metoda EPC?
3. Jakým způsobem je v metodě EPC modelován souběžný běh aktivit?

Úlohy k řešení 2.3.

Sestavte model podprocesu *Výroba* (obr. 2.8) tak, aby při nákupu materiálu byla zohledněna možnost kompletního nákupu veškerého požadovaného materiálu, nebo jeho části, případně žádného, pokud je tento ve firmě k dispozici. K tomuto účelu použijte logických spojek *OR-split* a *OR-join*. Souběžně s případným nákupem probíhá *Příprava výroby*. Vlastní *Zhotovení* proběhne až po této přípravě a v okamžiku, kdy je veškerý potřebný materiál k dispozici.

2.4. Strukturální modelování pomocí objektově orientované přístupu

Výklad

Podstata objektově orientovaného modelování

Posledním krokem specifikace byznys procesu je určení kým a čím bude proces realizován. V úvodní kapitole jsem si definovali pojmy *role* a *zdroj* (**definice 1.7** a **definice 1.8**), které vystupují jako aktivní prvky vykonání procesu. Tyto prvky, které proces realizují mohou být lidé nebo stroje. Kromě těchto aktivních prvků však proces obsahuje celou řadu dalších položek, které jsou aktivitami procesu spotřebovávány, modifikovány či vytvářeny. Může se jednat o data, materiál, dokumenty, produkty a podobně. Souhrně můžeme aktivně vystupující prvky a uvedené pasivní položky nazvat *objekty* a využít tak terminologii vlastní moderním objektově orientovaným metodám používaným v oblasti softwarového inženýrství. Poněvadž tyto metody mají dnes mnohem širší využití a i my se budeme v dalším textu věnovat jedné z nich – jazyku UML (Unified Modeling Language), musíme si na tomto místě definovat nové pojmy, se kterými budeme dále pracovat.

Definice 2.1 (*Objekt*): Objekt je entita s jasně definovanou hranicí a identitou zahrnující její chování a stavy.

Definice 2.2 (*Objektově orientovaná metoda*): Objektově orientovaná metoda je promyšlený postup popisu systému, jehož struktura je dána množinou propojených objektů, které prostřednictvím vzájemné komunikace (interakce) definují výsledné chování celého systému.

Právě popis objektů a jejich vzájemných propojení je předmětem strukturálního modelování byznys procesu. Tato struktura se zjevně v čase mění tak, jak se při vykonávání procesu objevují nové objekty a zanikají objekty původní. Hovoříme tedy o tzv. snímcích struktury, které zaznamenávají konfiguraci modelovaných objektů v daném časovém okamžiku. Poněvadž však může být takových snímků příliš mnoho, je nutné tyto konfigurace zobecnit a zajistit tak popis statické struktury platné pro všechny možné situace, které mohou nastat. K tomu slouží tzv. diagram tříd popisující obecně platné třídy objektů a jejich vzájemné relace nezávisle na čase. Diagram objektů pak popisuje výše zmíněné konfigurace konkrétních objektů kompatibilních s odpovídajícím diagramem tříd. Před dalším výkladem si však ještě uveďme definici nového pojmu, kterým je *třída objektu*.

Definice 2.3 (*Třída objektu*): Třídou rozumíme popis množiny objektů mající společnou strukturu, chování, vztahy a sémantiku.

Objekt může být instancí (konkretizací) právě jedné třídy (stejně jako např. strojírenský výrobek může být vyroben pouze podle jediné výrobní dokumentace). Jména tříd volíme tak, aby byla součástí slovníku popisujícího doménovou oblast. Pojmy *instance třídy* a *objekt* můžeme zaměňovat, neboť vyjadřují v tomto kontextu totéž.

Objektový model

Ukážeme si nyní podle jakých pravidel vytvoříme strukturální model byznys procesu s využitím diagramu tříd, který zavádí jazyk UML. Tento diagram tříd se sestává z následujících elementů (obr. 2.11):

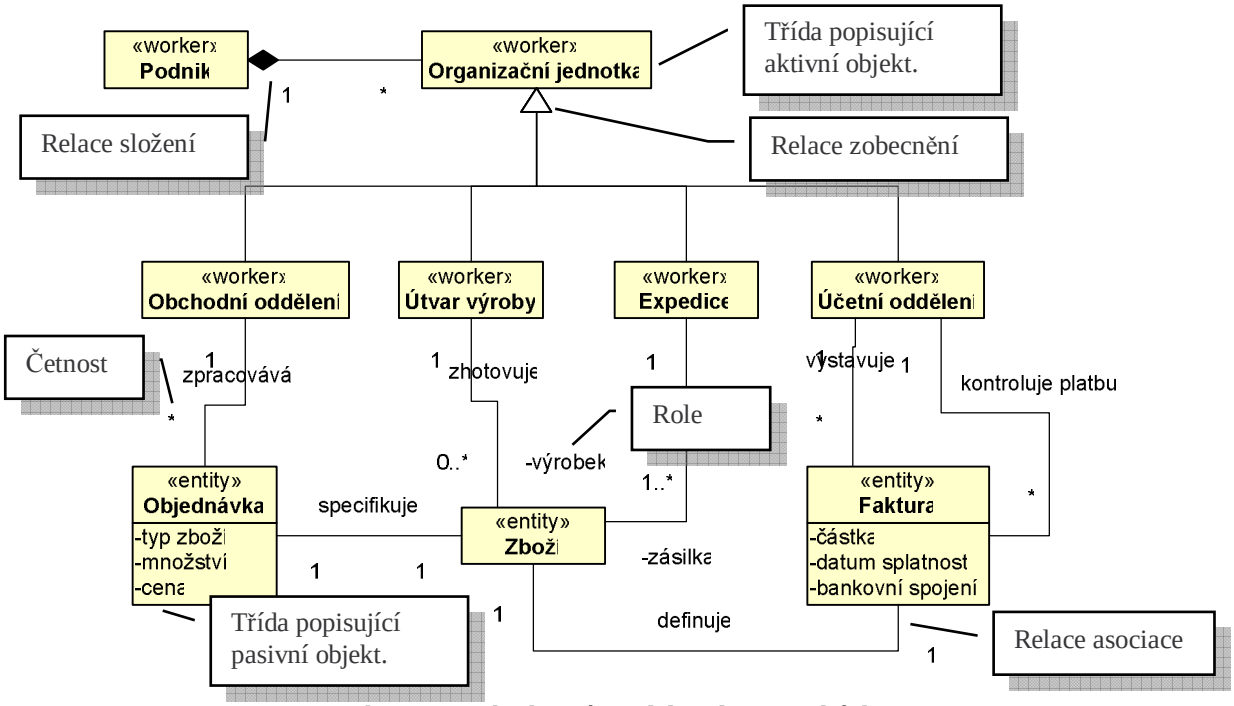
1. Třídy popisující aktivní (*worker*) a pasivní (*entity*) objekty definující strukturu byznys modelu.
2. Relace mezi třídami specifikující cestu, jak mohou objekty mezi sebou komunikovat.

Diagram tříd zobecňující všechny možné konfigurace objektů procesu řadí relace mezi třídami do následujících základních tří skupin:

- *Asociace* popisuje skupinu spojení (mezi objekty) mající společnou strukturu a sémantiku. Vztah mezi asociací a spojením je analogický vztahu mezi třídou a objektem. Jinými slovy řečeno, jedná se o dvousměrné propojení mezi třídami popisující množinu potenciálních spojení a mezi instancemi asociovaných tříd, stejně jako třída popisuje množinu svých potenciálních objektů.
- *Složení* popisující vztah mezi celkem a jeho částmi, kde některé objekty definují komponenty jejichž složením vzniká celek reprezentovaný jiným objektem.
- *Zobecnění* (generalizace) je taxonomický vztah mezi obecnějším elementem a jeho více specikovaným elementem, který je plně konzistentní s prvním z uvedených a k jeho specifikaci přidává další konkretizující informaci.

Každá relace pak na svém připojení ke třídě definuje, v jaké roli daný objekt vystupuje vůči druhému a jaká je jeho četnost (násobnost).

Dokumentujme si výše uvedené na **příkladu 2.1**. Cílem bude sestavit model z hlediska jeho strukturálních vlastností, z pohledu v něm obsažených tříd objektů a jejich vzájemných relací (obr. 2.11).



Obr. 2.11: Objektový model realizace zakázky

Třída *Podnik* se skládá obecně z několika (symbol *) *Organizačních jednotek*. Tyto jsou konkretizovány do *Obchodního oddělení*, *Útvaru výroby*, *Expedice* a *Účetního oddělení*. *Obchodní oddělení zpracovává Objednávku* jejíž struktura je dána *typem zboží*, *množstvím* a *cenou*. *Objednávka specifikuje Zboží*, které *zhotovuje Útvar výroby*. *Zboží* vystupuje v roli *výrobku* vůči *Útvaru výroby* a v roli *zásilky* vůči *Expedici*. Vyrobené *Zboží definuje Fakturu*, která je vystavena *Účetním oddělením*. *Účetní oddělení* také proti této *Faktuře kontroluje platbu*. Skutečně takto definovaný diagram tříd definuje všechny možné konfigurace objektů vystupujících v jednotlivých instancích procesu realizace zakázky. Dokladuje to např. i četnost 0..* u relace mezi *Úsekem výroby* a *Zbožím*, protože pokud je výrobek dostupný na skladě, není vyroben a uplatňuje se v četnosti uvedená 0. Jinými slovy vyjádřeno, v takové situaci *Útvar výroby* nevyrábí žádné *Zboží*.

Vzhledem k tomu, že diagram tříd je pouze jeden z mnoha, který poskytuje jazyk UML pro účely byznys modelování, popíšeme si tento jazyk v následující samostatné kapitole včetně hodnocení jeho pozitivních i negativních stránek.

Shrnutí pojmů 2.4.

Strukurální modelování.

Objekt, třída a jejich vztahy.

Diagram tříd.

Otázky 2.4.

1. Definujte pojmy objekt a třída objektu.
2. Jaké typy relací mezi třídami zavádí jazyk UML?

Úlohy k řešení 2.4.

Sestavte diagram tříd tvořený třídami *Obchodní oddělení*, *Útvar výroby*, *Výrobek* a *Materiál*. Popište situaci, ve které *Obchodní oddělení* nakupuje *Materiál*, který vyžaduje *Výrobek* ke svému zhotovení. *Útvar výroby* pak zhotovuje požadovaný *Výrobek* a spotřebovává *Materiál* k tomu určený.

2.5. Byznys modelování pomocí UML

Výklad

Jazyk UML

Jazyk UML byl primárně navržen svými autory (Booch, Rumbaugh a Jacobson) za účelem sjednocení různých přístupů při vytváření specifikací požadovaných v rámci procesu tvorby softwarového produktu. V dnešní době se však stal univerzálním standardizovaným jazykem pro vytvoření „výkresové dokumentace“ libovolných systémů. Specifikace jazyka obsahuje celou řadu diagramů, které popisují modelovaný systém ze všech možných náhledů a abstrakcí (viz kap. 2.1). Z hlediska použití jazyka UML pro potřeby byznys modelování se jedná především o tyto následující diagramy:

1. Diagram případů (scénářů) užití určený k popisu a analýze funkcí modelovaného systému.
2. Dynamický náhled popisující chování je vyjádřen v diagramu aktivit.
3. Logický (strukturální) náhled využívá v minulé kapitole popsaný diagram tříd.

Kromě těchto diagramů poskytuje UML i další, včetně těch, které se týkají fyzické realizace softwarových systémů. Tyto však pro nás nejsou důležité a v případě zájmu čtenáře je možné se odkázat na celou řadu dostupných publikací na toto téma např. [2].

Pokusme se v následujícím textu o kompletní popis byznys modelu z příkladu 2.1 pomocí jazyka UML, a to pomocí tří výše uvedených diagramů.

Diagramy případů užití

Funkční specifikace je v jazyce UML řešena prostřednictvím diagramů případu užití (*Use Case Diagram*). Případ (scénář) užití je pojem, který je v rámci byznys modelování definován následujícím způsobem.

Definice 2.4 (*Případ užití*): Případ užití je posloupnost akcí, které podnik či organizace realizuje v interakci se specifickými aktéry s cílem vytvořit výsledek požadované hodnoty.

V podstatě je tedy případ užití synonymem pro byznys proces s jedinou výjimkou, a tou je zahrnutí tzv. aktéra. Tím jsou veškeré aktivní objekty (zákaznící, dodavatelé, partneři, kolegové atd.) stojící vně daného procesu.

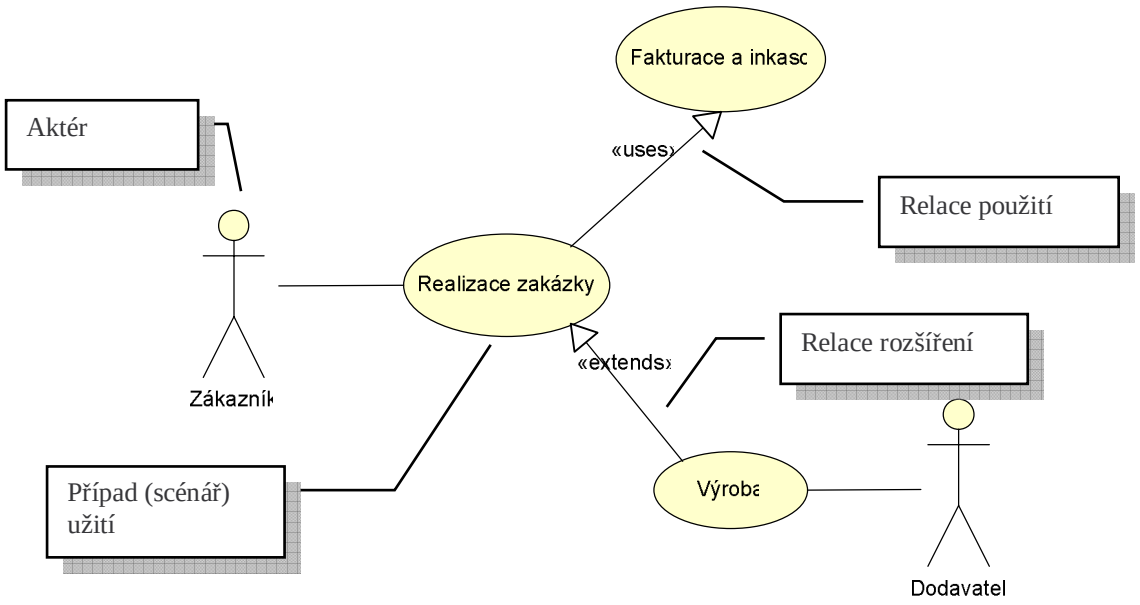
Definice 2.5 (*Aktér*): Aktér reprezentuje někoho nebo něco, co stojí mimo byznys proces specifikovaný daným případem užití.

Primárním účelem diagramů případů užití je tak dokumentovat interakce mezi službami, které jsou podnikem či organizací poskytovány a těmi, kterými jsou tyto služby požadovány. Opět se tak dostáváme k tomu, co již bylo uvedeno dříve, že takto vytvořený model identifikuje, co je vlastně účelem podnikání dané organizace a jaké funkce nabízí svému okolí.

Diagram případů užití používá následujících elementů ke svému sestavení (obr. 2.12):

1. Případy užití (*Use Cases*), které identifikují funkce realizované byznys procesy.
2. Aktéry (*Actors*) popisující externí objekty vstupující do interace se specifikovanými procesy.

Kromě toho, lze mezi procesy definovat relace *rozšíření* (*extends*) a *použití* (*uses*). První z uvedených typů relací popisuje situaci, kdy jeden případ užití rozšiřuje jiný a doplňuje tak scénář realizace procesu o další případné aktivity. Použití pak deklaruje nutnost zahrnout do vykonávání procesu proces jiný, vložený.



Obr. 2.12: Diagram případu užití realizace zakázky

Aktér *Zákazník* vstupuje do procesu *Realizace zakázky*, který může být v případě nedostupnosti zboží rozšířen o proces identifikovaný případem užití *Výroba*. Tento vstupuje do interakce s aktérem *Dodavatel*, který zodpovídá za poskytnutí materiálu požadovaného k výrobě. *Realizace zakázky* používá proces *Fakurace a inkaso* k finančnímu zajištění celé zakázky.

Diagramy případů užití explicitně identifikují procesy a jejich okolí. Neříkají však nic o tom, jak tyto procesy budou realizovány. K tomu slouží další z výše uvedených diagramů – diagram aktivit.

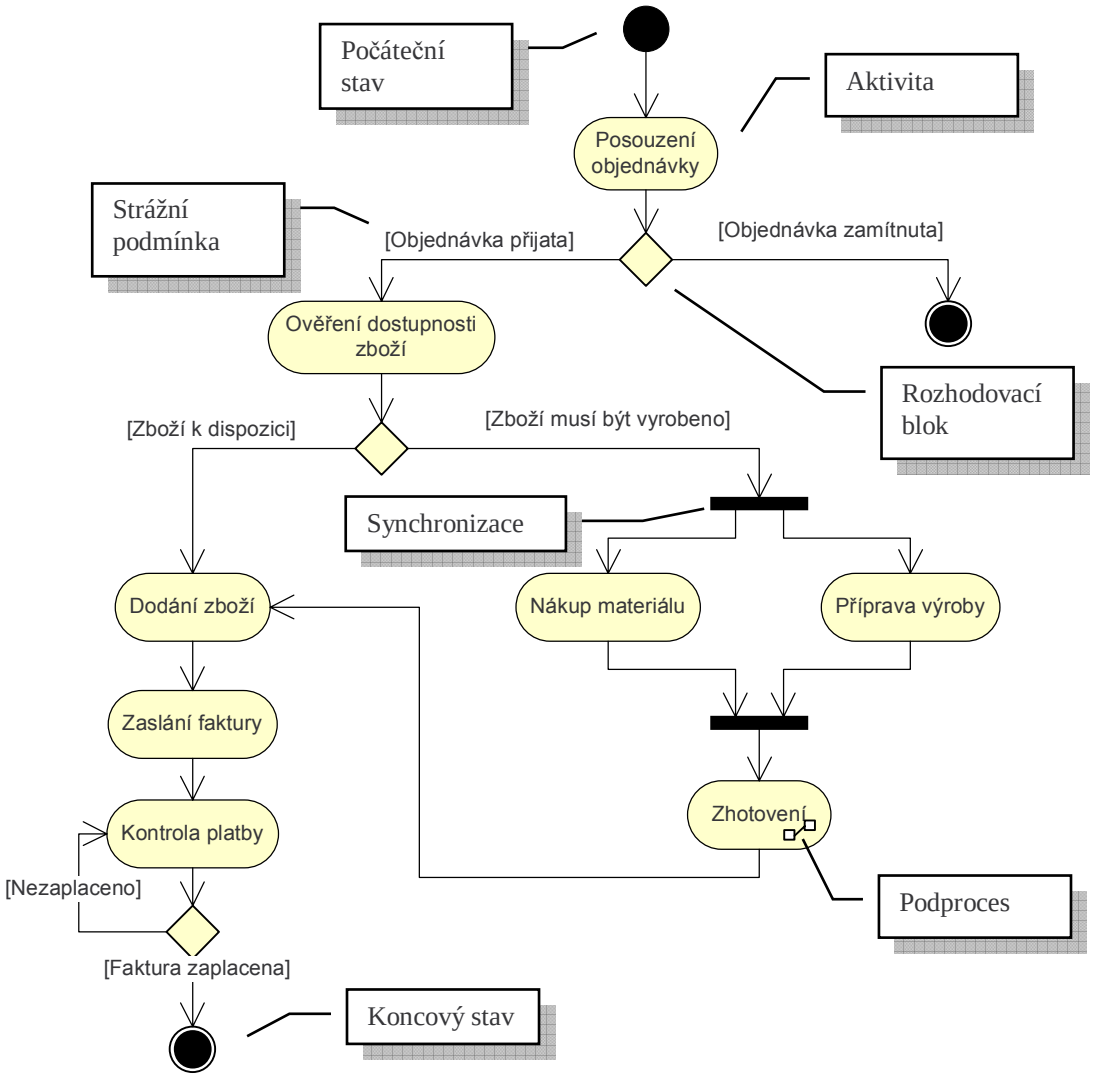
Diagram aktivit

Diagram aktivit popisuje toky činností pomocí aktivit reprezentujících *(akční) stavy* a *přechody* mezi nimi. Přechod mezi dvěmi stavy je realizován cestou ukončení předchozího stavu. Jinými slovy řečeno, přechod je implementován interním mechanismem ukončení činností vázaných na daný stav. Dalším účelem diagramu aktivit je definovat, kdo či který objekt zodpovídá za danou aktivitu, případně jaké objekty jsou aktivitami vytvářeny, spotřebovávány nebo modifikovány. Tímto způsobem jsou do jednoho diagramu promítnuty nejen toky řízení, ale také datové toky. Diagram aktivit tak zahrnuje informaci popsanou v rámci strukturálního modelování, v případě jazyka UML se jedná o, v předchozí kapitole zmíněný, diagram tříd.

Základní elementy, ze kterých je diagram aktivit sestaven jsou následující (obr. 2.13):

1. *Aktivita* reprezentuje vykonání atomické (dále nedělitelné) činnosti. V případě, že je aktivita strukturována do dalšího diagramu aktvit je symbol akčního stavu označen speciální, k tomuto účelu, definovanou ikonou.
2. *Startovací a ukončovací symboly* explicitně určují počáteční a koncový stav procesu.
3. *Rozhodovací blok* využívá definovaných tzv. strážních podmínek (*guards*) k větvení toku činností. Stejný symbol je možné použít ke sloučení těchto toků. Rozhodovací blok je analogický k logickým spojkám *XOR* používaným v EPC diagramech.
4. *Synchronizace* definují místa vytvoření a sloučení souběžných toků. Analogií jsou v tomto případě *AND* spojky používané v EPC.

Všechny tyto elementy jsou propojeny pomocí přechodů určujících řídící tok vykonávání činností.

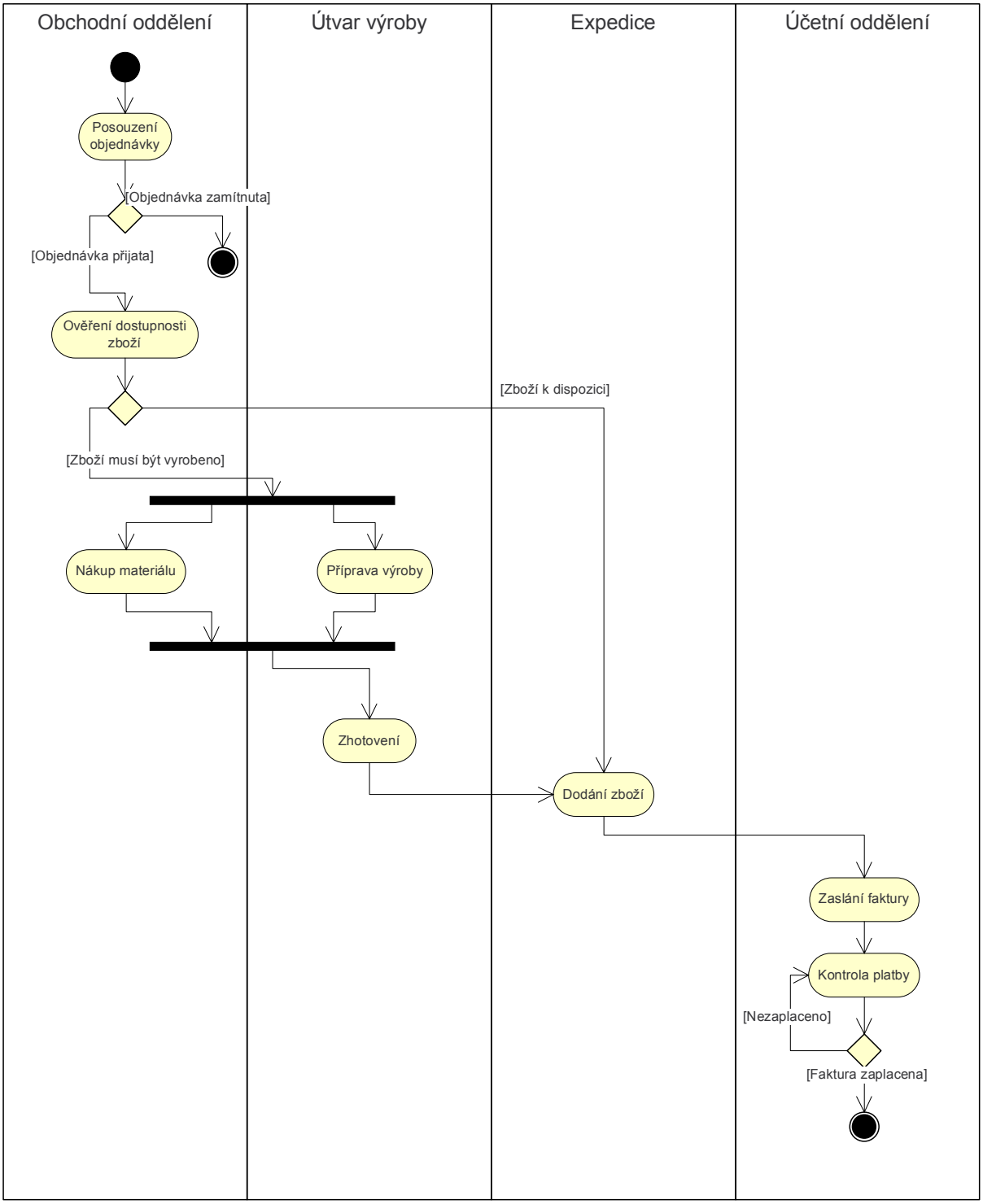


Obr. 2.13: Diagram aktivit

V případě realizace zakázky je první aktivitou *Posouzení objednávky*. Následně je proces větven dle splnění odpovídajících podmínek (*Objednávka přijata* a *Objednávka zamítnuta*). V případě zamítnutí objednávky proces končí. V opačné situaci je *Ověřena dostupnost zboží* s následným větvením podle výsledku této aktivity. V případě nutnosti zboží vyrobit dojde k vytvoření dvou souběžných toků činností reprezentovaných *Nákupem materiálu* a *Přípravou výroby*. V okamžiku, kdy jsou obě aktivity ukončeny, proběhne *Zhotovení*, které by mělo být popsáno v dalším diagramu aktivit podrobně specifikující tento podproces. Následně může být realizováno *Dodání zboží*. Po *Dodání zboží* následuje *Zaslání faktury* a v cyklu realizována *Kontrola platby*, dokud není pohledávka vyrovnána. Pak je proces definitivně ukončen.

Za krátkou poznámku stojí, že na rozdíl od EPC, kde každá aktivita a událost může mít maximálně jeden vstup a jeden výstup, aktivity v UML mohou slučovat více vstupů (viz *Dodání zboží*). Není tedy nutné používat *rozhodovacího bloku* ke sloučení alternativních toků. Lze tím sice dosáhnout úspornějšího grafického vyjadřování, ale z hlediska „dobrého strukturování“ se doporučuje používat raději explicitně vyjádřeného sloučení.

V úvodu této podkapitoly jsme deklarovali, že v diagramech aktivit je možné vyjádřit i zodpovědnost zdrojů za provádění jednotlivých činností. K tomuto účelu slouží tzv. „plavecké dráhy“ (*swimlanes*), které v diagramu obsahují právě ty aktivity, za které daný zdroj zodpovídá (obr. 2.14).

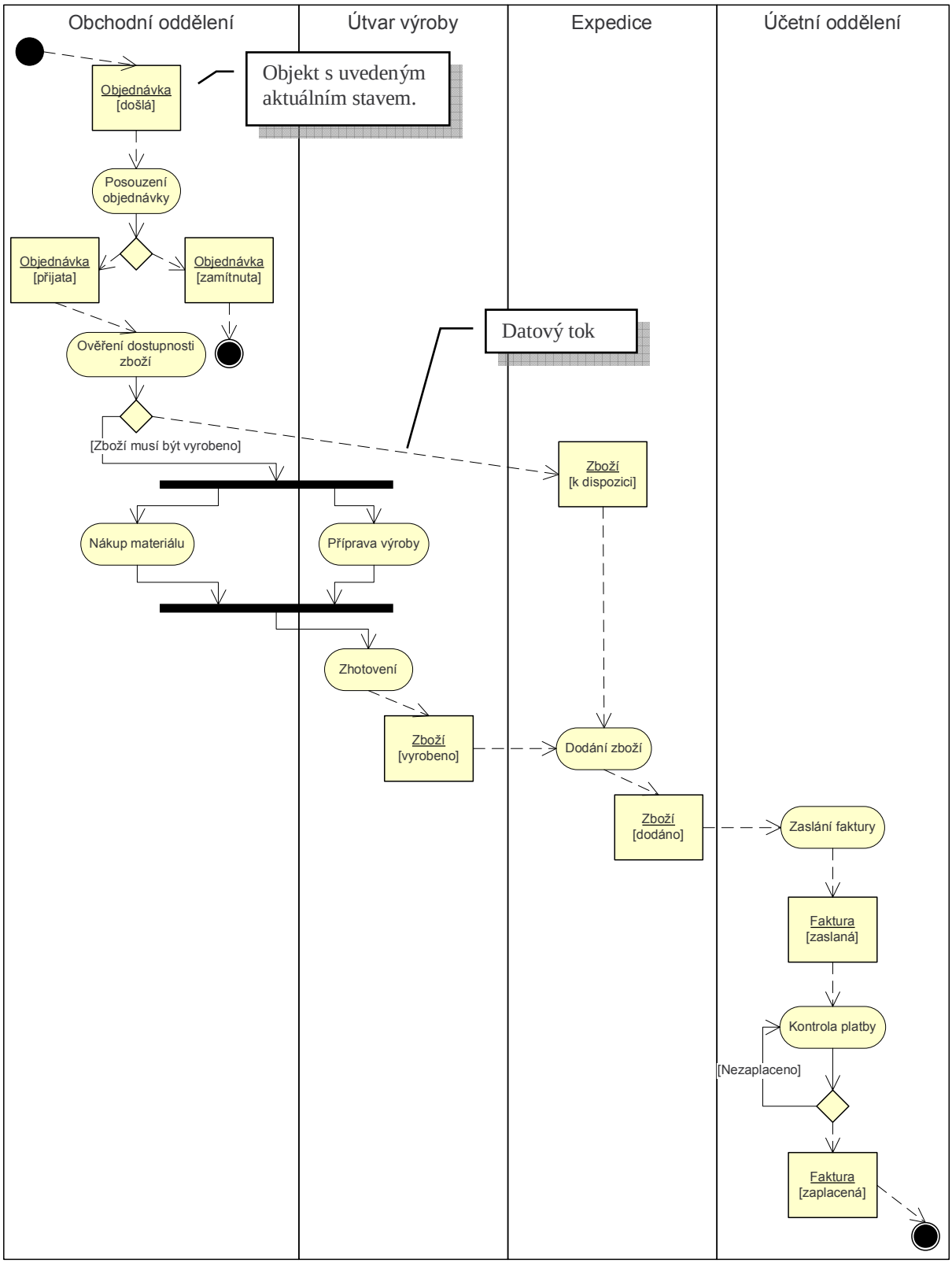


Obr. 2.14: Zodpovědnosti v diagramu aktivit

Z obrázku tedy vyplývá, že *Obchodní oddělení* zodpovídá za aktivity *Posouzení objednávky*, *Ověření dostupnosti zboží* a *Nákup materiálu*. *Útvar výroby* má v kompetenci *Přípravu výroby* a *Zhotovení*, zatímco *Expedice Dodání zboží*. *Účetní oddělení* zodpovídá za *Zaslání faktury* a *Kontrolu platby*.

Kromě toku řízení je možné v diagramu aktivit také popsat datový tok pomocí objektů, které vystupují v daném procesu. Každý objekt má definovány své stavy měnící se v průběhu vykonávání

jednotlivých aktivit (obr. 2.15). Objekty a jejich stavy tak vystupují v roli vstupních a výstupních podmínek těchto aktivit.



Obr. 2.15: Toky objektů a jejich stavy

V procesu realizace zakázky můžeme modelovat objekt *Objednávka*, který se na počátku nachází ve stavu *došlá*. Podle výsledku posouzení je stav změněn na *přijata* nebo *zamítnuta*. Obdobně jsou do diagramu aktivit zahrnuty objekty *Zboží* a *Faktura* včetně jejich stavů. Diagram aktivit se tak stává integrujícím vyjádřením procesu. Nevýhodou však je, že v případě složitých procesů, stejně jako u eEPC, je problémem zohlednit vše podstatné při dodržení čitelnosti takových diagramů.

Před závěrečným hodnocením metody si ještě připomeňme, že součástí byznys modelu je i strukturální specifikace vyjádřená pomocí diagramu tříd. Tento jsme si již ukázali v předchozí kapitole (obr. 2.11) a nebudeme se již o něm zmiňovat.

❑ Hodnocení metody

V závěru této kapitoly se opět, jako v předchozích případech, pokusme o základní hodnocení kladných i záporných vlastností jazyka UML. Hlavní výhody lze shrnout do následujícího:

- Jazyk UML poskytuje širokou škálu diagramů umožňujících, díky různým typům použitých abstrakcí, kompletní popis i velmi složitých byznys procesů.
- Notace jazyka UML je standardizována a je součástí velké řady softwarových produktů určených k modelování systémů. Není zanedbatelnou výhodou, že některé z těchto produktů jsou volně k dispozici.
- Přijetí jazyka UML za de facto standard při vývoji softwarových systémů zajišťuje návaznost vytvořených byznys modelů na specifikaci požadavků a vlastní návrh implementace informačních systémů. Komunity byznys orientovaných odborníků a softwarových inženýrů tak mohou bez zásadních problémů komunikovat pomocí společného, a především všem srozumitelného, jazyka.

Jazyka UML má zjevně i své nevýhody, které jsou velmi podobné těm, které jsou vlastní i metodě EPC:

- I když autoři jazyka a celá řada organizací pracujících na jeho standardizaci se snaží o formalizaci UML, zatím nelze tento jazyk považovat za formálně definovaný a mohou tak nastat problémy s odhalováním chyb ve specifikovaných procesech.

Je však třeba, i přes výše zmíněnou výtku, doplnit, že specifikace jazyka UML se stále vyvíjí s cílem zajistit jeho jednoznačnost z hlediska syntaxe i sématiky. Právě jeho propojení s vývojem software si tuto formalizaci vynucuje. Velkým pokrokem bylo vytvoření meta-modelu jazyka UML, který popisuje samotný modelovací jazyk a je tak možné zajistit sdílení vytvořených modelů mezi různými softwarovými nástroji.

Shrnutí pojmů 2.5.

Jazyk UML.

Případy užití a jejich diagramy.

Diagramy aktivit a jejich základní elementy.

Modelování zodpovědností a datových toků.

Otázky 2.5.

1. Jak je definován případ užití a jaký je jejich účel?
2. Kdo nebo co nazýváme aktérem procesu?

3. Jakým způsobem je řešen problém synchronizace souběžných toků činností?

Úlohy k řešení 2.5.

Navrhněte diagram aktivit účetního oddělení, ve kterém účetní nejprve vystaví fakturu a následně ji odešle zákazníkovi. V případě, že ale celková částka uvedená na faktuře je větší než 5000Kč, musí před odesláním fakturu autorizovat vedoucí účetního oddělení.

Korespondenční úloha 2.5.

Vytvořte pomocí jazyka UML modely hlavních byznys procesů videopůjčovny. Bude se jednat o následující tři procesy: *registrace nového zákazníka*, *vypůjčení* a *navrácení* videokazet příp. DVD nosičů.

2.6. Specifikace meta-modelu

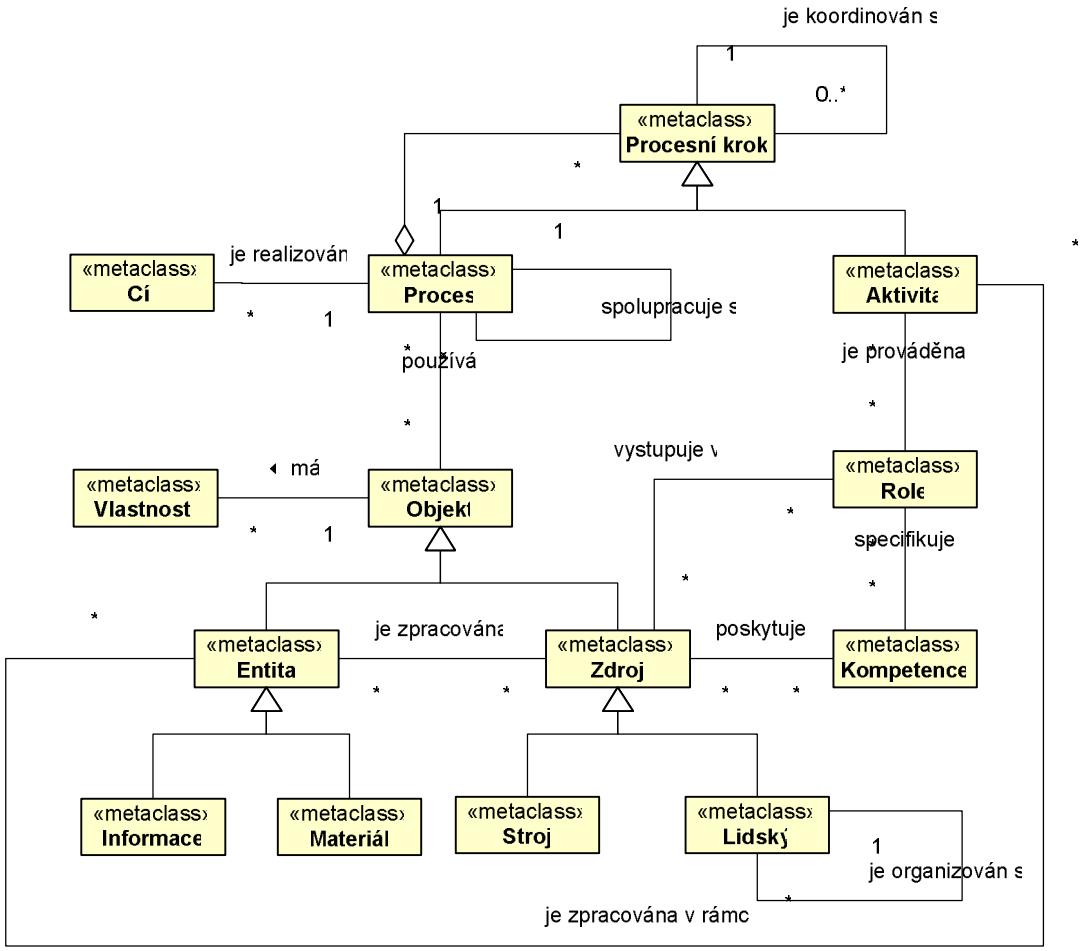
Výklad

Co je společné všem metodám

V předchozích kapitolách jsme se seznámili s několika modelovacími jazyky, které ve své podstatě usilovaly o totéž. Usilovaly o vytvoření modelu byznys procesu. K tomuto účelu sice používaly různé přístupy a abstrakce, ale ve své podstatě všude vystupovaly aktivity, objekty, zdroje apod.. Otázka tedy zní, zda-li bychom nenašli společnou specifikaci všech těchto jazyků. Jinými slovy se pokusíme o vytvoření společného modelu jazyků určených pro modelování procesů – tzv. *meta-model*.

Definice 2.4 (*Meta-model*): Meta-model je model, který definuje jazyk určený pro vytvoření modelu.

Uvedená definice tedy neříká nic jiného, než že meta-model je model popisující model. My jsme v předchozích kapitolách sestavovali modely realizace zakázky. Nyní si ukážeme jak vypadá model popisující všechny tyto modely sestavené podle různých metod.



Obr. 2.16: Meta-model

K účelu definice meta-modelu byznys procesu použijeme diagram tříd jazyka UML (obr. 2.16). Každý z modelů *Procesu*, které jsme vytvářeli, obsahuje *Procesní kroky*, kterým je buď atomická *Aktivita* nebo opět vnořený *Proces*. *Procesní kroky* jsou nějakým způsobem koordinovány (větvení, souběžnost, posloupnost atd.) a každý proces je navržen s cílem realizovat nějaké *Cíle*. *Proces* používá *Objekty* (s danými vlastnostmi), které mohou být aktivní (*Zdroje*) nebo pasivní (*Entity*). *Entity* mohou být *Informace* nebo hmatatelný *Materiál*. Na druhou stranu *Zdroje* jsou konkretizovány do *strojů* nebo *Lidských* zdrojů. *Zdroje* poskytují určité schopnosti (*Kompetence*), které jsou vyžadovány *Rolemi*. Zdroje tak vystupují v *Rolích* nutných k provedení *Aktivit*, které tak zpracovávají (spotřebovávají, modifikují nebo vytváří) odpovídající *Entity*.

Na první pohled se může zdát takový meta-model až příliš akademický. Na druhou stranu je si třeba uvědomit, že jeho znalost nám usnadní přístup i k dalším metodám a softwarovým nástrojům, které se používají k účelu modelovaní byznys procesů. Víme, co náme v těchto nástrojích hledat, a pokud se nám podaří namapovat metamodel na konkrétní názvy a pojmy, které daný nástroj používá, pak jeho zvládnutí je jen otázkou krátkého zacvičení.

Shrnutí pojmů 2.6.

Meta-model byznys procesu.

Otázky 2.6.

1. Porovnejte meta-model byznys procesu s jednotlivými diagramy. Jak jsou mapovány jejich jednotlivé elementy na objekty meta-modelu?

3. FORMALNÍ METODY SPECIFIKACE A ANALÝZY

Čas ke studiu kapitoly: 12 hodin

Cíl Po prostudování tohoto odstavce budete znát

- principy formálních metod specifikace,
- základy z oblasti konečných automatů a Petriho sítí,
- jaké vlastnosti lze ověřovat u procesů specifikovaných pomocí WF-sítí,
- jak formalizovat neformální metody specifikace.

3.1. Formální metody

Výklad

Formální a neformální specifikace

Specifikace modelovaného systému může být *formální* nebo *neformální*. Neformální specifikace používá přirozený jazyk, případně obrázky, tabulky a ostatní nástroje umožňující pochopit popisovaný systém. Tyto nástroje mohou být strukturovány a případně i standardizovány. V okamžiku, kdy taková notace získává charakter *precizní syntaxe i sémantiky*, stává se *formalismem*. Hovoříme o formální specifikaci. Jinými slovy vyjádřeno, *formální specifikace* je technika umožňující jednoznačně specifikovat modelovaný systém. Často hovoříme také o *semiformálních* metodách, které mají přesně danou syntaxi, ale netrváme na jejich úplné a precizní sémantice. Takovými metodami jsou i všechny přístupy popsané v minulých kapitolách. V dalším se podíváme nejprve na formální metody obecně a pak si ukážeme jakým způsobem lze formalizovat modely byznys procesů.

Formální metody

Formální metody zahrnují oblast jednak specifikace (popisu), ale také analýzy teto specifikace a její verifikaci. Cílem tedy není jen jednoznačně systém popsat, ale také ověřit jeho požadované vlastnosti. Obecně formální metody nedosáhly masivního nasazení. Důvodem je problém použitelnosti těchto metod při popisu rozsáhlých systémů, kdy čas a usílí nutné k takovému úkolu je příliš vysoké. Hlavní doménou pro využití formálních metod se tedy staly systémy vyžadující bezchybný chod v kritických situacích (řízení leteckých systémů, lékařský software apod.). Obvykle je však možné použít formálních metod k ověření alespoň těch nejdůležitějších částí běžně užívaných systémů.

Z hlediska samotné kategorizace formálních metod specifikace rozlišujeme mezi dvěmi základními styly – funkční a popisnou (deskriptivní) specifikací. *Funkční specifikace* popisuje systém z pohledu jeho požadovaného *chování* obvykle cestou modelu systému t.j. *abstraktním strojem* umožňujícím simulovat toto chování. Na druhou stranu *deskriptivní specifikace* definuje požadované vlastnosti systému čistě deklarativním způsobem. Poněvadž našim primárním úkolem je modelování procesů, budeme se v dalším věnovat právě formální specifikaci chování.

Formální specifikace chování

Jeden z nejznámějších přístupů k popisu chování systémů a tedy i nástrojů funkční specifikace jsou tzv. *konečné automaty*.

Definice 3.1 (*Konečný automat*): Konečný automat je uspořádaná pětice $KA = (Q, I, \delta, q_0, F)$, kde

- Q je konečná množina *stavů*,
- I je konečná neprázdná množina *vstupů*,
- δ: $Q \times I \rightarrow Q$ je *přechodová funkce*,
- $q_0 \in Q$ je *počáteční stav*
- a $F \subseteq Q$ je množina *koncových (přijímajících) stavů*.

Dle této definice probíhá *výpočet konečného automatu* nad řetězcem vstupů *input* následovně:

1. Začne se v počátečním stavu q_0 a na začátku řetězce vstupů *input*.
2. Přečte se aktuální vstup *i* řetězce *input* a přejde se do stavu určeného $\delta(q, i)$, kde q je současný stav automatu.
3. Předchozí bod se opakuje, dokud nejsou přečteny všechny vstupy z *input*.
4. Pokud je poslední stav automatu *koncový* ($q \in F$), pak je řetězec vstupů *přijat*, v opačném případě je *odmítnut*.

Pokusme se výše uvedené dokumentovat na následujícím příkladu.

Příklad 3.1 (*Zpracování faktury*): Mějme proces realizace zakázky z **příkladu 2.1**, kdy po dodání zboží je vystavena faktura a následně, v případě nutnosti, také autorizována. Faktura je pak odeslána zákazníkovi a průběžně kontrolována. Není-li faktura zaplacena, setrvává ve stavu nezaplacena. Zakázka je uzavřena až po konečném vyrovnání platby.

Konečný automat pro takovou úlohu můžeme definovat následujícím způsobem:

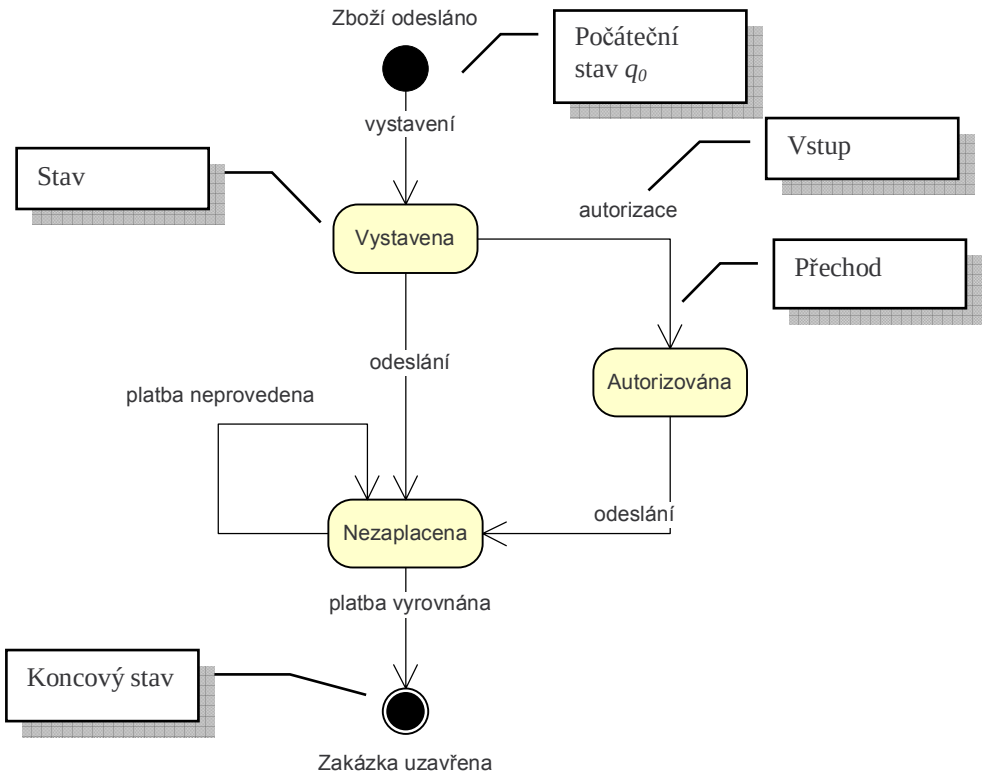
- Q = {*Zboží dodáno, Vystavena, Autorizována, Nezaplacena, Zakázka uzavřena*}.
- I = {*vystavení, autorizace, odeslání, platba vyrovnána, platba neprovedena*}.
- δ(*Zboží odesláno, vystavení*) = *Vystavena*,
  δ(*Vystavena, autorizace*) = *Autorizována*,
  δ(*Vystavena, odeslání*) = *Nezaplacena*,
  δ(*Autorizována, odeslání*) = *Nezaplacena*,
  δ(*Nezaplacena, platba neprovedena*) = *Nezaplacena*,
  δ(*Nezaplacena, platba vyrovnána*) = *Zakázka uzavřena*.
- *q0 = Zboží odesláno.*
- *F = {Zakázka uzavřena}.*

V případě, že bude řetězec vstupů tvořen např. následujícími posloupnostmi činností *vystavení* $\rightarrow$ *odeslání* $\rightarrow$ *platba vyrovnána* nebo *vystavení* $\rightarrow$ *autorizace* $\rightarrow$ *odeslání* $\rightarrow$ *platba neprovedena* $\rightarrow$ *platba neprovedena* $\rightarrow$ *platba vyrovnána*, pak celý proces fakturace a inkasa proběhl korektně, protože konečný automat *přijal* řetězec vstupů. Pokud by však řetězec vstupů byl definován následujícím způsobem *vystavení* $\rightarrow$ *autorizace* $\rightarrow$ *platba neprovedena*, pak automat odmítnul vstupní řetězec. V našem příkladu to znamená, že proces neskončí v požadovaném koncovém stavu, neboť ve stavu, kdy byla faktura *Autorizována* nedošlo k jejímu *odeslání* zákazníkovi.

Zjevně takto formálně popsaný proces faktury je přesně definovaný z hlediska syntaxe i sématiky. Nevýhodou však je náročnost sestavení takového konečného automatu a především nízká úroveň názornosti a čitelnosti. Naštěstí konečné automaty můžeme vyjádřit i jiným způsobem, než výše uvedeným formálním jazykem matematiky.

Graficky lze znázornit konečný automat pomocí grafu (*stavový diagram*), kde uzly grafu vyjadřují stavy systému a hrany označené vstupem definují přechody mezi těmito stavy (obr. 3.1). Jinými slovy vyjádřeno, hrana označená vstupem *i* spojuje uzel q_1 s uzlem q_2 právě když $\delta(q_1,i) = q_2$.

O tom, zda-li je vstupní řetězec přijat konečným automatem rozhodneme právě tehdy, když existuje posloupnost hran grafu (popisující daný automat) spojující počáteční stav q_0 s některým z koncových stavů z F tak, že spojení označení těchto hran definuje tento řetězec.



Obr. 3.1: Stavový diagram zpracování faktury

Na obrázku je uveden stavový diagram tak, jak jej definuje jazyk UML. Jazyk UML tak poskytuje v této části popisu procesu nástroje formálního charakteru. Bohužel se ale jedná o ten nejjednodušší formalismus, který selhává při popisu složitějších procesů, jejichž stavy nelze takto jednoduše vyjádřit. Když uvážíme souběžnost (paralelismus) při vykonávání procesu, pak zjistíme, že v nějakém stavu je např. nákup materiálu, v jiném stavu je příprava výroby a také příprava fakturace již mohla být zahájena. Proces se tak nachází ve stavu daném souhrnem jeho parciálních stavů, které už však tímto jednoduchým konečným automatem bude velmi složité popsat. V dalším textu si tedy ukážeme formální přístup, který tento problém adresuje a řeší na úrovni praktické použitelnosti.

Shrnutí pojmů 3.1.

Popis dynamických vlastností, konečné automaty a jejich grafická reprezentace..

Otázky 3.1.

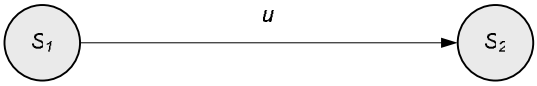
1. Jak je definován konečný automat?

3.2. Petriho sítě a jejich vlastnosti

Výklad

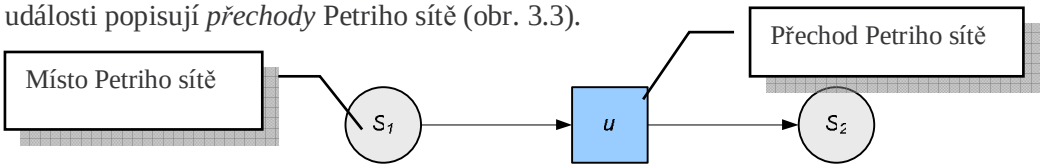
Úvod do problematiky

Petriho sítě vznikly za účelem rozšíření modelovacích možností konečných automatů. Jejich autorem je prof. Carl Adam Petri, který je navrhl jako nástroj určený k modelování a analýze procesů. V závěru minulé kapitoly jsme si ukázali, že konečné automaty popisují změny v modelovaném systému s využitím pojmů *stav* a *přechod* mezi stavy. Na obr. 3.2 je modelována změna, kdy událost *u* změní stav systému z původního S_1 na S_2.



Obr. 3.2: Modelování změny stavů pomocí konečného automatu

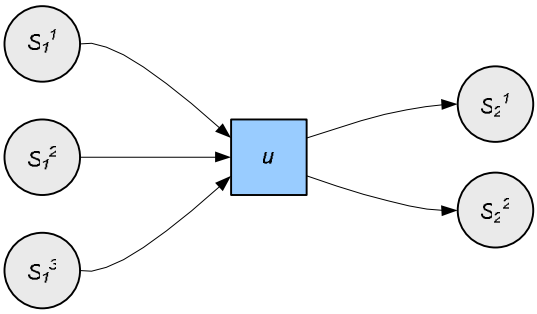
Poněvadž Petriho sítě jsou rozšířením konečných automatů, máme možnost stejnou situaci popsat i v případě Petriho sítí obdobným způsobem. Stavy jsou popsány prostřednictvím *míst* Petriho sítě a události popisují *přechody* Petriho sítě (obr. 3.3).



Obr. 3.3: Modelování změny stavů pomocí Petriho sítě

Co je však skutečným přínosem takového rozšíření pomocí Petriho sítí? Vraťme se k situaci popsané v závěru minulé kapitoly, kdy stav daného procesu je dán souhrnem jeho dílčích (*parciálních*) stavů. V tomto okamžiku narazíme na problém, že budeme muset v rámci konečných automatů vyjádřit každou takovou kombinaci explicitně samostatným prvkem $q \in Q$ z množiny jeho stavů. Při složitých procesech tak narazíme na „explozi“ počtu stavů, do kterých se proces může dostat. Navíc má tento přístup ještě jedno omezení, které vyplývá ze samotného názvu konečný automat. Tato množina stavů musí být vždy konečná.

Předpokládejme tedy, že stav S_1 je dán souhrnem jeho dílčích stavů S_1^1, S_1^2 a S_1^3 a obdobně stav S_2 je dán souhrnem dílčích stavů S_2^1 a S_2^2. Pak lze vyjádřit tuto situaci pomocí Petriho sítí s využitím míst a přechodů tak, jak je znázorněno na obrázku 3.4.



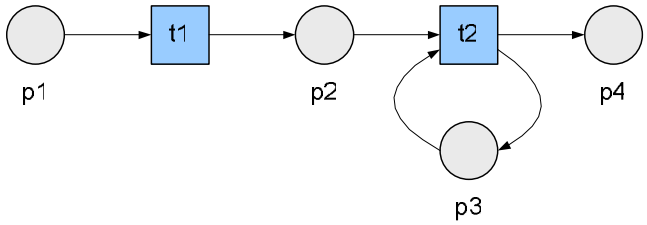
Obr. 3.4: Modelování dílčích stavů pomocí Petriho sítě

Takto znázorněná situace, kdy stav procesu je dán jeho dílčími stavy, které jsou obvykle vyjádřitelné nějakými podmínkami a spojení těchto podmínek s událostmi systému, je jádrem modelování s využitím Petriho sítí [8]. Z obr. 3.4 tedy vyplývá, že událost *u* může nastat pouze v případě, kdy jsou splněny všechny podmínky popisující dílčí stavy S_1^1, S_1^2 a S_1^3 a tato způsobí, že systém dosáhne stavu popsaného splněním podmínek příslušejícím místům S_2^1 a S_2^2.

Způsob jak vyjádřit, zda-li je daná podmínka spjatá s konkrétním místem splněna či nikoliv, řeší v případě Petriho sítí tzv. *značení místa*. Značením místa rozumíme nezápornou celočíselnou informaci, jež má v grafické reprezentaci Petriho sítě svůj obraz v podobě počtu černých teček (*značek* – anglicky *tokens*) umístěných v daném místě. Je-li podmínka splněna, pak místo musí obsahovat značku. Událost (např. reprezentována činností) může nastat, pokud všechna vstupní místa přechodu, který událost modeluje, obsahují značku. Provedení přechodu znamená, že z každého vstupního místa je značka odebrána a naopak, na všechna výstupní místa přechodu jsou značky přidány. Obecně počet odebíraných a přidávaných značek je dán ohodnocením hrany, která tato místa a přechod propojují. My v dalším textu ale budeme předpokládat, že ohodnocení je rovno jedné, a tutíž z každého vstupního místa je odebrána právě jedna značka a na výstupní místo je jedna značka přidána.

Pokusme se výše uvedené dokumentovat na nějakém jednoduchém příkladu.

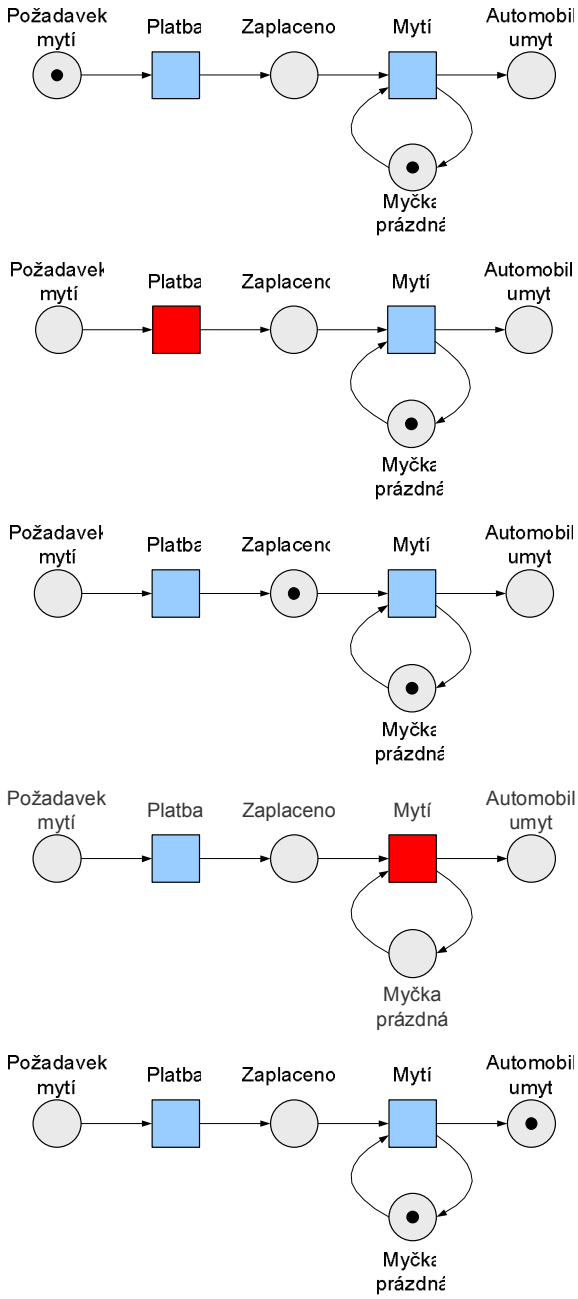
Příklad 3.2 (*Automyčka*): Mějme automyčku, která postupně umývá automobily tak, jak vznikají požadavky na jejich umytí. Vjezd do automyčky je podmíněn splněním dvou dílčích podmínek. První vyžaduje, že mytí je zaplaceno a druhá, že myčka je prázdná.



Obr. 3.5: Modelování procesu automyčky

Petriho síť na obr. 3.5 modeluje **příklad 3.2**. Stačí si za označení míst (p vychází z anglického *place*) *p1*, *p2*, *p3* a *p4* postupně dosadit podmínky *Požadavek mytí*, *Zaplaceno*, *Myčka prázdná* a *Automobil umyt*. Přechody *t1* a *t2* (t opět z anglického *transition*) pak odpovídají činnostem *Platba* a *Mytí*.

V dalším textu předpokládejme, že počáteční stav procesu mytí automobilu je dán existencí požadavku na umytí automobilu a myčka je prázdná. To znamená že na místo *p1* a *p3* umístíme značky vyjadřující splnění obou počátečních podmínek. Celou simulaci mytí automobilů si znázorníme na obr. 3.6.



Počáteční stav procesu je popsán značením na místech *p1* a *p3* vyjadřující, že je požádováno mytí a myčka je v daný okamžik prázdná. Z uvedeného značení (stavu) procesu vyplývá, že jediná činnost, která může nastat je *Platba*.

Průběh *Platby* byl zahájen odebráním značky z jejího vstupního místa.

Po provedení *Platby* byla značka přidána na její výstupní místo *p2* (*Zaplaceno*). V tento okamžik se stal proveditelným přechod reprezentující činnost *Mytí*. Tento přechod totiž obsahuje značky na všech svých vstupních místech.

Jsou odebrány značky ze vstupních míst činnosti *Mytí*, která je tímto prováděna.

Celý proces je ukončen. Požadavek na mytí byl splněn a myčka je opět prázdná.

Obr. 3.6: Simulace mytí automobilů pomocí Petriho sítě

Pokusili jsme se tedy o neformální popis Petriho sítí a dokumentovali jsme si jejich fungování na jednoduchém příkladě. Dále se tedy pokusíme o to, co dělá Petriho sítě ještě více zajímavými. Tím je jejich formální popis, který doplňuje názorné grafické zobrazení a umožňuje navíc přesně definovat některé vlastnosti procesů popsaných Petriho sítěmi. Nabízí se nám tak možnost vytvářet korektní a ověřené modely byznys procesů.

Formální definice Petriho sítí

Petriho sítě nabízí díky své formální definici celou řadu výhod. Na prvním místě je to možnost precizní a přesné specifikace procesu. Lze tak odstranit nejednoznačnosti, neurčitosti a kontradikce, kterým se nemusíme vyhnout v případě použití, v předchozích kapitolách, popsaných metod. Jimi používané diagramy jsou sice názorné, ale v případě složitých procesů máme jen malou šanci odhalit případné chyby. Začněme tedy s formální definicí Petriho sítě [7].

Definice 3.2 (*Petriho síť*). Petriho síť je uspořádaná trojice $PN = (P, T, F)$, kde

- P je konečná množina míst,
- T je konečná množina přechodů $(P \cap T = \varnothing)$
- a $F \subseteq (P \times T) \cup (T \times P)$ je množina propojení (*toková relace*).

Místo *p* nazýváme *vstupním místem* přechodu *t*, právě když existuje propojení z místa *p* do přechodu *t*. Místo *p* nazýváme *výstupním místem* přechodu *t*, právě když existuje propojení z přechodu *t* do místa *p*. Dále budeme používat $\bullet t$ k označení množiny vstupních míst přechodu *t*. Notace $t\bullet$, $\bullet p$ a $p\bullet$ mají obdobný význam, tedy např. pro $p\bullet$ platí, že označuje množinu všech následujících přechodů, které sdílí *p* jako své vstupní místo.

Pokud bychom chtěli na základě výše uvedeného formálně popsat Petriho síť pro **příklad 3.2** modelující proces automyčky, pak tato definice bude vypadat následujícím způsobem:

- $P = \{p1, p2, p3, p4\}$.
- $T = \{t1, t2\}$.
- $F = \{\langle p1, t1\rangle, \langle p2, t2\rangle, \langle p3, t2\rangle, \langle t1, p2\rangle, \langle t2, p3\rangle, \langle t2, p4\rangle\}$.

Tímto jsme popsali strukturální vlastnosti Petriho sítě. Nyní si zavedeme definici značení, které slouží k popisu stavů procesu.

Definice 3.3 (*Značení*). Značení *M* (*Marking*) Petriho sítě $PN = (P, T, F)$ je zobrazení z množiny míst *P* do množiny nezáporných celých čísel $\boldsymbol{N}$. $M: P \rightarrow \boldsymbol{N}$.

Způsobů jakými lze popsat stav Petriho sítě je několik. Může to být vektor, kde každá jeho položka odpovídá právě jednomu místu (např. [1, 0, 1, 0]) nebo se používá formule $1p1+0p2+1p3+0p4$, která odpovídá počátečnímu stavu (značení) M_0 z **příkladu 3.2**. Posledně uvedená formule je pak přepsatelná do úspornějšího tvaru *p1+p3* vynecháním míst neobsahujících žádnou značku.

Dynamika obsluhy požadavku na umytí automobilu lze tedy přepsat do následujících formulí popisujících stavy, kterými tento proces prochází:

1. *p1+p3*
2. *p2+p3*
3. *p3+p4*.

Pozn.: V případě dvou požadavků na mytí automobilu, by místo *p1* obsahovalo dvě značky a takový stav by byl popsán formulí *2p1+p3*.

K porovnání dvou stavů nám slouží relace částečného uspořádání. Pro libovolné dvě značení (stavy) M_1 a M_2 platí, že $M_1 \leq M_2 \Leftrightarrow \forall p \in P : M_1(p) \leq M_2(p)$. Dle **definice 3.3** $M(p)$ určuje počet značek na místě *p* ve stavu *M*.

Již z uvedeného příkladu je zřejmé, že počet značek se na jednotlivých místech mění. Pravidla, kterými se takové vykonání Petriho sítě řídí, jsme si již ukázali na začátku této kapitoly. Nyní nahradíme tento intuitivní popis formálním.

Definice 3.4 (*Evoluční pravidla*). Nechť je dána Petriho síť *PN* = (*P, T, F*) a její značení *M*.

1. Přechod $t \in T$ je *proveditelný* (*enabled*) při značení M, jestliže $\forall p \in \bullet t : M(p) \geq 1$.
2. Je-li přechod $t \in T$ proveditelný při značení *M*, pak jeho *provedením* (*firing*) získáme *následné* značení M' ke značení *M*, které je definováno takto:
   $\forall p \in P$:
   $$M'(p) = \begin{cases} M(p) - 1 & \text{jestliže} \quad p \in \bullet t \\ M(p) + 1 & \text{jestliže} \quad p \in t\bullet \\ M(p) & \text{jinak} \end{cases}$$

Pro danou Petriho síť *PN* = (*P, T, F*) a značení (stav) M_1 používáme následujících notací:

- $M_1 \xrightarrow{t} M_2$: přechod *t* je proveditelný při M_1 a provedení *t* vede na M_2,
- $M_1 \longrightarrow M_2$: existuje přechod *t* takový, že $M_1 \xrightarrow{t} M_2$:,
- $M_1 \xrightarrow{\sigma} M_2$: posloupnost přechodů $\sigma = t_1 t_2 t_3 .. t_{n-1}$ vede ke změně značení M_1 na M_n prostřednictvím (i prázdné) množiny dočasných značení M_2, ..., M_{n-1}, t.j. $M_1 \xrightarrow{t1} M_2 \xrightarrow{t2} \ldots \xrightarrow{tn-1} M_n$.

Definice 3.5 (*Dosažitelnost*): Nechť σ označuje posloupnost přechodů. Značení M_n nazýváme *dosažitelné* (*reachable*) ze značení M_1 (označujeme $M_1 \xrightarrow{*} M_n$), právě když $\exists \sigma : M_1 \xrightarrow{\sigma} M_n$.

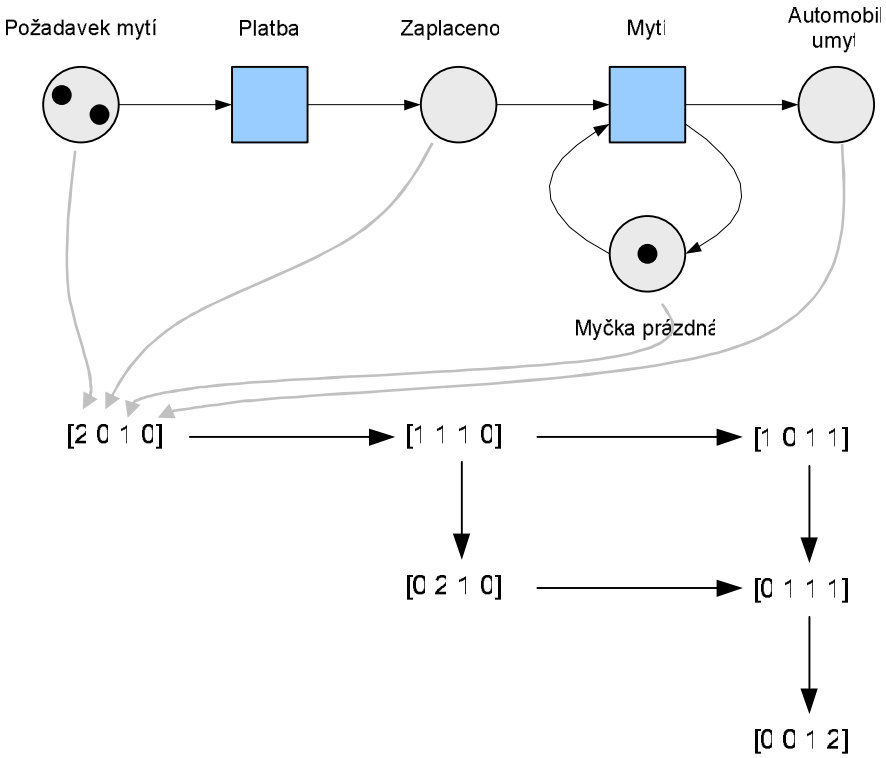
Z uvedeného tedy vyplývá, že pokud chceme ověřit, zda-li je požadované značení dostupné z výchozího, musíme nalézt takovou posloupnost přechodů, která bude postupně měnit značení Petriho sítě až na to požadované.

Definice 3.6 (*Dosažitelný stav*). Nechť je dána Petriho síť *PN* = (*P, T, F*) a její počáteční značení M_0. Značení *M* reprezentuje *dosažitelný stav* (*reachable state*), právě když $M_0 \xrightarrow{*} M$.

Právě problematika dosažitelnosti je jednou z klíčových otázek, na kterou hledáme odpověď při sestavování byznys procesu. Zřejmě takový proces, který nedosáhne svého koncového stavu je chybně navržený (obr. 2.10). Na druhou stranu můžeme mít opačný požadavek. Budeme se chtít vyhnout nepřípustným stavům. Cestou jak ověřit, zda-li daný stav je dosažitelný či nikoliv, je sestavení *grafu dosažitelnosti* (*reachability graph*). Graf dosažitelnosti je tvořen uzly, které definují dosažitelné stavy popsané vektorem značení a hranami (šipkami) označujícími změny těchto stavů. Algoritmus jeho sestavení je následující:

1. Nechť *X* je množina obsahující právě počáteční stav a množina *Y* je prázdná.
2. Vezmi prvek $x \in X$ a přidej jej do množiny *Y*. Spočítej všechny stavy dosažitelné z *x* uskutečněním proveditelných přechodů. Každý následný stav, který není obsažen v *Y* přidej do *X*.
3. Pokud *X* je prázdná množina algoritmus je ukončen, jinak běž do bodu 2.

Dokumentujme si výše uvedené na **příkladu 3.2** automyčky s jediným rozdílem. Předpokládejme na vstupu vznik dvou požadavků na umytí automobilu. Petriho síť popisující tuto situaci a odpovídající graf dosažitelnosti je pak znázorněn na obr. 3.7.



Obr. 3.7: Graf dosažitelnosti procesu automyčky s dvěmi požadavky

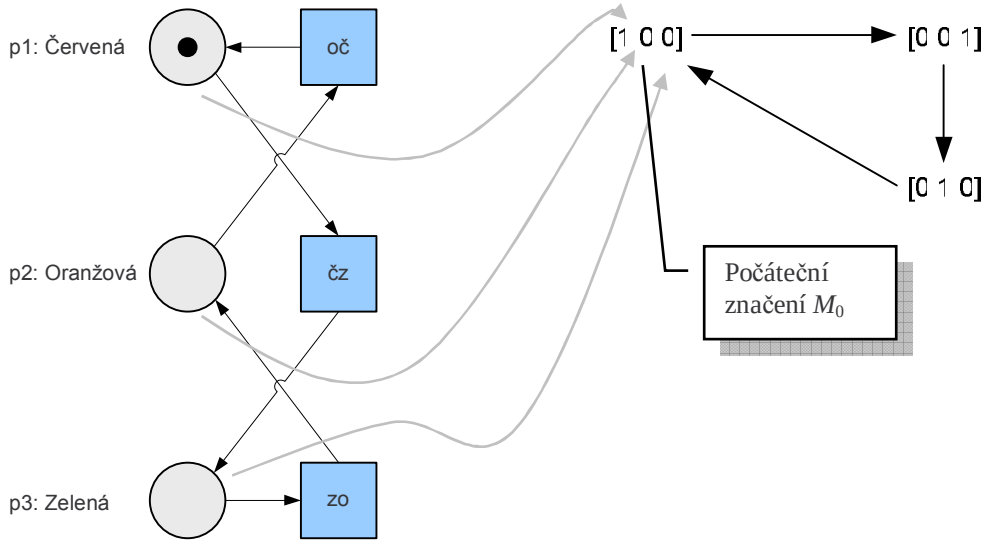
Z příkladu vyplývá, že stav [1,1,1,0] je dosažitelný provedením přechodu *Platba*. Pak mohou nastat dva možné stavy podle toho, zda je nejprve proveden přechod *Mytí* a nebo opětovně *Platba*. Hovoříme o tzv. *nedeterministecké volbě* Petriho sítě. Nikde není předurčeno, který z těchto přechodů je upřednostněn. Uzel, ze kterého nevedou žádné šipky je koncový stav. V našem případě je to stav popsaný vektorem [0,0,1,2]. Z uvedeného grafu a z případných dalších sestavených pro různé počty požadavků pak vyplývá obecný závěr, že každý počáteční stav daný vektorem značení [*x*,0,1,0] vede ke koncovému stavu [0,0,1,*x*]. To znamená, že nezávisle na počtu počátečních požadavků na mytí proces končí stejným počtem umytých automobilů.

Ukažme si na příkladu řízení světelné křižovatky úlohu, která vyžaduje nepřípustnost určitého stavu. Proces řízení světel musí být vždy takový, aby se v grafu dosažitelnosti neobjevil stav, který bude mít ve dvou křížících se směrech na semaforu zelenou.

Příklad 3.3 (*Křižovatka*). Mějme křižovatku s protínajícími se dvěmi jednosměrnými ulicemi. Každý směr bude mít své světelné řízení. Průjezd křižovatkou musí být bezpečný, přičemž je nutné zajistit střídavou průjezdnost v jednotlivých směrech.

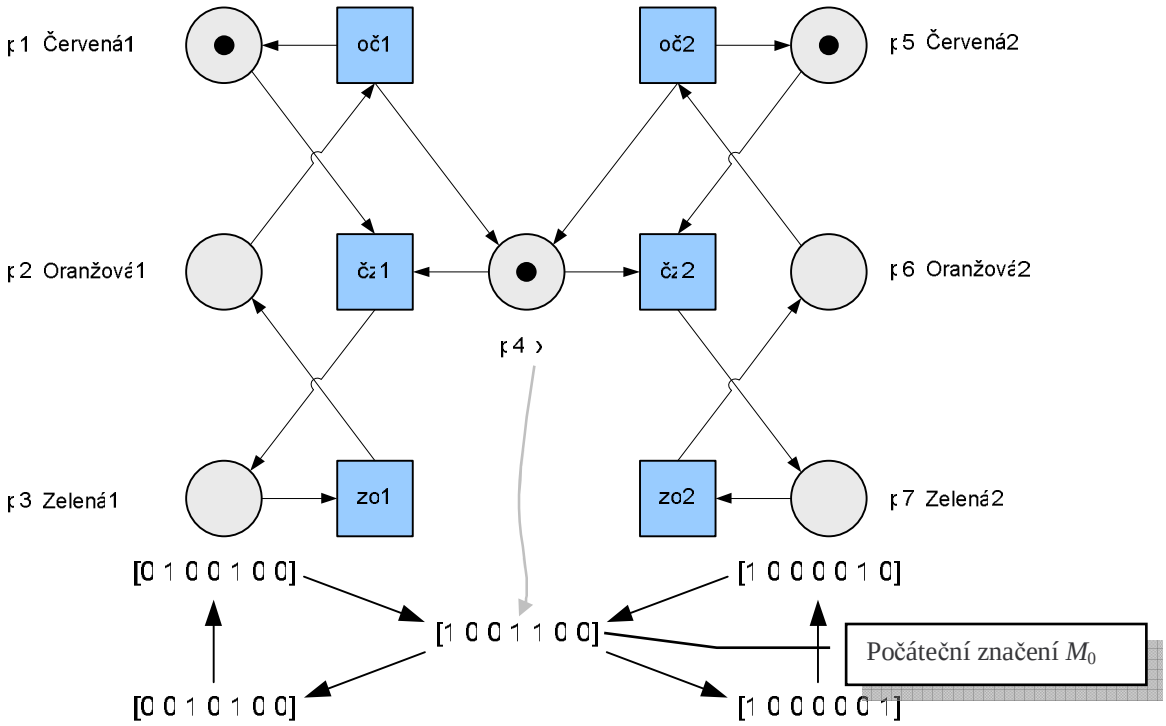
Pro účely modelování procesu řízení světel na křižovatce si nejprve vytvoříme model semaforu pro jeden jízdní směr (obr. 3.8).

Petriho síť je tvořena třemi místy a třemi přechody. Každé z míst odpovídá jednomu ze tří světel. V případě, že je dané světlo aktivní, pak obsahuje značku. Přechody zajišťují jejich střídání. Přechod *čz* mění červené světlo na zelené, přechod *zo* zelené světlo na oranžové a nakonec přechod *oč* zajišťuje přepnutí oranžové na červené světlo. Pro zjednodušení úlohy jsme vynechali oranžovou mezi přepnutím z červené na zelenou. Graf dosažitelnosti pak dokladuje korektnost požadovaného přepínání světel.



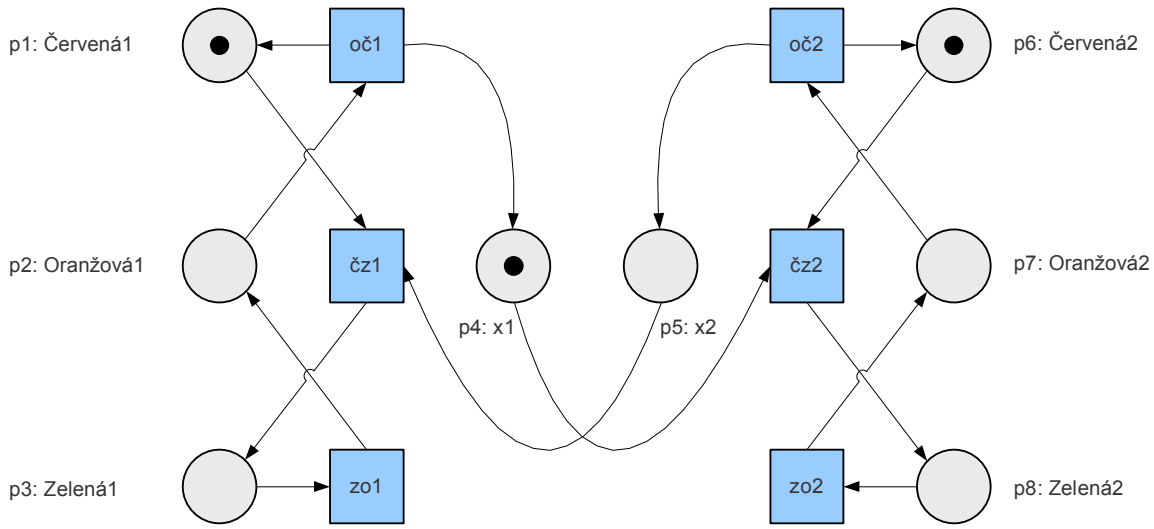
Obr. 3.8: Proces semaforu a jeho graf dosažitelnosti

Naše křižovatka však požaduje druhý semafor, přičemž přepínání světel mezi nimi musí být synchronizováno tím způsobem, že na obou nesmí svítit současně zelené světlo. K tomu účelu propojíme oba semafory sdíleným místem *x*, které bude obsahovat jedinou značku. Toto místo bude podmiňovat možnost přepnutí z červené na zelenou. Právě jediná značka na tomto místě modeluje tzv. vzájemnou výlučnost (*mutual exclusion*), protože jakmile je přechod z červené na zelenou jednoho ze semaforů proveden, značka je odebrána a je zablokováno provedení téhož u semaforu druhého. Značka je „vrácena“ na místo *x* až v okamžiku přepnutí semaforu zpět na červenou (obr. 3.9).



Obr. 3.9: Proces řízení světel na křižovatce a jeho graf dosažitelnosti

V grafu dosažitelnosti zjevně chybí stav, který by současně obsahoval na pozici p3 a p7 jedničku. Jinými slovy řečeno, je splněna podmínka bezpečného průjezdu křižovatkou, protože nikdy nemůže na obou semaforech svítit zelené světlo. Přesto má tento proces jeden velký nedostatek. Ten vyplývá z nedeterminismu přechodu z počátečního stavu M_0 = [1,0,0,1,1,0,0] do dvou možných následných stavů. Tento už dříve zmíněný nedeterminismus může způsobit, že pouze jeden ze semaforů bude střídat svá světla, zatímco druhý bude stále setrvávat na červené. Jinými slovy vyjádřeno, není splněn požadavek střídavého průjezdu. Výsledné řešení splňující všechny podmínky dokumentuje obr. 3.10.



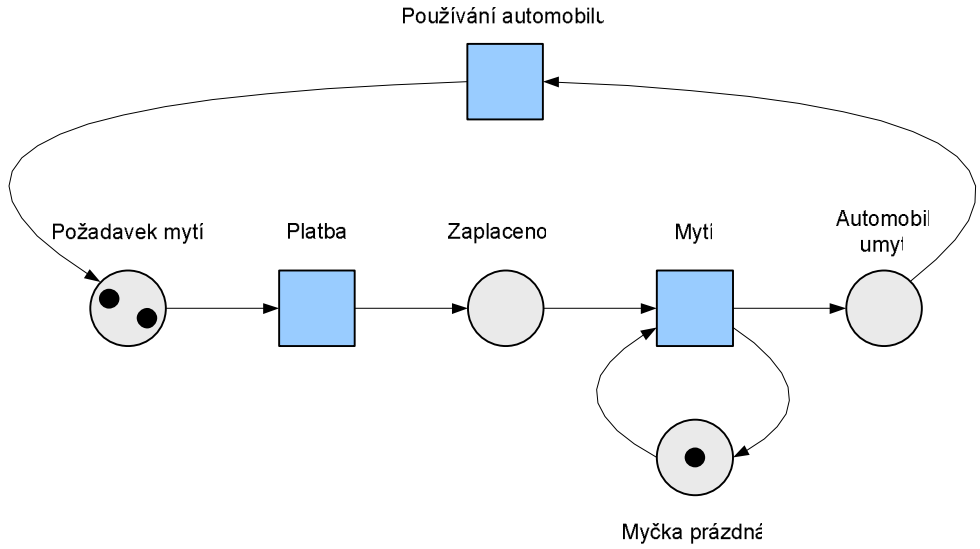
Obr. 3.10: Korektní proces řízení světel na křižovatce

Na závěr tohoto odstavce si uveďme ještě dvě další vlastnosti, které lze při analýze procesů navržených pomocí Petriho sítí využít. Jedná se o tzv. *živost* Petriho sítě, která je definována následujícím způsobem:

Definice 3.7 (*Živost*). Nechť je dána Petriho síť *PN* = (*P*, *T*, *F*) a její počáteční značení M_0. Značení $M : M_0 \xrightarrow{*} M$ je živé (*live*), jestliže pro $\forall t \in T$ existuje značení $M' : M \xrightarrow{*} M'$ takové, že přechod *t* je v M' proveditelný. Jsou-li všechna značení $M : M_0 \xrightarrow{*} M$ živá, pak Petriho síť se nazývá *živá Petriho síť*.

Pokusme se o volnou interpretaci této definice. *Petriho síť je živá*, pokud je možné dosáhnout – pro každý její přechod *t* a pro každý stav *M* dosažený z počátečního značení M_0 – stav M', ve kterém je přechod *t* proveditelný. To znamená, že v živé Petriho síti je možné každý přechod provést volitelně mnohokrát. Nemůže tedy dojít k jejímu zablokování v nějakém stavu, kdy není žádný z přechodů proveditelný (*deadlock*) nebo uvíznutí v nekonečné smyčce, kterou nelze opustit (*livelock*).

Příklad řízení světelné křižovatky znázorněný na obr. 3.10 je živá Petriho síť, zatímco proces automyčky z obrázku 3.7 nikoliv. Pokud bychom ale tento posledně jmenovaný proces rozšířili o další přechod např. *Používání automobilu* propojující místa *Autmobil umyt* a *Požadavek mytí*, pak i v tomto případě se bude jednat o živou Petriho síť (obr. 3.11).



Obr. 3.11: Živá Petriho síť procesu mytí automobilů

Poslední vlastnost, kterou budeme definovat a dále také využívat je *omezenost* Petriho sítě.

Definice 3.8 (*Omezenost, bezpečnost*). Nechť je dána Petriho síť *PN* = (*P*, *T*, *F*) a její počáteční značení M_0. Místo $p \in P$ se nazývá omezené, jestliže $\exists n \in N : \left(\forall M : M_0 \xrightarrow{*} M\right) : M(p) \leq n$. Je-li každé místo Petriho sítě *PN* omezené (*bounded*), pak *PN* se nazývá *omezenou Petriho sítí*. Místo $p \in P$ se nazývá bezpečné (*safe*), jestliže platí $\left(\forall M : M_0 \xrightarrow{*} M\right) : M(p) \leq 1$. Je-li každé místo Petriho sítě *PN* bezpečné, pak *PN* se nazývá *bezpečnou Petriho sítí*.

Neformálně vyjádřeno, Petriho síť je omezená, pokud existuje pro každé místo přirozené číslo *n* tak, že v každém dosažitelném stavu je počet značek na tomto místě menší nebo roven tomuto číslu. Speciálním případem je bezpečná síť, která vyžaduje, že počet značek je maximálně roven hodnotě 1.

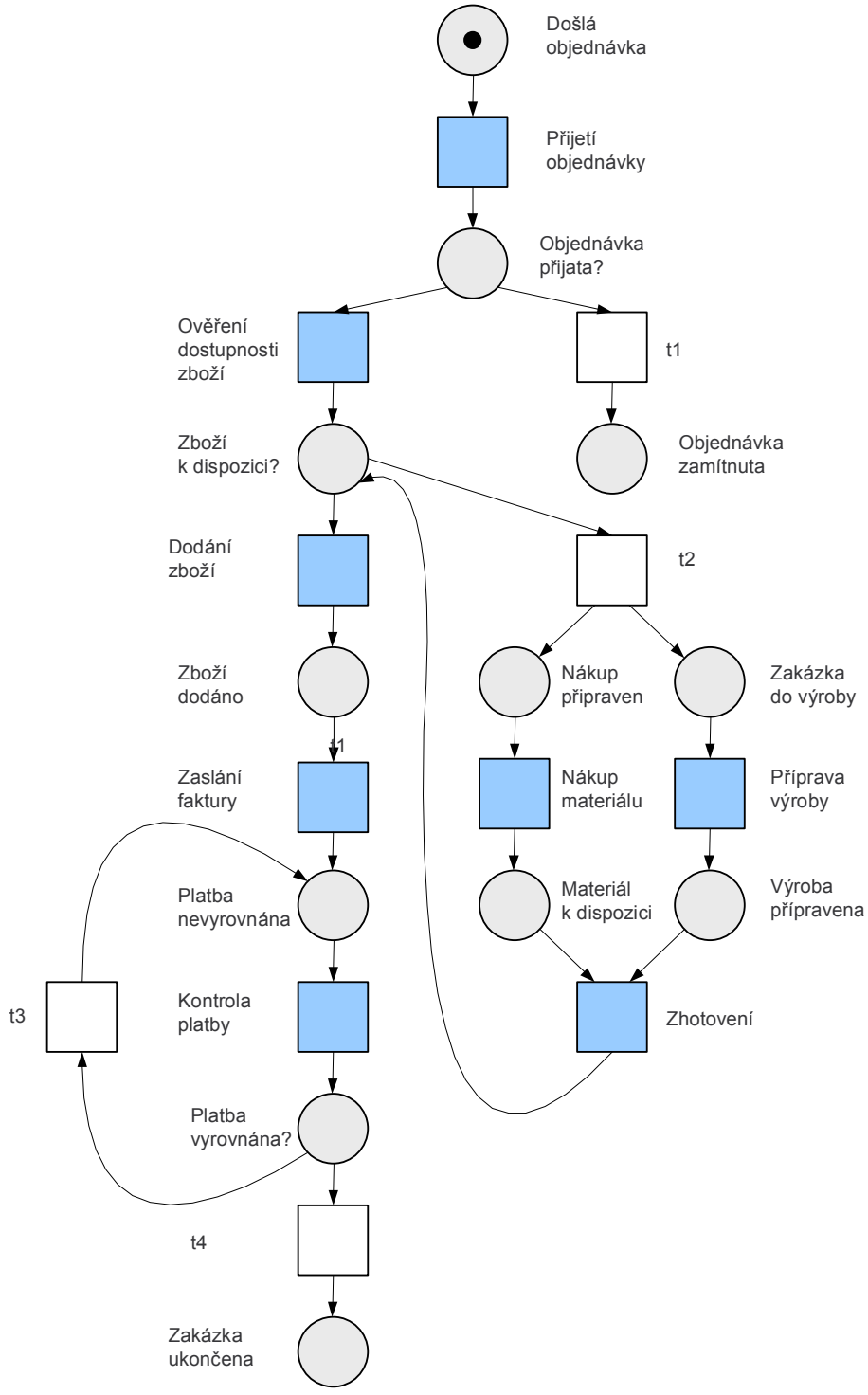
Pokud se podíváme zpět na oba diskutované příklady procesů automyčky a řízení světelné křižovatky, tak první z nich je omezen a druhý dokonce bezpečný z pohledu terminologie Petriho sítí. Při analýze procesu pak můžeme zjišťovat, zda-li je např. u procesu automyčky místo *Myčka prázdná* bezpečné. Pokud není, znamená to, že může obsahovat více než jednu značku a není vyloučeno najetí dalšího automobilu do myčky během mytí předchozího.

❑ Petriho sítě vyšší úrovně

Jak je vidět, Petriho sítě mají kromě názorného grafického vyjádření i solidně definované matematické základy, které se nabízí k využití v různých softwarových nástrojích pro specifikaci a analýzu *workflow* - počítačově zpracovatelným byznys procesům. Přesto však klasické Petriho sítě naráží na problémy modelování skutečných a složitých procesů. Z těchto důvodů se věnovalo hodně usílí ve směru jejich rozšíření s cílem dosáhnout zvýšení jejich modelovací síly. Ta rozšíření, která zde zmíníme a také budeme používat jsou následující: *barevné* rozšíření, *časové* rozšíření a *hierarchické* rozšíření.

V případě rozšíření Petriho sítí o *barvy* se jedná o přiřazení hodnoty (barvy) značce. Díky tomu lze mezi jednotlivými značkami rozlišovat co vyjadřují a provádění přechodů může probíhat na základě informace, která není explicitně vyjádřena v grafu Petriho sítě, ale je ukryta v hodnotách značek a v procedurách spojených s prováděním přechodů. To znamená, že na rozdíl od klasických Petriho sítí proveditelný přechod nemusí být proveden, poněvadž hodnota značky nesplňuje podmínky jeho provedení. Přítomnost značek na vstupních místech tedy nepostačuje k provedení přechodu. Obdobně přechod nemusí produkovat značky na všechna svá výstupní místa, ale pouze na ta, která jsou určena hodnotou spotřebované značky a předdefinovanými pravidly provedení přechodu.

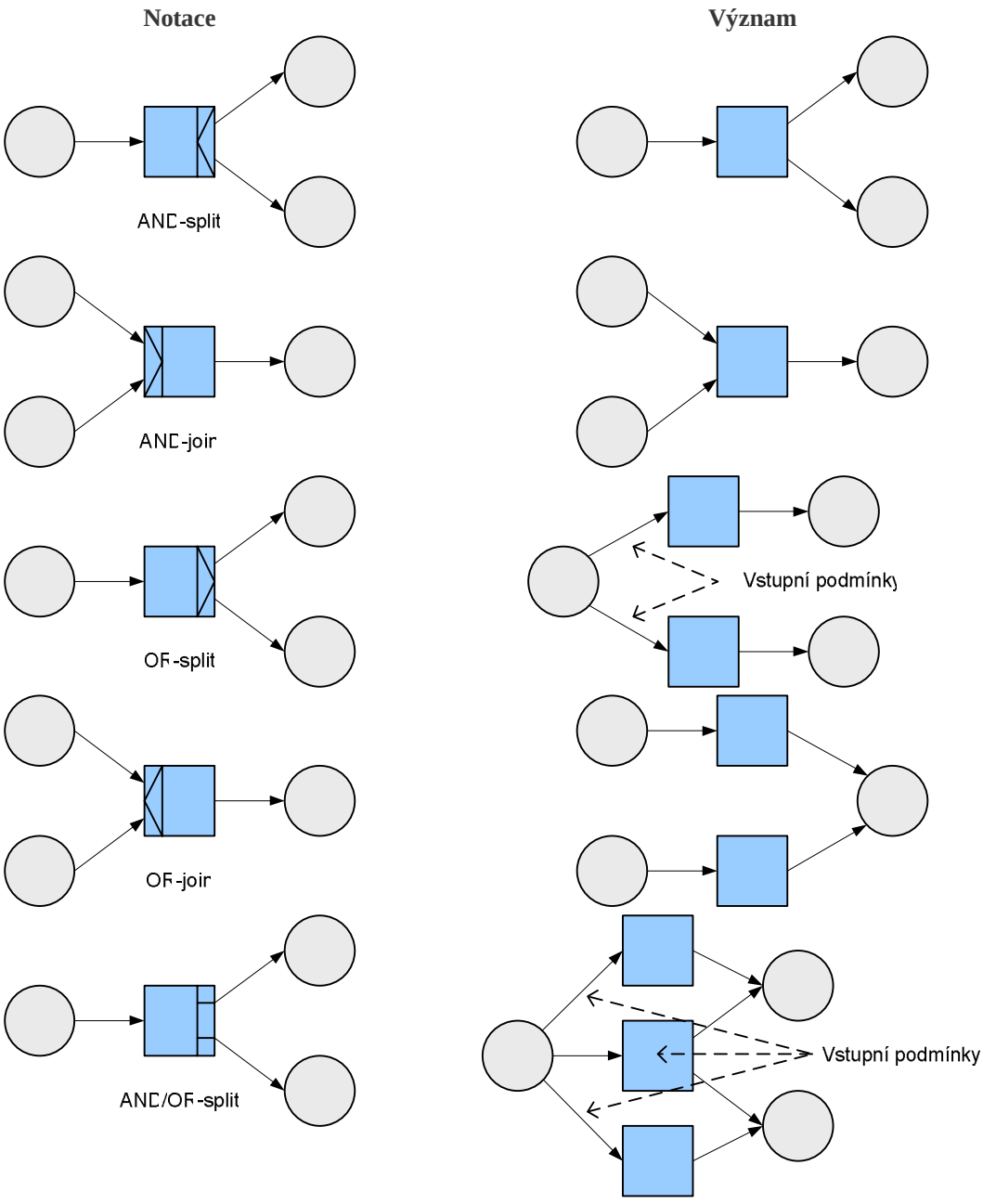
Ukažme si nyní, jak by vypadal **příklad 2.1** realizace zakázky modelovaný pomocí *barvené* Petriho sítě.



Obr. 3.12: Realizace zakázky pomocí barvené Petriho sítě

Z modelu je patrné, že jediné v čem se liší klasická Petriho síť od barvené je řešení nedeterminismu vlastnímu klasickým Petriho sítím v místech *Objednávka přijata?*, *Zboží k dispozici?* a *Platba vyrovnána?*. Pomocí vstupních pravidel následných přechodů lze rozhodnout o tom, zda-li bude objednávka přijata či zamítnuta (značka reprezentující objednávku musí být asociována se seznamem

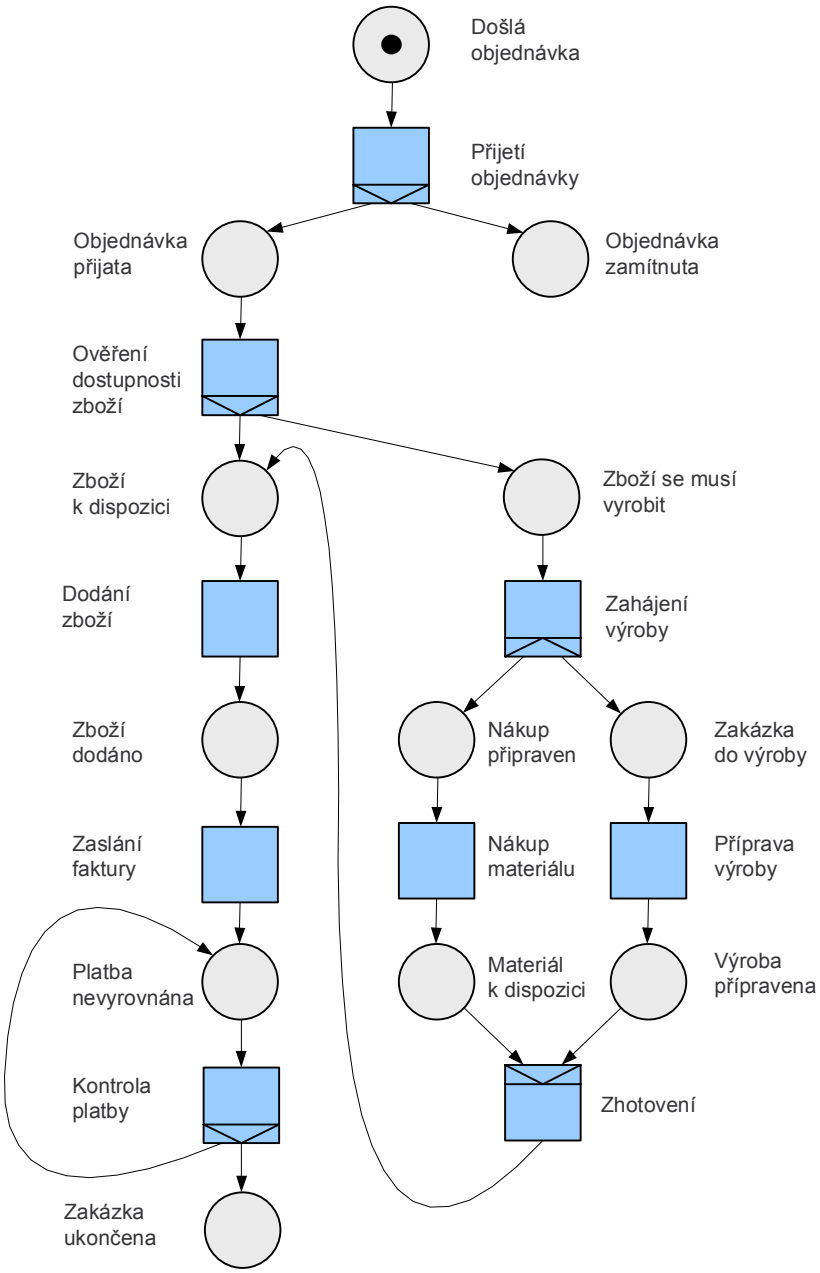
požadovaného zboží), zda-li je nutné zboží vyrobit (značka potvrzuje/nepotvrzuje dostupnost zboží ve skladu) a zda-li je nutné opět kontrolovat platbu, pokud tato nebyla vyrovnána (značka musí nést binarní informaci o tom, zda je či není platba vyrovnána). V modelu byly použity činnosti, které se již objevily v modelech řešených pomocí metod uvedených v kapitole 2. Nicméně formalismus Petriho sítí si vynutil zavedení několika dalších přechodů t_1, t_2, t_3 a t_4. Přechody t_1, t_3 a t_4 slouží k modelování rozdělení toku procesu na (v našem případě dvě) alternativní větve (*OR-split*). Přechod t_2 rozděluje proces na dva souběžné (paralelní) toky činností nákupu materiálu a přípravy výroby (*AND-split*). Z hlediska barevných Petriho sítí jsme však nikde nevyužili možnost selektivního umístění značek na výstupní místa přechodů. Než si ukážeme jak by vypadal proces realizace zakázky v tomto případě, zaveďme si nové notace značení přechodů, které nám umožní lépe modelovat logické spojky a zvýšit tak čitelnost specifikovaných procesů (obr. 3.13).



Obr. 3.13: Notace a význam speciálně zavedených přechodů

Za povšimnutí stojí *OR-split*, který dle sémantiky Petriho sítí vyjadřuje fakt, že značka může být umístěna na právě jedno z míst popisujících alternativní možnosti. *AND/OR-split* potom popisuje varianty obsazení značkou prvního, druhého nebo obou míst současně.

Díky zavedené notaci, která však stále dodržuje všechny výše uvedená pravidla týkající se formální specifikace Petriho sítí, můžeme proces realizace zakázky přepsat do úspornějšího i názornějšího tvaru, který se velmi blíží diagramům metod popsaných v minulé kapitole (obr. 3.14). Na rozdíl od nich, je však takto sestavený model formálně specifikovatelný, což přináší všechny výše uvedené výhody. Model lze jednoduše implementovat a pomocí počítače také ověřit požadované vlastnosti jako je např. dosažitelnost koncového stavu, živost, bezpečnost a podobně. To by u ostatních metod bylo možné řešit jen s obtížemi, pokud vůbec.



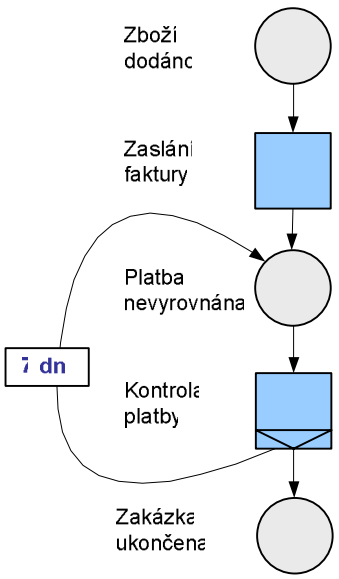
Obr. 3.14: Modifikace Petriho sítě

Barevnost Petriho sítí nám tedy umožňuje na základě hodnot asociovaných se značkou modelovat *vstupní* a *výstupní* podmínky přechodů (preconditions a postconditions) a významně tak napomoci modelování složitých procesů

Časové rozšíření Petriho sítě je postaveno na dvou základních principech. Každá značka s sebou nese *časové razítko*, které určuje, kdy může být značka z místa odebrána. Proveditelný přechod tedy může být proveden a spotřebovat značku ze svého vstupního místa až v okamžiku, kdy aktuální čas je roven nebo překročil časové razítko značky. Jinými slovy vyjádřeno, *čas proveditelnosti* přechodu je nejvčasnější okamžik, kdy všechna jeho vstupní místa obsahují *dostupnou* značku. Díky tomuto principu je jako prvý proveden přechod, jehož proveditelný čas je nejnižší. V případě, že je více přechodů se stejným časem proveditelnosti, je dle principu klasických Petriho síti provedena nedeterministická volba jednoho z nich.

Jestliže je přechod proveden a jsou jím vytvořeny značky, pak každé z nich je přiřazeno časové razítko rovné nebo pozdější než byl čas provedení přechodu. Přechod tak značce může přiřadit *zpoždění*. Časové razítko značky tak bude dáno časem provedení přechodu plus toto zpoždění. Samotný přechod je však proveden okamžitě, tedy nespotřebovává žádný čas. Pokud bychom chtěli modelovat činnost, která trvá určitý časový interval, pak budou mít značky produkované přechodem, reprezentující tuto činnost, nastaveno časové razítko na čas provedení plus hodnota tohoto časového intervalu.

Příklad zavedení času do modelu procesu si ukážeme na realizaci zakázky z obr. 3.14. Přechod reprezentovaný činností *Kontrola platby* probíhá v cyklu, dokud není platba vyrovnána a následně *Zakázka ukončena*. Model procesu však nic neříká o tom, kdy se má kontrola opakovat v případě nezaplacené faktury. Předpokládejme, že tato kontrola by měla být prováděna vždy jednou za týden. Pak lze rozšířit model procesu o zpoždění 7 dní, které bude přiřazeno značce přechodem *Kontrola platby* v případě, že podmínka zaplacení nebyla splněna (obr. 3.15).



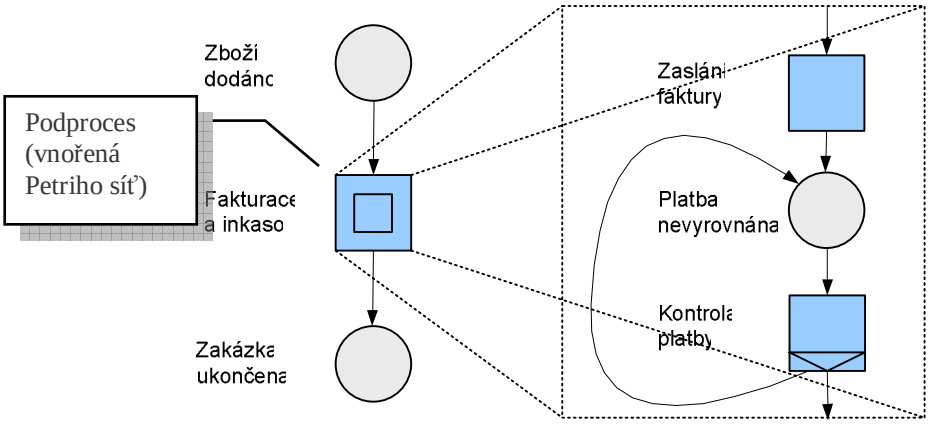
Obr. 3.15: Doplnění části Petriho sítě realizace zakázky o časový rozměr

Z modelu vyplývá, že značka produkovaná přechodem *Kontrola platby* v případě nezaplacení bude obsahovat časové razítko dané časem provedení přechodu plus zpožděním 7 dní. Nová kontrola tedy nebude moci proběhnout dříve, dokud nebude aktuální čas roven tomuto časovému razítku.

Hierarchické rozšíření Petriho sítí adresuje problém vytváření modelů složitých procesů, které obsahují jiné procesy jako své podprocesy. Při vytváření modelů tak můžeme postupovat způsobem postupného zpodrobňování jednotlivých přechodů, které budou nahrazeny jinou Petriho sítí (postup od

zhora dolů) nebo naopak skládat výsledné procesy z připravených komponent (postup zdola nahoru). V běžné praxi se tyto dva přístupy obvykle kombinují a doplňují navzájem.

Pokusme se výše uvedené demostrovat na části našeho procesu realizace zakázky, která se zabývá problematikou fakturace a inkasa (obr. 3.15). Proces obsahuje dvě činnosti: *Zaslání faktury*, *Kontrola platby* a jedno místo *Platba nevyrovnána*. My se pokusíme tuto část definovat jako podproces, na který se bude proces realizace zakázky odkazovat (obr. 3.16).



Obr. 3.16: Hierarchizace Petriho sítí

Hlavní výhodou takového přístupu je možnost vytvářet modely velmi složitých procesů na principu „rozděl a panuj“. Navíc identifikace takových podprocesů přímo podporuje znovupoužití již dříve definovaných procesních komponent a usnadňuje tak vývoj modelů.

V této kapitole jsme si ukázali co jsou to Petriho sítě, definovali jsme celou řadu jejich vlastností a nakonec jsme se věnovali jejich rozšíření cestou barev, času a hierarchizace. V dalším textu si ukážeme jak lze Petriho sítě dále modifikovat k výhradnímu účelu modelování byznys procesů.

Shrnutí pojmů 3.2.

Petriho síť a její značení.

Evoluční pravidla.

Dosažitelnost, dosažitelný stav a graf dosažitelnosti.

Živost, omezenost a bezpečnost.

Petriho sítě vyšší úrovně, barevnost, čas a heirarchizace.

Otázky 3.2.

1. Co je to Petriho síť?
2. Jak jsou modelovány stavy v Petriho sítích?
3. Popište algoritmus sestavení grafu dosažitelnosti.
4. Jaká se používají rozšíření Petriho sítí pro potřeby zvýšení jejich modelovací síly?

Úlohy k řešení 3.2.

1. Sestavte Petriho síť popisující semafor, který prochází oranžovou při přepínání světel z červené na zelenou i ze zelené na červenou.
2. Dokažte pomocí grafu dosažitelnosti, že spolupráce dvou semaforů z obr. 3.10 je skutečně korektní a splňuje podmínky stanovené ve specifikaci **příkladu 3.3**.
3. V **příkladu 3.3** řízení světelné křižovatky přiřaďte výstupům přechodů časový rozměr (viz 3.15) tak, aby úloha mohla být řešena dle modelu specifikovaného na obr. 3.9. Zpoždění zaveďte tak, že červená bude v obou směrech svítit 30s, zelená 25s a oranžová pouze 5s.

3.3. Modelování procesů pomocí WF-sítí

Výklad

Petriho sítě a modelování byznys procesů

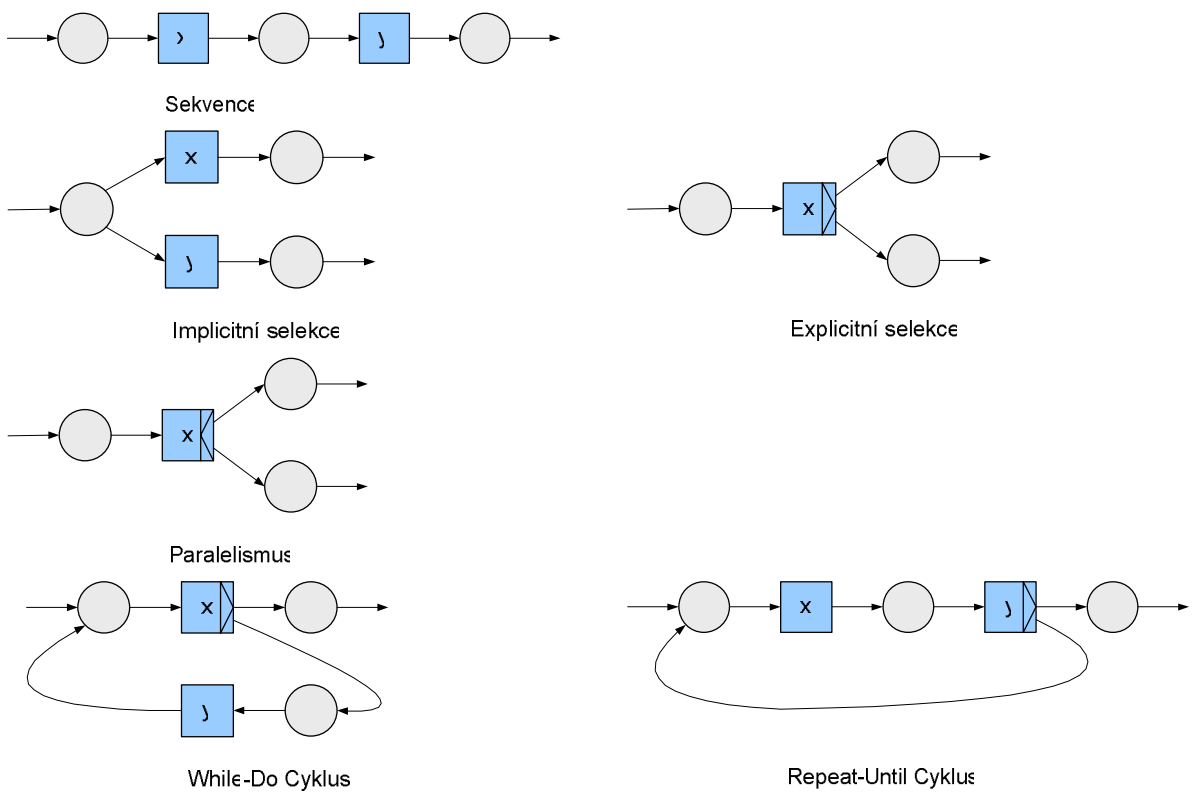
V minulé kapitole jsme popsali, jak lze procesy, a obecně systémy, modelovat pomocí Petriho sítí. Nyní si ukážeme, jak lze koncept Petriho sítí modifikovat pro účely modelování byznys procesů. Poněvadž předpokládáme, že námi modelované procesy budou určeny pro zpracování, i ve fázi vytváření samotných modelů, pomocí počítače, budeme v dalším textu často používat i vhodnějšího pojmu workflow. Modifikované Petriho sítě určené právě pro modelování workflow budeme tedy nazývat WF-sítě (WorkFlow – WF-nets) [7]. Ukážeme si nejprve, jaké budou požadavky kladené na WF-sítě, aby plnily tento účel.

Každý byznys proces má svůj začátek a konec. Z toho vyplývá, že Petriho síť modelující takový byznys proces by měla mít své jedno počáteční a jedno koncové místo. Každá instance procesu by tak měla mít na tomto počátečním místě svou značku modelující konkrétní případ, který bude procesem specifikovaným Petriho sítí zpracován. V rámci vykonávání instance procesu se pak na ostatních místech objevují značky tak, jak se objevují podmínky nutné k jeho úspěšnému ukončení. Případ (instance procesu) je ukončen v okamžiku, kdy je umístěna značka na koncové místo označující ukončení procesu. V ten okamžik by neměla být na žádném z míst značka odpovídající danému případu. To by znamenalo, že případ nebyl ve skutečnosti ukončen a model workflow je nekorektně navržen. Více zpracovávaných případů pak modelujeme pomocí barev přiřazených jednotlivým případům. Je zde však i druhá možnost, a to vytvořit samostatnou kopii Petriho sítě pro každý jednotlivý případ. Kdybychom měli v rámci procesu zpracovávány tři případy (např. červený, modrý a zelený), pak můžeme mít v Petriho síti tři sady značek v barvách odpovídajících jednotlivým případům, a nebo tři samostatné Petriho sítě bez nutnosti barvení značek. Obě varianty jsou možné. Pro jednoduchost znázornění ale budeme v dalším uvažovat právě druhou z variant, reprezentovanou samostatnou Petriho sítí pro každý případ – instanci procesu.

Podíváme-li se na meta-model byznys procesu (obr. 2.16), pak pro jednotlivé procesní kroky (podprocesy nebo aktivity) platí, že jsou koordinovány. Tato koordinace je standardizována do následujících čtyř typů řízení toku činností (obr. 3.17):

1. *Sekvence* popisuje postupné provedení aktivit. Obykle je tím dána jejich vzájemná závislost, kdy jedna aktivita *x* produkuje výstup požadovaný za vstup následující druhé aktivity *y*.
2. *Selekce* popisuje větvení toku činností do jedné z alternativních možností. V podstatě větvení je možné realizovat dvěmi způsoby – *implicitně* nebo *explicitně*. V případě skrytého (implicitního) větvení se jedná o nedeterministické rozhodnutí, případně rozhodnutí vyplývající ze vstupních podmínek aktivit, které rozhodnou o tom, která z nich odebere značku ze vstupního místa. Zřejmé (explicitní) větvení je provedeno na základě jasně daného pravidla v rámci provedení aktivity *x* a značka je pak umístěna na právě jedno z jejích výstupních míst. Odtud také vyplývá i rozdíl, kdy je vlastní rozhodnutí o větvení provedeno. V rámci explicitní selekce je to ihned, zatímco v případě implicitní selekce je rozhodnutí provedeno později, v okamžiku, kdy se aktivita *x* nebo druhá aktivita *y* stane proveditelnou.
3. *Paralelismus* definuje situaci, kdy více než jedna aktivita jsou proveditelné souběžně nebo v libovolném pořadí. Jejich provedení se tedy navzájem neovlivňuje. Aktivita *x* toho docílí umístěním značek na všechna svá výstupní místa.
4. *Cyklus* popisuje opakující se vykonání aktivit. V prvém případě cyklu nazvaný *while-do* není zaručeno, že aktivita *x* musí být provedena, pokud není splněna podmínka cyklu. Druhý

případ cyklu nazvaný *repeat-until* zaručuje, že aktivita *x* je provedena minimálně jednou a opakuje se, pokud je splněna podmínka cyklu.



Obr. 3.17: Implementace řídících struktur ve WF-sítích

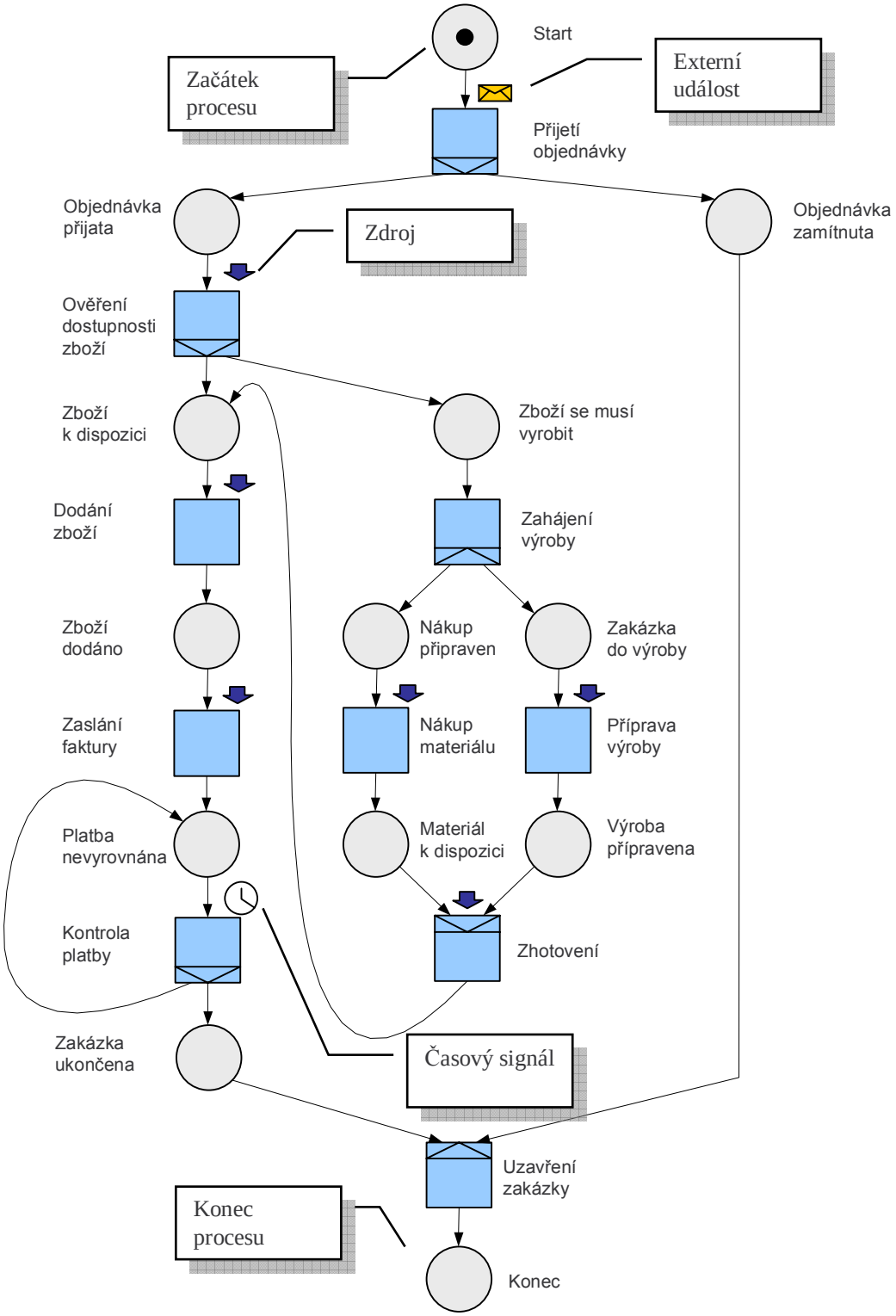
Poslední požadavek, který je nutno zahrnout do modelování reálných byznys procesů, jsou mechanismy spuštění aktivity, které jsou v Petriho sítích modelovány pomocí provedení přechodů. Petriho síť realizuje přechod okamžitě, kdy je tento přechod proveditelný. V reálné situaci jsme ale vázáni na celou řadu dalších podmínek, které musí být splněny. Pokusíme-li se o jejich abstrakci, pak se v podstatě jedná se o následující mechanismy spuštění:

1. Aktivita je spuštěna *zdrojem*, který musí být dostupný.
2. Spuštění aktivity je podmíněno přijetím *externí události* (zpráva, dokument apod.).
3. Spuštění aktivity je podmíněno *časovým signálem*.

Aktivity, které nejsou podmíněny žádným mechanismem jejich spuštění jsou provedeny okamžitě.

Pozn.: Z hlediska formálního popisu Petriho sítí se nedopouštíme žádných nesrovnalostí, poněvadž všechny tyto podmínky mohou být modelovány dalším vstupním místem přechodu, kde značka vyjadřuje přítomnost zdroje, externí události nebo časového signálu. Pro jednoduchost zápisu autor WF-sítí (Wil van der Aalst) však zavádí symboly, které názorně definují způsob spuštění aktivity.

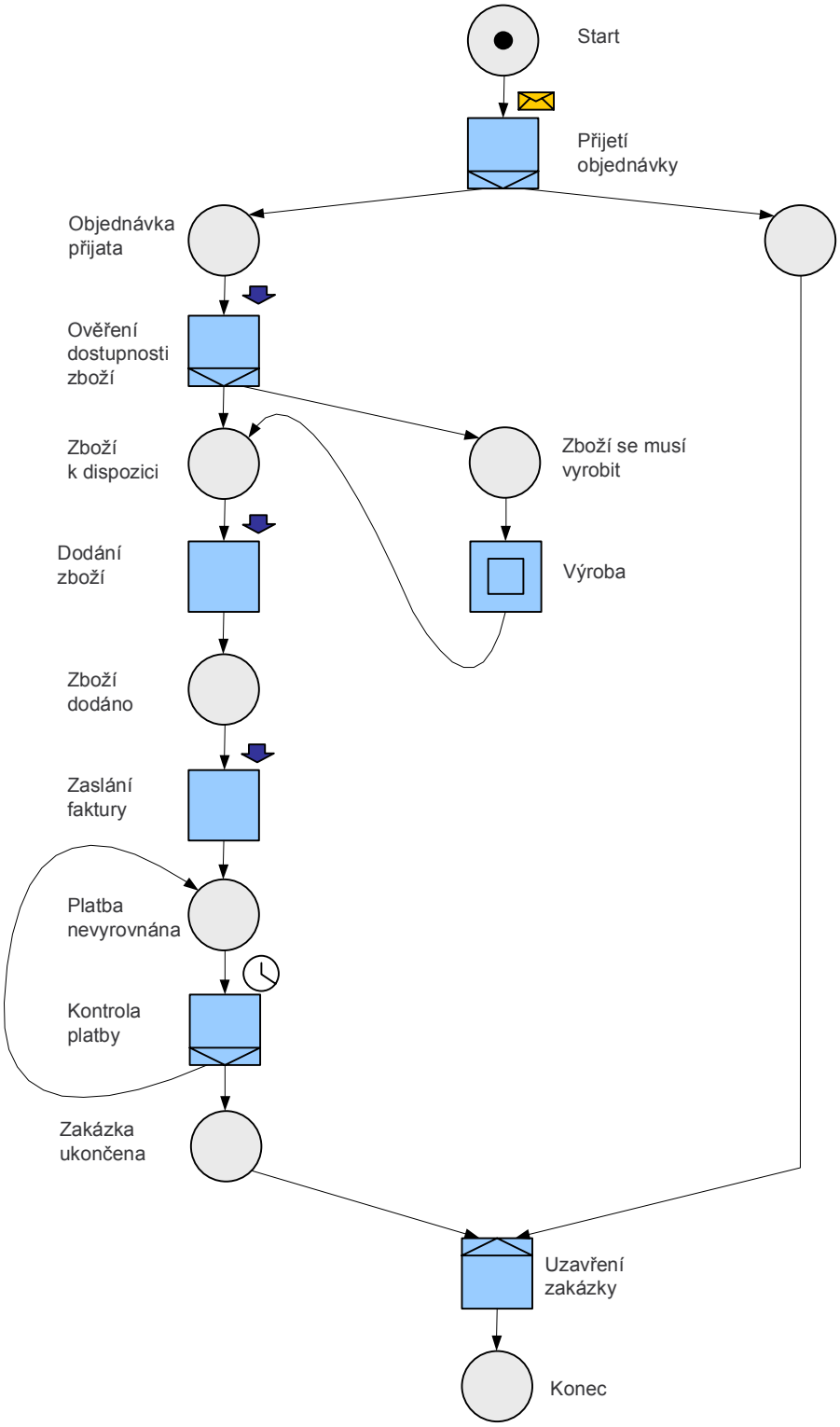
Na základě všech těchto uvedených požadavků může model realizace zakázky z **příklad 2.1** vypadat tak, jak je znázorněn na obrázku 3.18.



Obr. 3.18: Specifikace realizace zakázky pomocí WF-sítě

Z modelu vyplývá, že proces může být zahájen až v okamžiku přijetí externí události (*Přijetí objednávky*). Ta je reprezentována došlou objednávkou. Další činnosti vyžadují zdroje, kromě těch, které jsou spuštěny automaticky (*Zahájení výroby* a *Uzavření zakázky*). Jediná aktivita podmíněna časovým signálem je *Kontrola platby*, která bude provedena až po uplynutí určitého časového intervalu (viz časové rozšíření Petriho sítí).

Stejně jako u Petriho sítí i v případě WF-sítí lze modelovat strukturování procesů do podprocesů pomocí principů hierarchizace (obr. 3.19).



Obr. 3.19: Hierarchická dekompozice u WF-sítí

Výroba je v tomto případě nahrazena vnořenou WF-sítí.

❑ Formální definice WF-sítí

Stejně jako v minulé kapitole, tak i zde si formalizujme to, co bylo popsáno zatím pouze na intuitivní úrovni. Nejdříve si ale definujme pojem *cesta* (*path*) v Petriho sítích:

Definice 3.9 (*Cesta*). Nechť je dána Petriho síť $PN = (P, T, F)$. Cesta z uzlu $u_1 \in P \cup T$ do uzlu $u_n \in P \cup T$ je posloupnost $(u_1, u_2, ..., u_n)$ taková, že $(u_i, u_{i+1}) \in F$ pro $1 \leq i \leq k-1$.

Jinými slovy řečeno, uzlem rozumíme přechod nebo místo, přičemž platí, že u_1 je předchůdcem u_2, u_2 je předchůdcem u_3 atd. až po poslední uzel u_n, který již nemá svého následníka

WF-síť je formálně definována následujícím způsobem:

Definice 3.10 (*WF-síť*). Petriho síť $PN = (P, T, F)$ je WF-síť, právě když platí:

1. Existuje právě jedno počáteční místo $i \in P$ tak, že $\bullet i = \emptyset$.
2. Existuje právě jedno koncové místo $o \in P$ tak, že $o\bullet = \emptyset$.
3. Každý uzel $u \in P \cup T$ leží na cestě od počátečního místa i do koncového místa o.

První dvě vlastnosti jsou zřejmé. Třetí vyjadřuje fakt, že každé místo a každý přechod leží na cestě od počátečního do koncového místa. Petriho síť modelující workflow by tedy neměla mít „ztracené“ aktivity a podmínky. Díky tomuto požadavku, každa aktivita a každá podmínka může být dosažena z počátečního místa průchodem několika šipek znázorňující tokovou relaci. Stejně tak je možné z libovolného místa či aktivity obdobným způsobem dosáhnout koncového místa. Místa a přechody, které tomuto požadavku nevyhovují, nemohou přispět úspěšnému ukončení procesu.

Pozn.: Nesmíme zaměňovat pojmy dosažitelnost stavu a být na cestě. První pojem se týká značení, tedy stavů Petriho sítě, zatímco druhý popisuje její strukturální vlastnost.

Shrnutí pojmů 3.3.

WF-sítě.

Řídící struktury.

Mechanismy spuštění aktivit.

Otázky 3.3.

1. Jaké jsou požadavky kladené na workflow z hlediska dalšího rozšíření Petriho sítí?
2. Jaké jsou mechanismy spuštění aktivit?

Úlohy k řešení 3.3.

Navrhněte WF-síť pro následující proces zásilkové služby. Mějme dány následující aktivity: *Přijetí objednávky*, *Odeslání faktury*, *Přijetí platby*, *Odeslání zboží* a *Zrušení zakázky*. Po přijetí objednávky je zákazníkovi odeslána faktura. Pokud se do určitého času platba neobjeví na účtu (externí událost aktivity *Přijetí platby*), nemůže být zboží odesláno a zakázka je zrušena.

3.4. Analýza vlastností byznys procesů

Výklad

Spolehlivě specifikované workflow

V kapitole o Petriho sítích jsme uvedli některé jejich nejdůležitější vlastnosti. Poněvadž WF-sítě jsou Petriho sítěmi, platí pro ně stejná pravidla ověřování jejich vlastností a dosažitelností stavů. V [7] je však uvedena ještě jedna klíčová vlastnost, která se však týká pouze WF-sítí. Tato vlastnost se nazývá *spolehlivost* (*soundness*). Neformálně jsme tuto vlastnost již zmínili v minulé kapitole. V podstatě spolehlostivost představuje splnění následujících minimálních požadavků kladených na byznys proces: *proces neobsahuje žádné nepotřebné aktivity a každý případ (instance) procesu musí být kompletně ukončen*. Kompletně ukončen znamená, že neexistuje žádná aktivita či podmínka týkající se daného případu, která by byla aktivní i po ukončení procesu. Převedeno do řeči WF-sítí. Neexistence nepotřebných aktivit znamená, že všechny přechody musí být dosažitelné z počátečního stavu a z nich musí být dosažitelný také koncový stav. Kompletní ukončení lze interpretovat tak, že pro každou značku na počátečním místě sítě se po konečném počtu provedení přechodů objeví značka na místě koncovém, přičemž všechna ostatní místa jsou prázdná. Formálně můžeme výše uvedené vyjádřit následujícím způsobem:

Definice 3.11 (*Spolehlivost*). Nechť je dána WF-síť *PN* = (*P, T, F*) s počátečním místem $i \in P$ a koncovým místem $o \in P$. Pro počáteční stav M_0 platí $M_0(i) = 1 \wedge \forall p \in P \setminus \{i\} : M_0(p) = 0$. Pro koncový stav M_{end} platí $M_{end}(o) = 1 \wedge \forall p \in P \setminus \{o\} : M_{end}(p) = 0$. Proces modelovaný pomocí WF-sítě *PN* je *spolehlivý*, právě když platí:

1. Pro každý stav *M* dosažitelný ze stavu M_0 existuje posloupnost přechodů vedoucí ze stavu *M* do koncového stavu M_{end}. Formálně: $\forall M : \left(M_0 \xrightarrow{*} M\right) \Rightarrow \left(M \xrightarrow{*} M_{end}\right)$.
2. Koncový stav M_{end} je jediný dosažitelný stav z počátečního stavu M_0, který obsahuje na místě *o* značku. Formálně: $\forall M : \left(M_0 \xrightarrow{*} M \wedge M \geq M_{end}\right) \Rightarrow \left(M = M_{end}\right)$.
3. WF-síť PN neobsahuje přechody, které nemohou být provedeny. Formálně: $\forall t \in T \; \exists M, M' : M_0 \xrightarrow{*} M \xrightarrow{t} M'$.

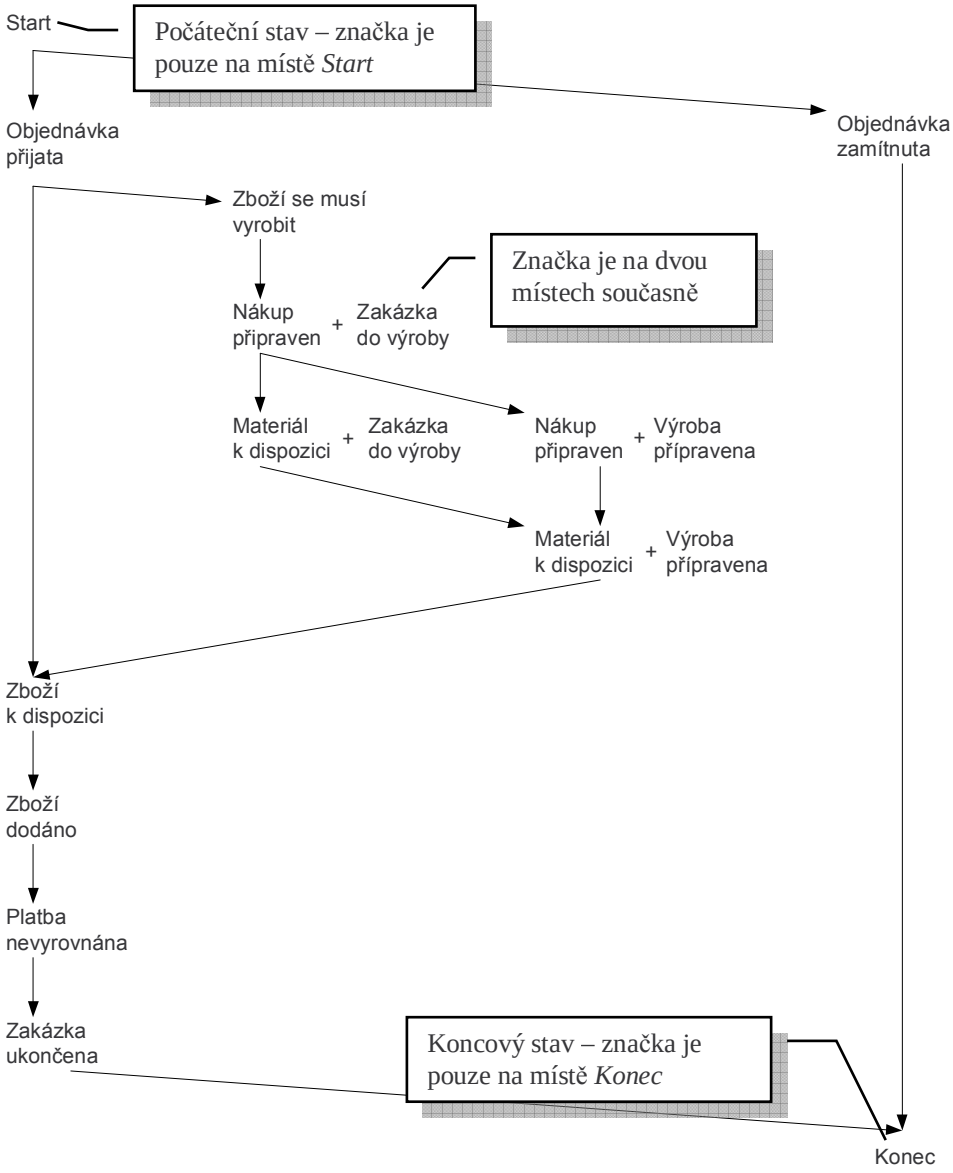
Otázka, která logicky vyplývá z výše uvedeného zní: jakými postupy jsme schopni ověřit spolehlivost pomocí WF-sítě specifikovaného procesu?

Verifikace s použitím počítače

Ověření, že navržený proces je skutečně spolehlivý, spočívá v ověření splnění tří výše definovaných požadavků (**definice 3.11**). K tomuto účelu lze využít graf dosažitelnosti. Tento graf musí mít právě jeden počáteční uzel a právě jeden koncový uzel odpovídající počátečnímu a koncovému značení WF-sítě (jedna značka pouze na počátečním resp. koncovém místě). Dále tento graf musí doložit, že pro každou aktivitu existuje v tomto grafu přechod, který odpovídá provedení této aktivity.

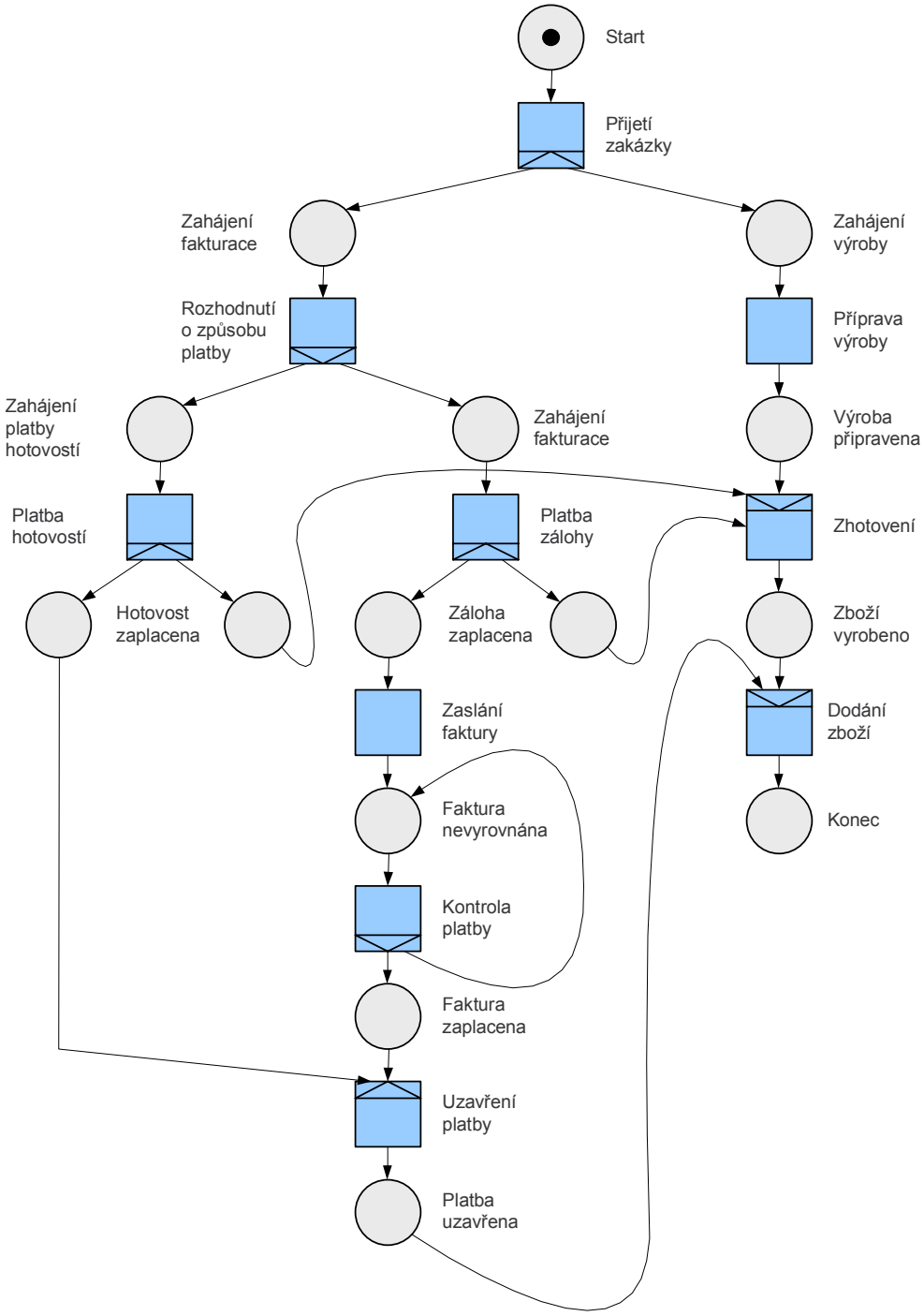
Dokumentujme si použití grafu dosažitelnosti k účelu ověření spolehlivosti modelu procesu realizace zakázky specifikovaného pomocí WF-sítě (obr. 3.18). Z důvodů lepší orientace nebudeme používat k popisu stavu vektor, ale formule typu p1+p3, pokud je na místech p1 a p3 značka (viz poznámka **definice 3.3**). Graf dosažitelnosti (obr. 3.20) má skutečně jediný počáteční a jediný koncový uzel. Pokud je značka na místě Konec, není na žádném jiném místě sítě. Při sestavení grafu byly použity všechny přechody (aktivity) a ze všech stavů dosažitelných z počátečního stavu se lze dostat i do stavu koncového. Model realizace zakázky vytvořený pomocí WF-sítě je tedy spolehlivý. Příkladem

nespolehlivého procesu (obr. 3.21) je WF-síť, která modeluje modifikanou realizaci zakázky z **příkladu 2.2**. V tomto případě by došlo k zablokování (deadlock) procesu ve stavech *Zboží vyrobeno+Záloha zaplacena+Platba uzavřena* nebo *Zboží vyrobeno+Hotovost zaplacena+Platba uzavřena* podle toho, zda zákazník platil zálohovým způsobem nebo celou částku hotově. Tato informace je opět dohledatelná v grafu dosažitelnosti.



Obr. 3.20: Graf dosažitelnosti WF-sítě realizace zakázky

Použití grafu dosažitelnosti s sebou ale nese dva základní problémy. Jak již vyplývá z názvu tohoto odstavce, je nutné při analýze spolehlivosti využít počítače, protože jen trochu složitější procesy vedou k velmi velkým grafům dosažitelnosti. Navíc, při kumulaci míst na jednom z míst sítě, by docházelo ke generování nekonečně velkého grafu dosažitelnosti odpovídající nekonečnému počtu nově vznikajích stavů.



Obr. 3.21: Chybně definovaný proces

Řešení uvedených problémů se nabízí cestou překladu vlastnosti *spolehlivosti* na dvě představené vlastnosti klasických Petriho sítí, kterými jsou *živost* a *omezenost* sítě. Tento postup vychází z principu „zkratování“ WF-sítě tím způsobem, že se propojí koncové místo s počátečním pomocí přidáním přechodu *t**. Mimochodem, takového postupu jsme již jednou použili v kapitole 3.2 (obr. 3.11). *Pokud takto nakrátko propojená WF-síť je živá a omezená, pak je také spolehlivá a naopak* [7]. Výhoda této transformace problému *spolehlivosti* na ověření *živosti* a *omezenosti* spočívá především v tom, že dnes, díky výzkumu v oblasti Petriho sítí, existují softwarové nástroje umožňující poslední dvě vlastnosti efektivně ověřit. Navíc, lze těmito nástroji i zjistit, co je příčinou nespolehlivosti sítě. My

si ale ještě ukážeme jiný přístup jak vytvářet spolehlivé modely workflow. Přístup, který nevyžaduje pomoc počítače a je plně v rukou tvůrce těchto modelů.

❑ Metody návrhu spolehlivých procesů

Základní úvaha stojící za požadavkem vytváření *spolehlivých* modelů workflow je, že se pokusíme vytvořit model procesu z komponent, které jsou již *spolehlivé*. Jinými slovy řečeno, pokusíme se o využití postupů běžných v praktickém životě, kdy podmínkou spolehlivosti systému je mít k dispozici i spolehlivé komponenty, ze kterých je celek sestaven. Tato idea je plně aplikovatelná i v oblasti WF-sítí. Jediný další požadavek, který přidáme ke *spolehlivosti* takových komponent a z nich vytvářených větších sítí je i to, že všechny tyto WF-sítě musí být *bezpečné* – počet značek na jejích místech nesmí být větší než 1 (definice 3.8).

Metoda vytváření *spolehlivých* a *bezpečných* sítí vychází z následujících principů. Mějme dvě WF-sítě *X* a *Y*, které jsou *spolehlivé* a *bezpečné*. První z nich obsahuje přechod *t*, který má právě jedno vstupní a jedno výstupní místo. Provedeme substituci přechodu *t* sítí *Y* tak, že počáteční místo *i* sítě *Y* odpovídá vstupnímu místu přechodu *t* a koncové místo *o* sítě *Y* odpovídá výstupnímu místu přechodu *t*. Obdržíme tak novou síť, která je opět *spolehlivá* a *bezpečná* [7]. Tyto vlastnosti výsledné sítě vyplývají z toho, že vložená WF-síť se chová jako původní přechod, tedy spotřebuje značku na svém vstupním místě a po konečném počtu kroků produkuje právě jednu značku na své výstupní místo. Bezpečnost sítí požadujeme proto, že v okamžiku, kdy budou na vstupním místě substituovaného přechodu *t* dvě a více značek, pak vložená síť nemusí garantovat *spolehlivost* a svým chováním tak odpovídat původnímu přechodu *t*. Vlastnost spolehlivosti je totiž ověřována pouze pro případ, kdy na počátečním místě je právě jedna značka. Formalizovat výše uvedené lze pomocí matematické věty o sestavitelnosti [7]:

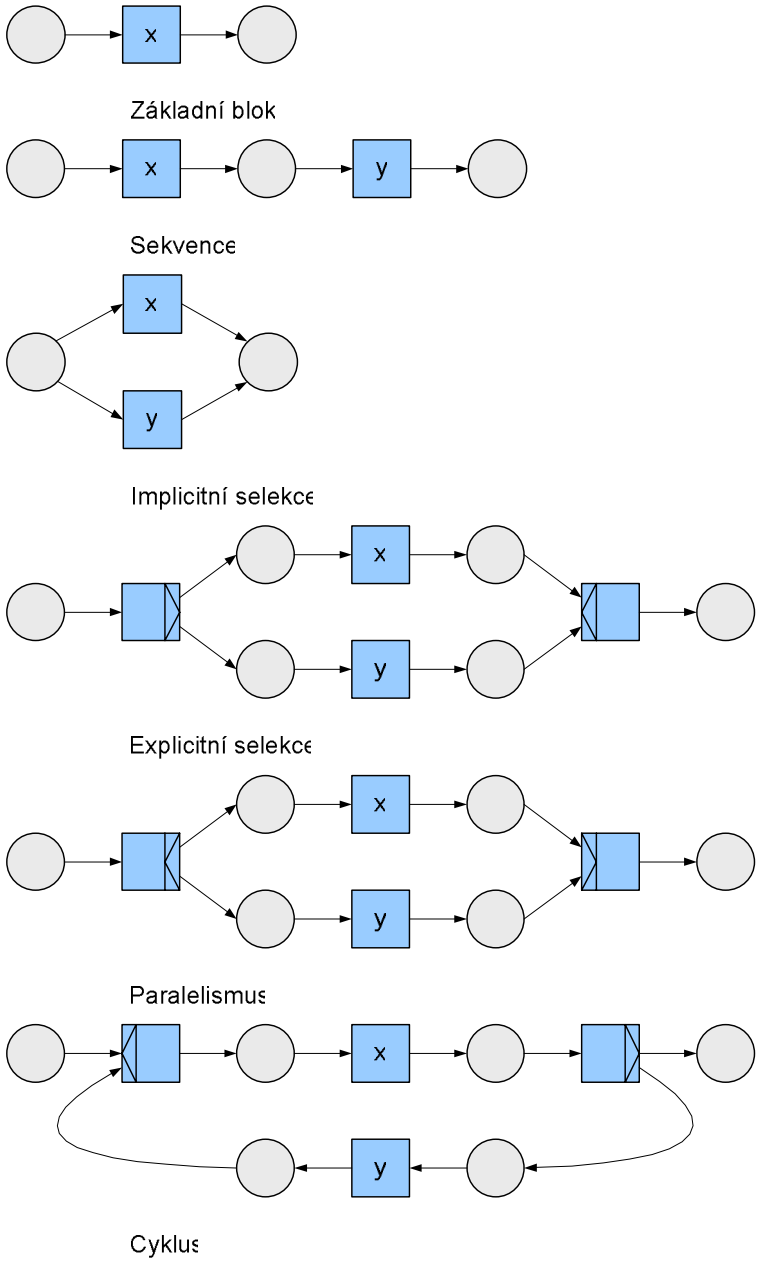
Věta 3.1 (*Sestavitelnost*): Nechť jsou dány dvě WF-sítě $PN_1 = (P_1, T_1, F_1)$ a $PN_2 = (P_2, T_2, F_2)$ tak, že $T_1 \cap T_2 = \emptyset$, $P_1 \cap P_2 = \{i, o\}$ a $t^+ \in T_1$. $PN_3 = (P_3, T_3, F_3)$ je WF-síť získaná substitucí přechodu t^+ sítí PN_2, t.j. $P_3 = P_1 \cup P_2$, $T_3 = (T_1 \setminus \{t^+\}) \cup T_2$ a

$$\begin{aligned} F_3 = & \{(x,y) \in F_1 : x \neq t^+ \wedge y \neq t^+\} \cup \\ & \{(x,y) \in F_2 : \{x,y\} \cap \{i,o\} = \emptyset\} \cup \\ & \{(x,y) \in P_1 \times T_2 : (x,t^+) \in F_1 \wedge (i,y) \in F_2\} \cup \\ & \{(x,y) \in T_2 \times P_1 : (t^+,y) \in F_1 \wedge (x,o) \in F_2\}. \end{aligned}$$

Nechť je dáno počáteční značení M_0 všech tří sítí předpisem $M_0(i) = 1 \wedge \forall p \in P_k \setminus \{i\} : M_0(p) = 0$ pro $k \in \{1,2,3\}$. Síť PN_1 a PN_2 jsou *spolehlivé* a *bezpečné*, právě když je *spolehlivá* a *bezpečná* síť PN_3.

Důkaz: Podstatou důkazu je to, že každý stav (značení) PN_3 může být mapován na stav PN_1 a PN_2 a naopak. Navíc je podstatné, že všechny sítě jsou bezpečné. Pokud by totiž došlo k vícenásobné aktivaci podsítě PN_2, pak by její chování nemohlo být slučitelné s prostým provedením přechodu t^+ v síti PN_1. *Spolehlivost* sítě PN_2 není zaručena v případě, že na jejím počátečním místě jsou dvě a více značek. Podrobný důkaz lze nalézt v [6].

Pokusme se nyní o ukázku toho, jak lze vytvářet *spolehlivé* a *bezpečné* sítě s využitím komponent. Nejprve si definujme množinu „stavebních bloků“ (komponent), které budeme k účelu sestavení výsledné sítě používat (obr. 3.22).

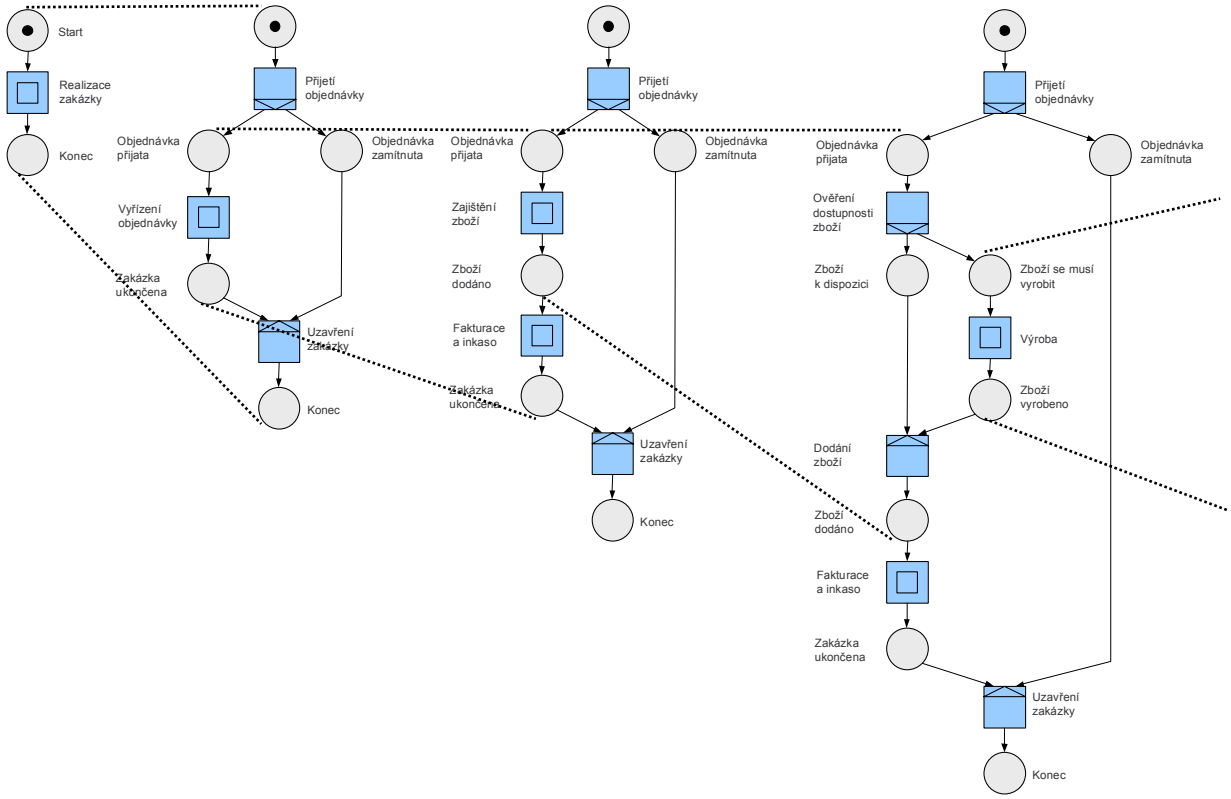


Obr. 3.22: Dobře strukturované komponenty

Množina navržených komponent má jednu další zajímavou vlastnost. Všechny tyto komponenty jsou tzv. *dobře strukturované*. Tato vlastnost vyjadřuje fakt, že všechna použitá rozdělení toku procesu jsou zase odpovídajícím způsobem sloučena. Každý AND-split je ukončen AND-join a OR-split je ukončen OR-join. Výsledná Petriho síť vytvořená z takových komponent je opět *dobře strukturována*, vyvážena z hlediska použitých „logických spojek". Velkou výhodou takto *dobře strukturované* Petriho sítě je, že *spolehlivá* a *dobře strukturovaná* síť je také *bezpečná* [7]. Budeme-li tedy používat *dobře strukturovaných* komponent, máme zaručeno že jsou také *bezpečné* a lze z nich vytvářet nové *spolehlivé* WF-sítě.

Množina použitých komponent nemusí být nutně uzavřená, ale lze ji postupně rozšiřovat i o nově vytvořené komponenty. Jediná podmínka je, že použitá komponenta je opět *spolehlivá* a *bezpečná*, přičemž jsme si ukázali že druhá vlastnost je zaručena při použití dobře strukturovaných komponent.

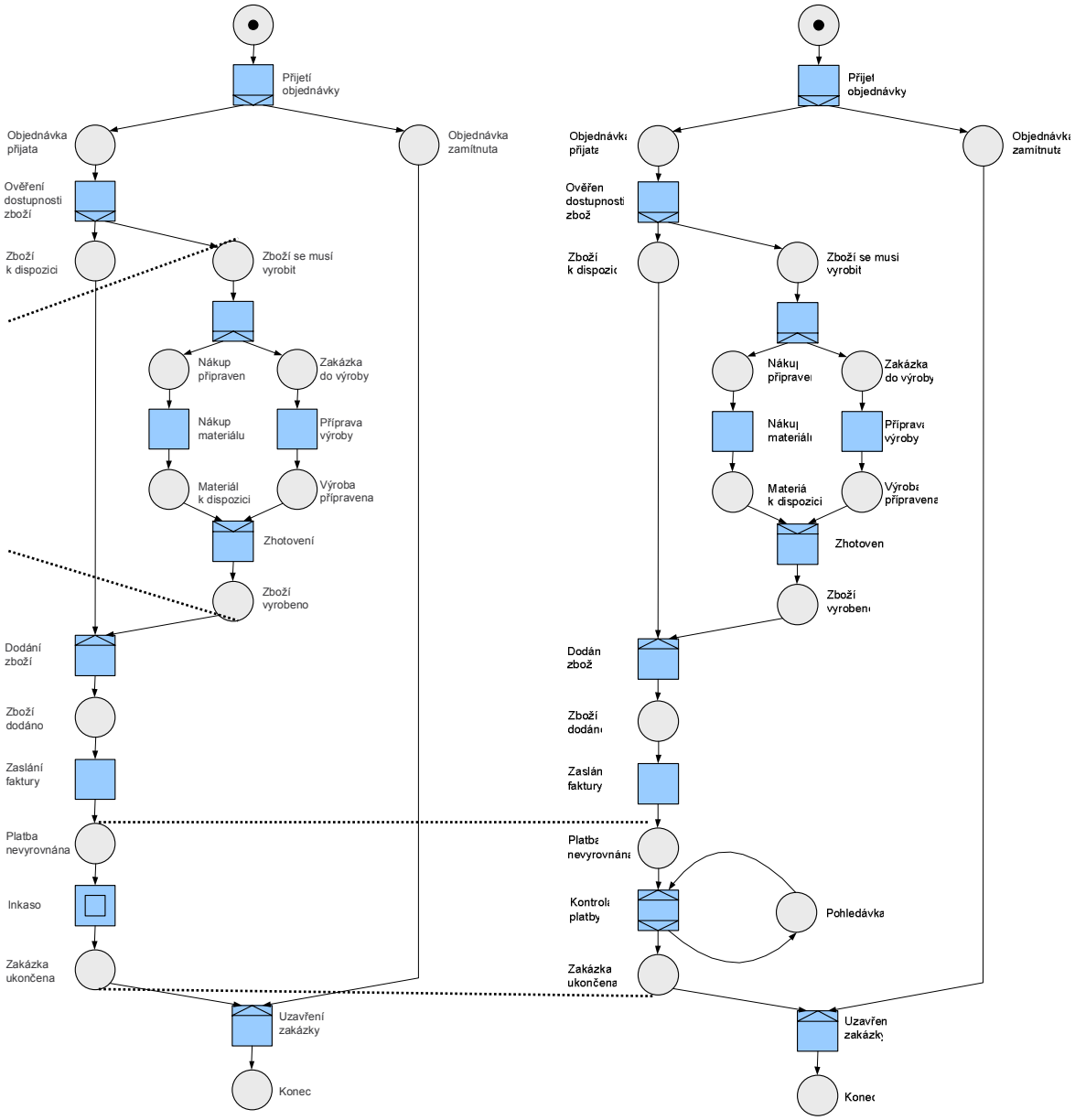
Ukažme si nyní na následujících konstrukcích, jak lze postupem skládání z komponent vytvořit WF-síť, která je *spolehlivá*. Opět pro srovnání použijeme příklad 2.1 realizace zakázky. Podstata tohoto spočívá v postupném nahrazování jednotlivých přechodů složitější komponentou dle pravidel popsaných výše (**věta 3.1**). Celý postup je dokumentován na následujících obrázcích 3.23 a 3.24.



Obr. 3.23: Konstrukce spolehlivé WF-sítě

Nejprve jsme počátaeční síť nahradili komponentou *Explicitní selekce*. Jeden z přechodů, který obsahuje standardní komponenta jsme vypustili, protože není aktivita, která by jej reprezentovala. Tímto se nedopouštíme žádné chyby, protože i takto modifikovaná komponenta je i nadále *spolehlivá*, *dobře strukturovaná* a tím i *bezpečná*. Následuje náhrada přechodu jedné z větví komponentou *Sekvence*. První z přechodů je opět substituován *Explicitní selekcí*. Jedna z větví této komponenty je nahrazena *Paralelismem*. Nakonec je aplikována modifikovaná komponenta *Cyklus* na přechod *Inkaso* s cílem modelovat opakující se provádění kontroly platby. Posloupnost substitucí tedy byla následující: *Základní blok* → *Explicitní selekce* → *Sekvence* → *Explicitní selekce* → *Paralelismu* → *Cyklus*. Záznam těchto substitucí je důležitá především z důvodů provedení pozdějších změn ve specifikaci workflow. V případě, že je třeba provést změnu, je nutné postupně odstraňovat přidané komponenty až do bodu, který se má změnit. Provede se změna cestou aplikace jiné komponety a workflow se opět vyskládá zpět do nové podoby. Jedině tímto způsobem lze zajistit, že i změněná WF-síť je *spolehlivá*.

Ve srovnání s původním modelem 3.18, vytvořeným neřízeným způsobem, je tato WF-síť lépe strukturována. Je však důležité dodat, že *spolehlivá* může být i síť, která není *dobře strukturována*. Dobrým zvykem je však vytvářet modely cestou použití dobře strukturovaných komponent, které zaručují nejen korektně vytvořený model workflow, ale tento model je také lépe čitelný a lze jej snadno měnit při zachování požadovaných vlastností.



Obr. 3.24: Konstrukce spolehlivé WF-sítě pokr.

Shrnutí pojmů 3.4.

Spolehlivost.

Počítačová verifikace.

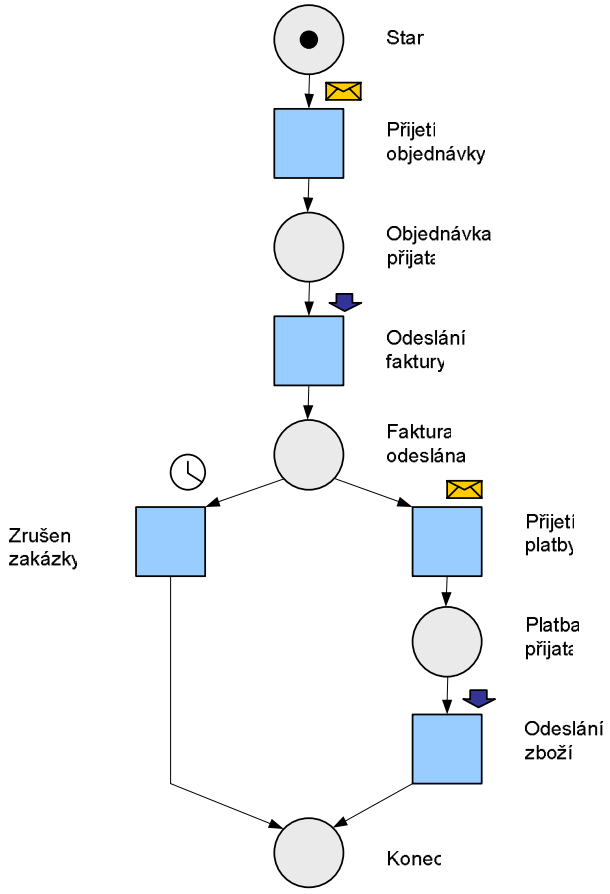
Skládání procesů z komponent.

Otázky 3.4.

1. Jak je definována spolehlivost?
2. Jakým způsobem lze ověřit spolehlivost WF-sítě bez použití grafu dosažitelnosti?
3. Jaké vlastnosti musí splňovat komponenty určené pro vytváření procesů skládáním?

Úlohy k řešení 3.4.

Mějme dánu WF-síť (obr. 3.25). Propojte tuto síť přechodem t^* nakrátko. Je tato síť živá a omezená? Je také spolehlivá?



Obr. 3.25: Proces zásilkové služby

3.5. Formalizace a verifikace neformálně definovaných metod

Výklad

Formalizace EPC

V minulé kapitole jsme se pokusili dokumentovat, co je největším přínosem formálního přístupu ke specifikaci byznys procesů – workflow. Poněvadž nelze počítat s tím, že by uživatelé opustili své softwarové nástroje, které implementují metody uvedené v kap.2, ukážeme si jak lze tyto neformální, příp. semiformální přístupy doplnit o žádoucí přesnost v jejich syntaxi i sémantice. Cílem je vytvořené modely formálně specifikovat a také ověřit jejich požadované vlastnosti (např. *dosažitelnost koncového stavu* apod.). Jedna z velmi často používaných metod je postavena na EPC diagramech. Cestu jak je lze formalizovat takto specifikované procesy si ukážeme na následujících stránkách [5].

Začněme formální definice syntaxe EPC:

Definice 3.12 (*Event-driven Process Chain (1)*): Event-driven Process Chain je uspořádaná pětice *EPC* = (*E*, *F*, *C*, *T*, *A*):

- *E* je konečná množina *událostí*,
- *F* je konečná množina *funkcí*,
- *C* je konečná množina *logických spojek* (*konektorů*),
- $T : C \to \{\wedge, XOR, \vee\}$ je *funkce mapující konektory na typ konektoru* a
- $A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$ je množina *propojení*.

Máme tedy tři typy uzlů: události (*E*), funkce (aktivity) (*F*) a konektory (*C*). Typ konektoru je dán funkcí $T : T(c)$, která každému konektoru přiřádí jednu z logických spojek $(\wedge, XOR, \vee)$. Relace A popisuje jak jsou události, aktivity a konektory navzájem propojeny. Z této relace vyplývá, že není možné vzájemně propojit dvě funkce ani dvě události.

Definice 3.13 (*Cesta, elementární cesta*): Nechť je dán *EPC* = (*E*, *F*, *C*, *T*, *A*). Cesta *p* (*path*) z uzlu (*node*) n_1 do uzlu n_k je posloupnost $(n_1, n_2, ..., n_n)$ taková, že $(n_i, n_{i+1}) \in A$ pro $1 \le i \le k-1$. Cesta *p* je *elementární*, právě když pro libovolné uzly n_i a n_j na cestě *p* platí $i \neq j \Rightarrow n_i \neq n_j$.

Pojem cesta nám umožňuje zavedení definice C_{EF} (množina konektorů, která se nachází na cestě od události k funkci) a C_{FE} (množina konektorů na cestě od funkce k události). Na základě funkce *T* můžeme rozdělit množinu konektorů *C* na diskjunktní množiny $C_\wedge$, $C_\vee$ a C_{XOR}. Množiny C_J a C_S se používají k odlišení spojek typu join a split.

Definice 3.14 (N, $C_\wedge$, $C_\vee$, C_{XOR}, $\bullet$, C_J, C_S, C_{EF}, C_{FE}): Nechť je dán *EPC* = (*E*, *F*, *C*, *T*, *A*).

- $N = E \cup F \cup C$ je množina *uzlů EPC*.
- $C_\wedge = \{c \in C : T(c) = \wedge\}$
- $C_\vee = \{c \in C : T(c) = \vee\}$
- $C_{XOR} = \{c \in C : T(c) = XOR\}$

- Pro $n \in N$:
  $\bullet n = \{m : (m,n) \in A\}$ je množina *vstupních* uzlů uzlu *n* a
  $n\bullet = \{m : (n,m) \in A\}$ je množina *výstupních* uzlů uzlu *n*.
- $C_J = \{c \in C : |\bullet c| \geq 2\}$ je množina *join* konektorů.
- $C_S = \{c \in C : |c \bullet| \geq 2\}$ je množina *split* konektorů.
- $C_{EF} \subseteq C$ tak, že $c \in C_{EF}$, právě když existuje cesta $p = (n_1, n_2, ..., n_{k-1}, n_k)$ tak, že $n_1 \in E \wedge n_2, ..., n_{k-1} \in C \wedge n_k \in F$ a $c \in \{n_2, ..., n_{k-1}\}$.
- $C_{FE} \subseteq C$ tak, že $c \in C_{FE}$, právě když existuje cesta $p = (n_1, n_2, ..., n_{k-1}, n_k)$ tak, že $n_1 \in F \wedge n_2, ..., n_{k-1} \in C \wedge n_k \in E$ a $c \in \{n_2, ..., n_{k-1}\}$.

Předcházející definice nám umožňuje specifkovat další požadavky kladené na EPC diagram:

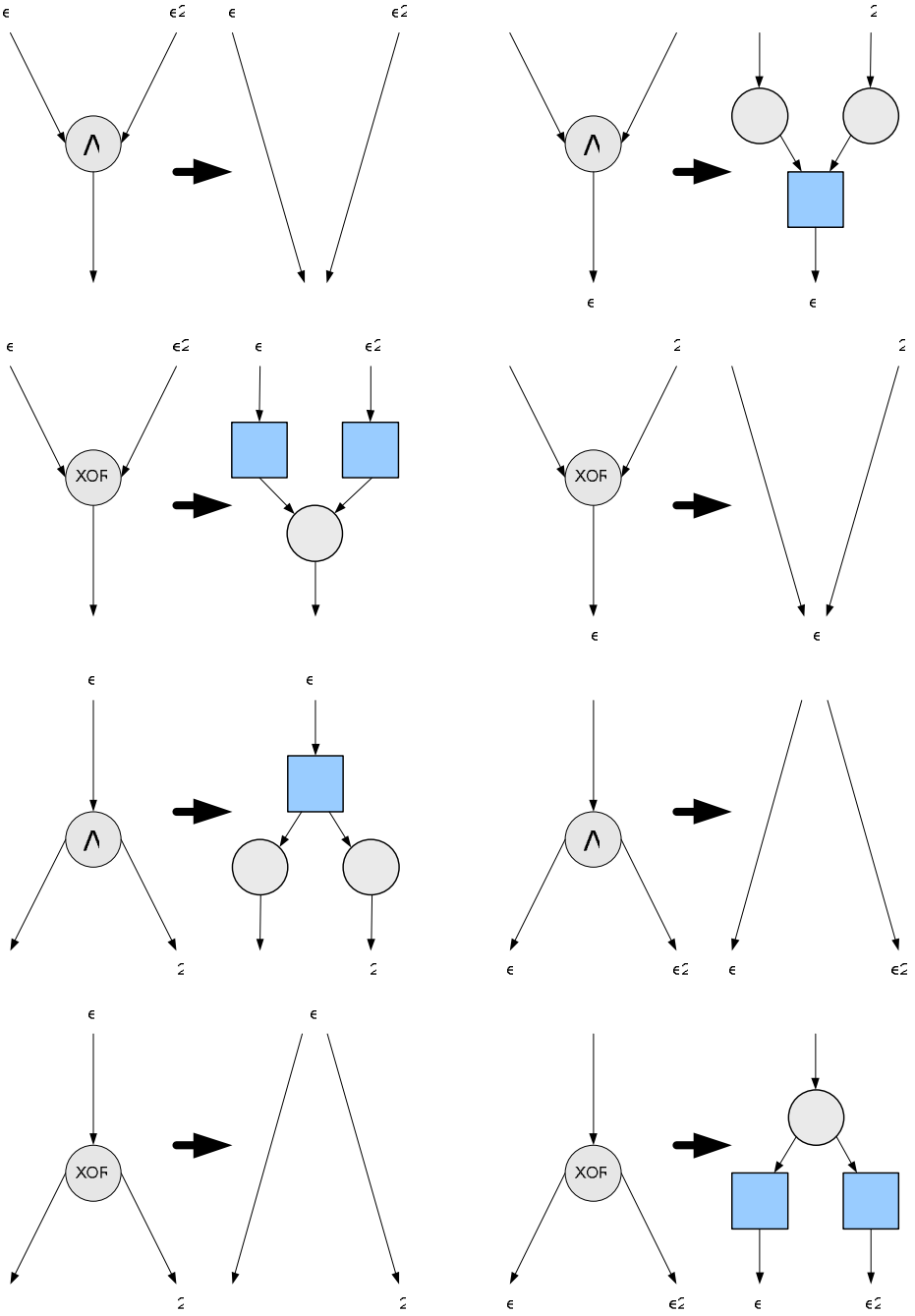
Definice 3.15 (*Event-driven Process Chain (2)*): Event-driven process chain *EPC* = (*E*, *F*, *C*, *T*, *A*) splňuje následující požadavky:

- Množiny *E*, *F* a *C* jsou *navzájem disjunktní*, t.j. $E \cap F = \emptyset \wedge E \cap C = \emptyset \wedge F \cap C = \emptyset$.
- Pro každou událost $e \in E : |\bullet e| \leq 1 \wedge |e \bullet| \leq 1$.
- Existuje alespoň jedna událost $e \in E : |\bullet e| \leq 0$ nazvaná *počáteční událost*.
- Existuje alespoň jedna událost $e \in E : |e \bullet| \leq 0$ nazvaná *koncová událost*.
- Pro každou funkci $f \in F : |\bullet f| = 1 \wedge |f \bullet| = 1$.
- Pro každý konektor $c \in C : |\bullet c| \geq 1 \wedge |c \bullet| \geq 1$.
- Graf reprezentující EPC je slabě propojený, což znamená, že pro každé dva uzly $n_1, n_2 \in N : (n_1, n_2) \in (A \cup A^{-1})^*$.
- C_J a C_S rozdělují *C*, t.j. $C_J \cap C_S = \emptyset \wedge C_j \cup C_S = \emptyset$.
- C_{EF} a C_{FE} rozdělují *C*, t.j. $C_{EF} \cap C_{FE} = \emptyset \wedge C_{EF} \cup C_{FE} = \emptyset$.

První požadavek vyjadřuje, že každý uzel má své jméno. Ostatní požadavky jsou vázány na relaci propojení A. Každá událost může mít maximálně jednoho předchůdce a jednoho následníka. Události, které nemají svého předchůdce jsou počáteční událostí, události bez následníka jsou koncové. Každá funkce má právě jednoho předchůdce a jednoho následníka. Konektory na základě toho, zda se jedná o *split* resp. *join* mají jednoho resp. více předchůdců a více resp. jednoho následníka. Pro každé dva uzly existuje cesta, která je propojuje nezávisle na orientaci šipek.

Splněním všech výše uvedených požadavků dostáváme syntakticky správně specifikované EPC modely procesů.

Dále si ukážeme, jak definovat sémantiku EPC. K tomuto účelu využijeme Petriho sítí, které sémantiku mají přesně dánu. V podstatě to znamená, že se pokusíme o namapování EPC na Petriho sítě, a to tím způsobem, že událost bude mapována na místo a funkce na přechod Petriho sítě. Složitější je problém konektorů, které mohou vést na množinu propojení Petriho sítě nebo na malou síť míst a přechodů (obr. 3.26). Konektor $\vee$, určený v EPC pro spojení alternativních toků, navíc nemá jasně daný význam, a proto jej z dalšího postupu vyloučíme a budeme používat výhradně kontektory $\wedge$ a *XOR*.



Obr. 3.26: Mapování konektorů EPC na kontrukce Petriho sítě

Nyní budeme uvedené principy mapování EPC na Petriho sítě přesně specifikovat v následující definici:

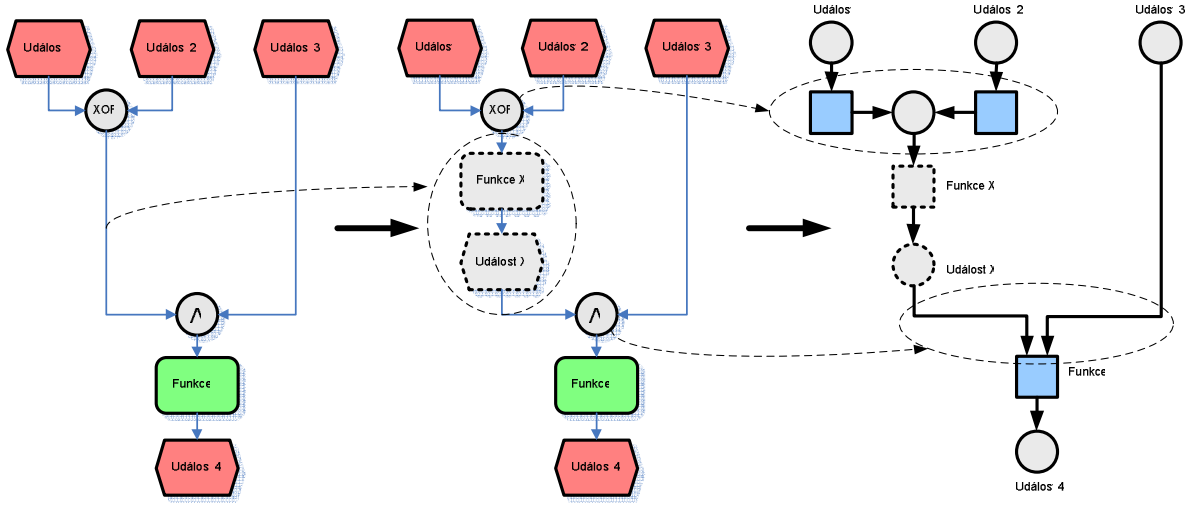
Definice 3.16 (*Mapování*): Nechť je dán Event-driven process chain *EPC* = (*E*, *F*, *C*, *T*, *A*) s $C_{\vee} = \varnothing \wedge A \cap (C \times C) = 0$. $PN(EPC) = (P^{PN}, T^{PN}, F^{PN})$ je Petriho síť generovaná *EPC* tak, že platí $P^{PN} = E \cup \left(\bigcup_{c \in C} P_c^{PN}\right)$, $T^{PN} = F \cup \left(\bigcup_{c \in C} T_c^{PN}\right)$ a $F^{PN} = (A \cap ((E \times F) \cup (F \times E))) \cup \left(\bigcup_{c \in C} F_c^{PN}\right)$.

Tabulka 3.1 popisuje jak jsou definovány P_c^{PN}, T_c^{PN} a F_c^{PN}.

	P_c^{PN}	T_c^{PN}	F_c^{PN}
$c \in C_{EF} \cap C_J \cap C_\wedge$	$\varnothing$	$\varnothing$	$\{(x,y): x \in \bullet c \wedge y \in c\bullet\}$
$c \in C_{FE} \cap C_J \cap C_\wedge$	$\{p_x^c : x \in \bullet c\}$	$\{t^C\}$	$\{(x, p_x^c): x \in \bullet c\} \cup \{(p_x^c, t^c): x \in \bullet c\} \cup \{(t^c, x): x \in c\bullet\}$
$c \in C_{EF} \cap C_J \cap C_{XOR}$	$\{p^C\}$	$\{t_x^c : x \in \bullet c\}$	$\{(x, t_x^c): x \in \bullet c\} \cup \{(t_x^c, p^c): x \in \bullet c\} \cup \{(p^c, x): x \in c\bullet\}$
$c \in C_{FE} \cap C_J \cap C_{XOR}$	$\varnothing$	$\varnothing$	$\{(x,y): x \in \bullet c \wedge y \in c\bullet\}$
$c \in C_{EF} \cap C_S \cap C_\wedge$	$\{p_x^c : x \in c\bullet\}$	$\{t^C\}$	$\{(x, t^c): x \in \bullet c\} \cup \{(t^c, p_x^c): x \in c\bullet\} \cup \{(p_x^c, x): x \in c\bullet\}$
$c \in C_{FE} \cap C_S \cap C_\wedge$	$\varnothing$	$\varnothing$	$\{(x,y): x \in \bullet c \wedge y \in c\bullet\}$
$c \in C_{EF} \cap C_S \cap C_{XOR}$	$\varnothing$	$\varnothing$	$\{(x,y): x \in \bullet c \wedge y \in c\bullet\}$
$c \in C_{FE} \cap C_S \cap C_{XOR}$	$\{p^C\}$	$\{t_x^c : x \in c\bullet\}$	$\{(x, p^c): x \in \bullet c\} \cup \{(p^c, t_x^c): x \in c\bullet\} \cup \{(t_x^c, x): x \in c\bullet\}$

Tabulka 3.1: Mapování EPC konektorů $c \in C$ na místa, přechody a propojení

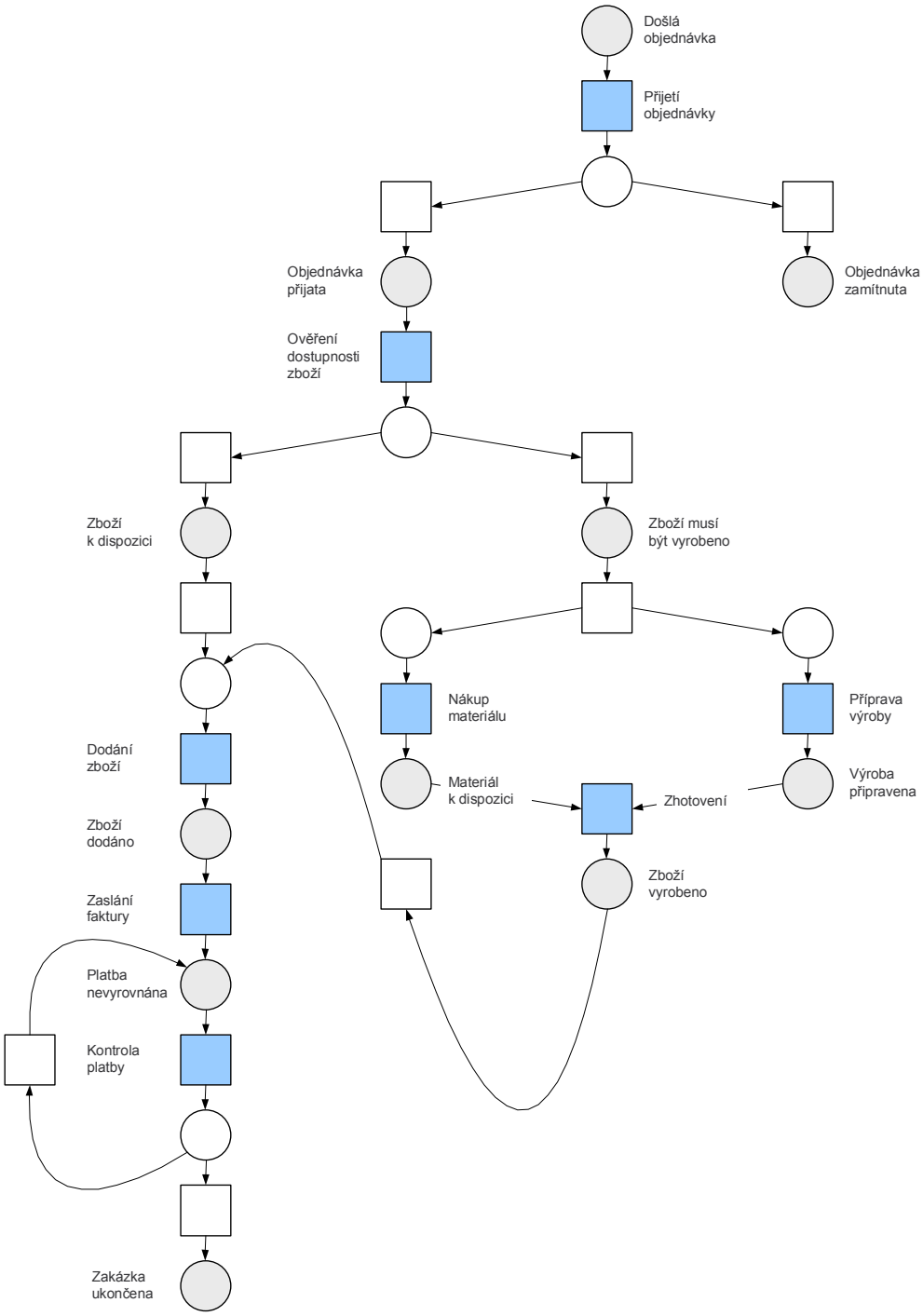
Daná definice má ale ještě jedno omezení. Předpoklad $A \cap (C \times C) = 0$ požaduje, že konektory jsou vždy propojeny pouze s funkcemi a událostmi. Nelze tedy spojit více konektorů za sebou. Namísto dalšího rozšíření tabulky 3.1, je možné použít alternativní přístup. Ten spočívá v nahrazení propojení (šipky) mezi dvěmi konektory fiktivní událostí a funkcí. Výsledný EPC model pak lze bez problémů mapovat na Petriho síť podle výše definovaných pravidel (obr. 3.27).



Obr. 3.27: Transformace EPC modelu na Petriho síť

Z obrázku je patrné, že se nejprve do EPC modelu včlenily fiktivní *Funkce X* a *Událost X* namísto propojení mezi dvěmi konektory *XOR-join* a $\wedge$*-join*. Pak na *XOR-join* byla aplikována substituce z 3. řádku tabulky 3.1. Poslední substituce bylo nahrazení $\wedge$*-join* strukturou Petriho sítě z 1. řádku téže tabulky.

Nyní není problém transformovat libovolný EPC model do podoby Petri sítě s jedinou výjimkou. EPC model procesu nesmí obsahovat kontektor $c \in C_\vee$. Dokumentovat výše uvedené můžeme na *PN*(*EPC*) procesu realizace zakázky (obr. 3.28) generovaného z původního EPC modelu (2.7).



Obr. 3.28: Transformace EPC diagramu realizace zakázky na Petriho síť

Nevybarvené (fiktivní) přechody a místa, které nemají svůj vzor v původním modelu, byly dodány transformací dle tabulky 3.1. Podstatné je to, že z původního neformálního popisu jsme získali model, který má své zázemí v přesně dané teorii a naskýtá se nám tak možnost ověřit tento proces pomocí postupů představených v minulých kapitolách.

❑ Verifikace EPC modelů

Klíčovou vlastností, kterou jsme ověřovali u modelů workflow specifikovaných pomocí WF-sítí byla tzv. *spolehlivost*. Abychom mohli tutéž vlastnost ověřovat také u EPC modelů, omezíme se na takové EPC, které jsou regulární [5].

Definice 3.17 (*Regularita*): Event-driven process chain *EPC* je *regulární*, právě když:

- *EPC* má právě dvě speciální události e_{start} a e_{final}. Událost e_{start} je počátečním uzlem, pro který platí $\bullet e_{start} = \emptyset$. Událost e_{final} je koncovým uzlem, pro který platí $e_{final} \bullet = \emptyset$.
- Každý uzel $n \in N$ je na cestě e_{start} z do e_{final}.

Pozornější čtenář si uvědomil, že na obecné EPC modely je kladen stejný požadavek jako na Petriho sítě, abychom je mohli považovat za WF-sítě (viz definice 3.10). Odtud již nejsme daleko od definování vlastnosti *spolehlivosti* pro EPC modely:

Definice 3.18 (*Spolehlivost (EPC)*): Regulární Event-driven process chain EPC je *spolehlivý*, právě když:

1. Pro každý stav *M* dosažitelný z *počátečního stavu* (pouze událost e_{start} je platná) existuje zřetězení událostí a funkcí (aktivit), které vedou do stavu koncového (pouze událost e_{final} je platná).
2. Koncový stav je jediný stav dosažitelný z počátečního stavu, kdy je platná pouze událost e_{final}.
3. Neexistuje nedostupná funkce (aktivita), t.j. pro každou funkci $f \in F$, existuje zřetězení událostí a funkcí, které vedou k *f*.

Podíváme-li se na specifikaci byznys procesu popsaného na obr. 2.10, zjistíme, že tento model není *spolehlivý*, i když splňuje předpoklady *regulárního* Event-driven process chain. Na tomto obrázku je také znázorněno místo, které problém způsobuje. Je evidentní, že čím je proces složitější, tím hůře bude možné nalézt taková problémová místa. Může se tak stát, že proces selže až při běhu informačního systému, který bude workflow implementovat. Abychom předešli těmto nežádoucím situacím, je tedy nutné všechny modely ověřit. Možnost transformovat EPC modely na Petriho sítě a využít tak postupů použitelných u Petriho sítí, skýtá reálnou a efektivní šanci jak dosáhnout skutečně kvalitních návrhů byznys procesů a jejich workflow.

Stejně jako v případě WF-sítí, tak i zde však můžeme pro dobrý návrh udělat více už během vytváření modelů. Není vždy nutné ověřovat až hotové a mnohy velmi složité modely. Cesta k tomuto vede přes již zmíněné skládání procesů z komponent, které jsou *dobře strukturovány*. Opět je žádoucí zajistit správné párování logických spojek, čili *XOR-join* musí zakončovat tok rozdělený spojkou *XOR-split* a stejně tak *AND-split* musí být ukončen pomocí *AND-join*. Pokud proces nesplňuje podmínku *dobrého strukturování*, tak by měl být ověřen, zda-li je skutečně korektně navržen. Pozor, neznamená to, že proces, který není *dobře strukturován*, nemůže být *spolehlivý*. Pouze je s ním třeba nakládat s vyšší obezřetností.

Ukázali jsem si, jak lze neformální metody formalizovat a zajistit tak přesnost a korektnost v nich vytářených modelů procesů. Je si však třeba také přiznat, že se jedná o netriviální postup vyžadující mnohem více usilí, než se na prvý pohled zdá. Odměnou je dosažení vysoké úrovně kvality návrhu byznys procesů.

Shrnutí pojmů 3.5.

Formální definice EPC.

Transformace EPC na Petriho sítě.

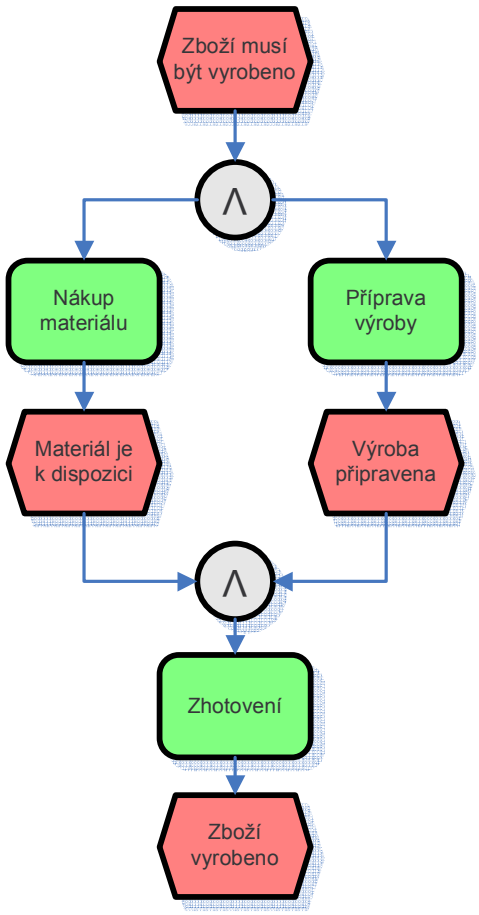
Ověření vlastností EPC modelů.

Otázky 3.5.

1. Co je to regulární EPC model a v čem se liší od obecných?
2. Definujte pojem spolehlivost EPC.

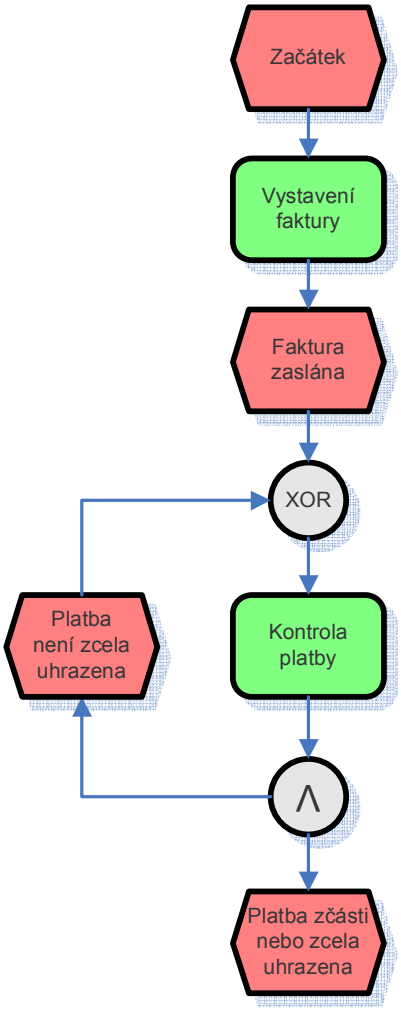
Úlohy k řešení 3.5.

1. Mějme dán model procesu výroby (obr. 3.29). Transformujte tento model na Petriho síť a ověřte, zda-li se jedná o WF-síť a zda-li je spolehlivá.



Obr. 3.29: EPC diagram procesu výroby

2. Mějme dán proces fakturace a inkasa umožňující platby po částech (obr. 3.30). Ověřte na nakrátko propojené WF-síti generované z EPC modelu, zda-li se jedná o spolehlivě definovaný proces.



Obr. 3.30: EPC diagram fakurace a inkasa

4. ZÁVĚR

Čas ke studiu kapitoly: 4 hodiny

Cíl Po prostudování tohoto odstavce budete znát

- některé softwarové nástroje používané k účelu byzny modelování.

4.1. Softwarové nástroje pro specifikaci a analýzu byznys modelů

Výklad

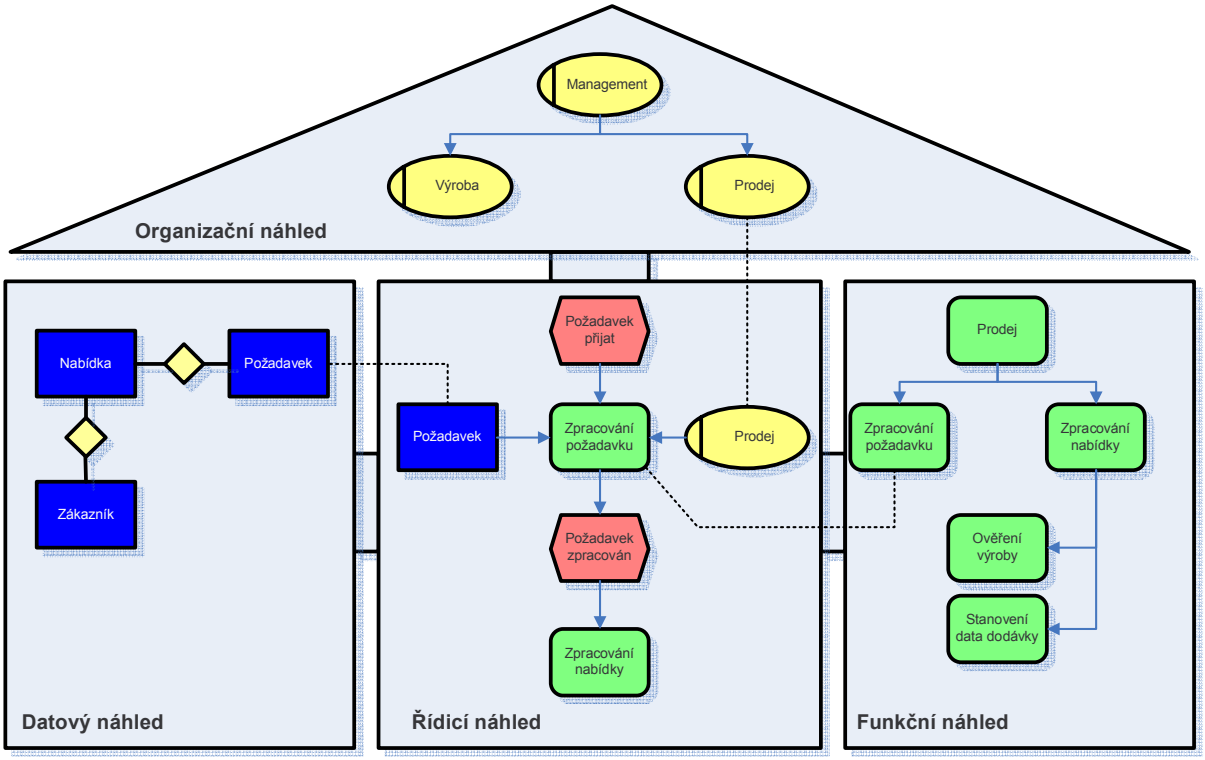
Cílem této kapitoly není popsat kompletní architekturu a obsluhu systémů používaných pro účely byznys modelování. Je ponecháno na čtenáři, aby využil uvedené odkazy na několik známých produktů z této oblasti a pokusil se v nich implementovat některé z příkladů uvedených v minulých kapitolách.

❑ Nástroje orientované na EPC diagramy

Snad nejznámějším softwarovým systémem pro modelování a řízení softwarových procesů je rodina produktů ARIS (ARchitecture of Integrated Systems) společnosti IDS-Scheer (www.ids-scheer.com). Tento nástroj specifikuje byznys proces ze všech jeho hledisek a v celé šíři. Jedná se o následující pohledy (obr. 4.1):

- *Organizační náhled* slouží ke specifikaci organizační struktury. Lze modelovat organizační jednotky, funkční místa, role i samotné zaměstnance.
- *Datový náhled* umožňuje specifikovat datový model, který je následně vázán na proces.
- *Funkční náhled* definuje funkce, které musí organizace plnit.
- *Řídicí náhled* slouží k provázání všech těchto abstrakcí s cílem popsat, jak je samotný proces vykonáván. Základem pro modelování je zde přístup postavený na eEPC.

Vytvořené procesy lze simulovat a tímto způsobem i ověřovat korektnost provedeného návrhu. V posledních verzích systém poskytuje i vazbu na nástroje postavené na jazyku UML. V současné době neexistuje volně dostupná verze.



Obr. 4.1: Schématické znázornění systému ARIS

❑ Nástroje orientované na UML

Poněvadž jazyk UML je dnes považován za de facto standard v oblasti specifikací modelů pro vývoj informačních a softwarových systémů obecně, existuje celá řada nástrojů, které jej implementují. Důležité je, že jazyk UML je dnes standardizován (www.uml.org) skupinou OMG (Object Management Group), což s sebou nese celou řadu výhod. Jedná se především o zajištění přenositelnosti modelů vytvořených v jazyce UML mezi různými softwarovými nástroji. Základem pro tuto přenositelnost je propracovaný meta-model, který je respektován výrobci softwaru pro modelování v jazyce UML. Kromě vysoce profesionálních systémů, jako je např. *Rational Rose* firmy IBM, však existuje i celá řada softwarových nástrojů, které ve své tzv. Community Edition jsou volně k dispozici uživatelům. Jeden příklad za všechny je systém *Poseidon for UML* firmy Gentleware (www.gentleware.com).

Použití těchto systémů má své výhody, ale i řadu nevýhod. Hlavní výhodou je, že modely specifikované v takových nástrojích lze dále ve stejném nástroji zpracovat pro potřeby softwarových specifikací. Nedochází tak ke zbytečným ztrátám a omylům při překladu byznys modelů do modelů softwarového díla. Navíc se díky standardizací jazyka UML usnadňuje komunikace mezi byznys partnery, kteří sice pracují na svých procesech relativně samostatně, ale dříve či později budou muset své procesy navzájem propojit. Společný modelovací jazyk tento postup výrazně usnadňuje.

Hlavní nevýhodou je fakt, že byznys modelování tvoří pouze menší podmnožinu funkcionality, které tyto softwarové nástroje poskytují. To znamená, že tyto systémy jsou výrazně složitější z hlediska jejich použití a vyžadují mnohdy náročné zaučení. Je také neefektivní investovat do takového nástroje značné finanční prostředky, když se stejně nepředpokládá jiné využití, než je právě byznys modelování.

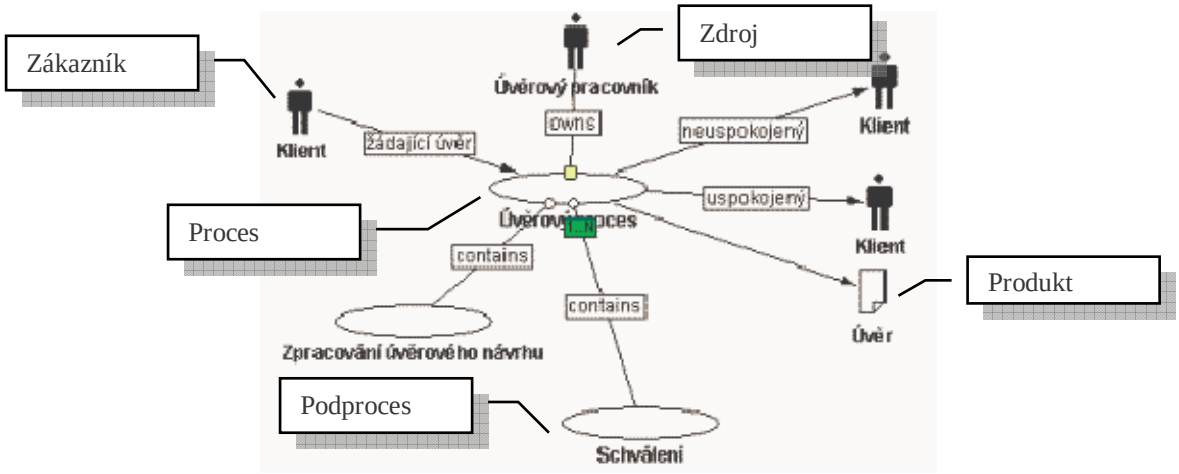
❑ Nástroje orientované na Petriho sítě

Metoda BPM (Business Process Modeling) reprezentuje původní nástroj autora využívající posledních výsledků výzkumu a vývoje v oblasti modelování procesů [4]. Základní charakteristiky této metody lze shrnout do následujících tří bodů:

- BPM je *formalizovaný* nástroj vizuálního modelování založený na teorii Petriho sítí vyšší úrovně;
- BPM umožňuje jak *strukturální analýzu* nutnou pro definování organizační struktury, tak analýzu prostřednictvím *simulace* nutnou ke kvantifikaci procesu (doba trvání, náklady ...). Definované modely lze následně použít i k počítačem podporovanému řízení a monitorování procesů;
- BPM je *nezávislý nástroj analýzy* umožňující definovat požadavky na informační systém.

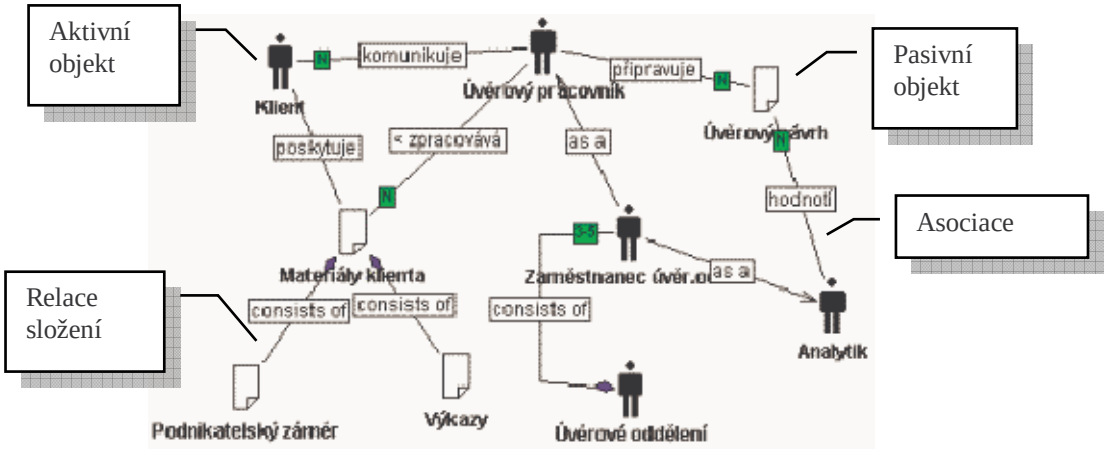
Vlastní vytvoření modelu libovolného procesu podniku je nezávislé na typu procesu (podpůrný či základní) i oblasti aplikace (výroba, bankovnictví či služby). Samotný model specifikuje podnikový proces ze tří relativně nezávislých pohledů - *funkčního*, *objektového* a *koordinačního*.

Funkční model identifikuje architekturu procesů včetně všech jejich zákazníků a produktů. Cílem je nalézt odpověď na otázku, jaké funkce jsou po podniku požadované, jaké procesy je realizují a jaká je jejich struktura (obr. 4.2).



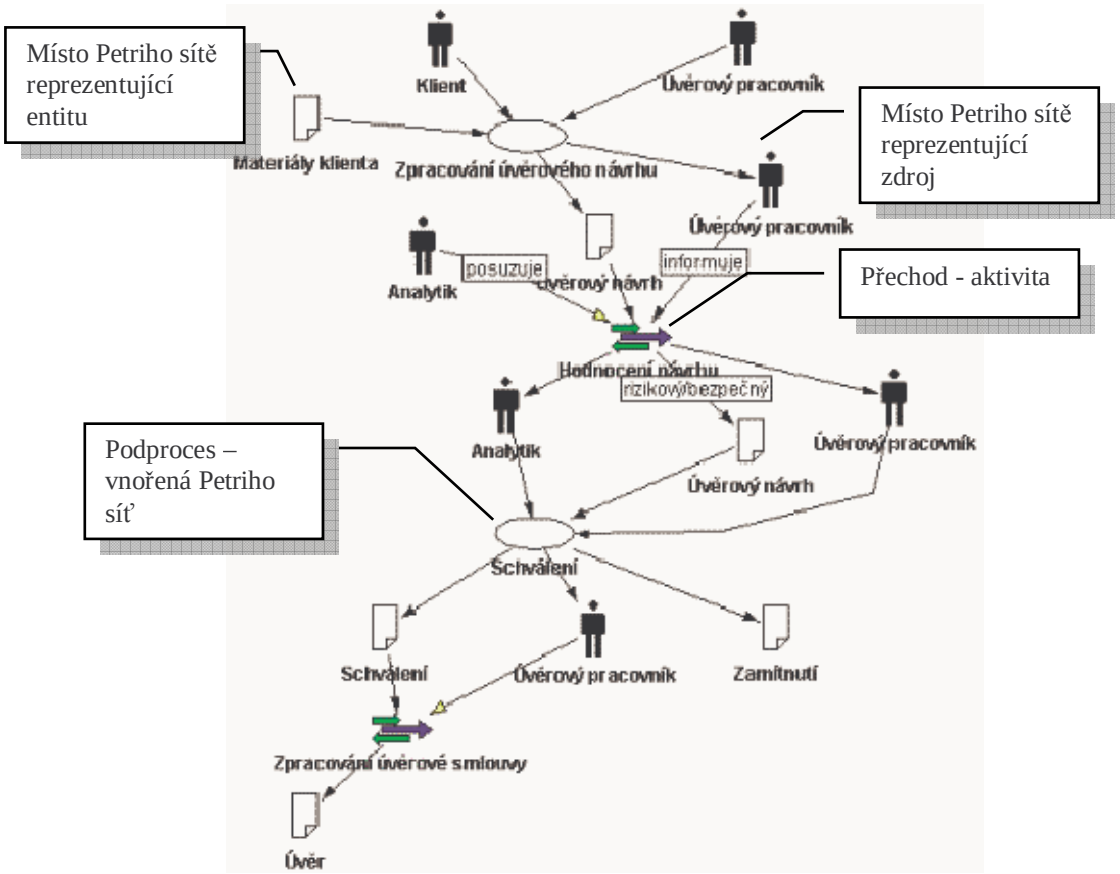
Obr. 4.2: Funkční model metody BPM

Objektový model definuje statickou strukturu entit (objektů), které jsou vyžadovány pro realizaci jednotlivých procesů a podprocesů. Jinými slovy se snaží odpovědět na otázku kým a čím procesy realizovat (obr. 4.3).



Obr. 4.3: Objektový model metody BPM

Koordinační model specifikuje jednotlivé činnosti včetně jejich řazení. Jinými slovy je zde hledána odpověď na otázku jak realizovat procesy. Koordinační aspekt modelování je jeden z nejdůležitějších, neboť umožňuje definovat souběžně vykonávané činnosti s přesným určením podmínek jejich paralelního provedení (obr. 4.4).



Obr. 4.4: Koordinační model metody BPM

Poslední koordinační model dokumetuje přímou návaznost na Petriho sítě s tím, že metoda BPM zavádí místa reprezetující nejen tok entit, ale také zdrojů. Správa a řízení zdrojů se tak stává součástí Petriho sítě.

Softwarový produkt implementující metodu BPM se jmenuje BPStudio a jeho verze s omezeným rozsahem modelů je volně k dispozici na stránkách autora http://vondrak.cs.vsb.cz/download.html .

4.2. Několik slov závěrem

Výklad

Závěrečné shrnutí

Cílem této publikace bylo seznámit čtenáře s metodami používanými pro účely byznys modelování. První část byla věnována problematice tzv. neformálních, nebo, lépe řečeno, semiformálních metod postavených na grafických jazycích. Byly diskutovány tři typy abstrakcí – *funkční* náhled, specifikace *chování* a *strukturální* popis. K těmto třem přístupům byly prezentovány jejich nejznámější zástupci. Pro účely funkční specifikace byla vybrána metoda IDEF, konkrétně IDEF0. Událostmi řízené procesní řetězce EPC sloužily k popisu chování, nebo-li popisu řídícího toku procesu. Poslední z uvedených abstrakcí reprezentovaná strukturálním popisem procesu nás uvedla prostřednictvím diagramu tříd do jazyka UML. Jazyk UML jsme následně použili k popisu demonstračního procesu realizace zakázky ve smyslu specifikace všech tří náhledů – funkčního, chování a strukturálního. Ukázali jsem si, co je společné pro všechny byznys procesy nezávisle na jejich modelovacích nástrojích a pokusili jsme se o sestavení univerzálního *meta-modelu* – modelu, který popisuje model. Co je pro všechny tyto metody typické? Snahou jejich tvůrců bylo nalezení výrazových prostředků (jazyka), který je zvládnutelný širokou komunitou odborníků tak, aby mohli mezi sebou snáze komunikovat a aby jimi vytvářené modely byly přesné a jednoznačné. Bohužel, právě posledně jmenované požadavky se dají jen stěží zajistit v případě chybějící přesně definované *syntaxe* a *sémantiky – formalizace* metody.

Problematika formálního přístupu byla náplní druhé části studijního materiálu. Pokud přistoupíme na to, že dříve či později námi návržené modely byznys procesů budou realizovány pomocí počítačů a tomu odpovídajícho softwarového vybavení (hovoříme o *workflow*), pak korektnost takových specifikací je absolutní nutností. Není sice problémem implementovat procesy, které byly ověřeny alespoň pomocí simulací, ale nikdy nemáme jistotu jejich 100% verifikace. Proto jsme se věnovali teorii *Petriho sítí* a její modifikaci pro potřeby modelování workflow. Zavedli jsme si formální definice jejich syntaxe i sémantiky. Vybrali jsme některé z důležitých vlastností Petriho sítí a nakonec jsme se zabývali problematikou tzv. *spolehlivého* procesu. Poněvadž asi nelze počítat s tím, že od tohoto okamžiku budou všichni zainteresování používat Petriho sítí (konkrétně *WF-sítí*) k modelování procesů, ukázali jsme si jak lze přistoupit k formalizaci neformálně definovaných metod. Konkrétně jsme si tento postup ukázali na populárních EPC modelech. Podstatou problému bylo nalezení cesty, jak transformovat pomocí EPC specifikované modely na Petriho sítě, které již dokážeme ověřovat díky celé řadě algoritmů používaných k jejich analýze. Jedná se sice o cestu netriviální, ale řešitelnou i v softwarových nástrojích, které výše uvedené metody implementují. Otevírá se tak cesta k tomu, že budeme moci nejen jednoduše vytvářet modely procesů, ale především tyto modely budou korektní.

O tom, že modelování byznys procesů je aktuálním tématem dnešních dnů není asi pochyb. Re-engineering firem, vylepšování jejich procesů, implementace moderních ERP systémů, propojování firem do sítí spolupráce a v neposlední řadě i systémy řízení jakosti jsou oblastmi, které si vynucují se věnovat problematice modelování byznys procesů. Za samozřejmost se při tom předpokládá korektnost takto popsaných procesů. Jak je vidět, ne všechny věci, které se zdají samozřejmé jsou samozřejmostí. V případě modelování procesů jsme se o tom již přesvědčili.

Jako autor tohoto studijního materiálu pevně věřím, že vás jeho obsah inspiroval, obohatil o vědomosti a znalosti, které pak v praktickém životě využijete. K tomu vám čtenářům přeji mnoho úspěchů.

Literatura

[1] Mayer, R.J., Painter, M.: IDEF Family of Methods, Technical Report, Knowledge Based Systems, Inc., College Station, TX, 1991

[2] Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language User Guide, Addison Wesley Longman, Inc., 1999

[3] Schmuller, J.: Teaching Yourself UML in 24 Hours, Sams, 1999

[4] Vondrak, I., Szturc, R., Kruzel, M.: Company Driven by Process Models, European Concurrent Engineering Conference ECEC '99, SCS, Erlangen-Nuremberg, Germany, pp. 188-193, 1999

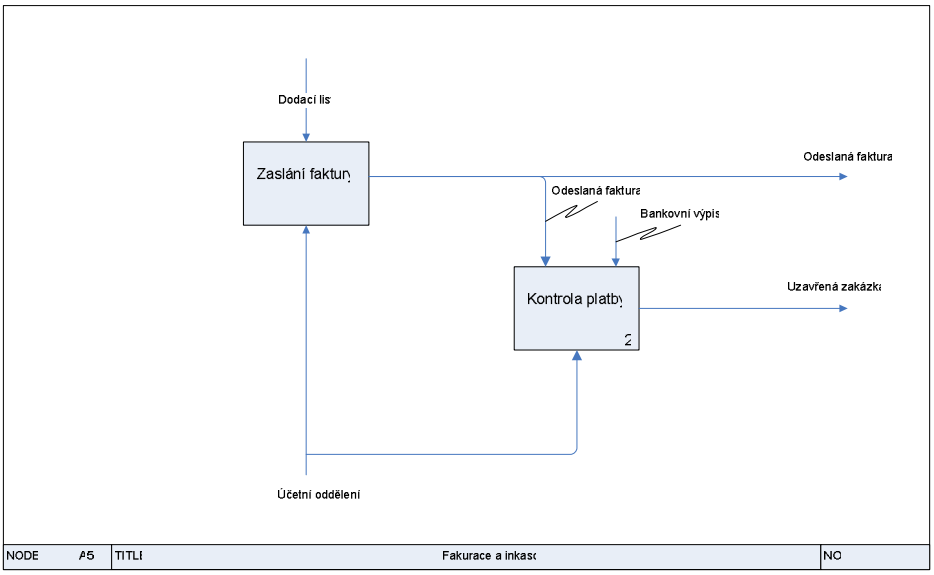
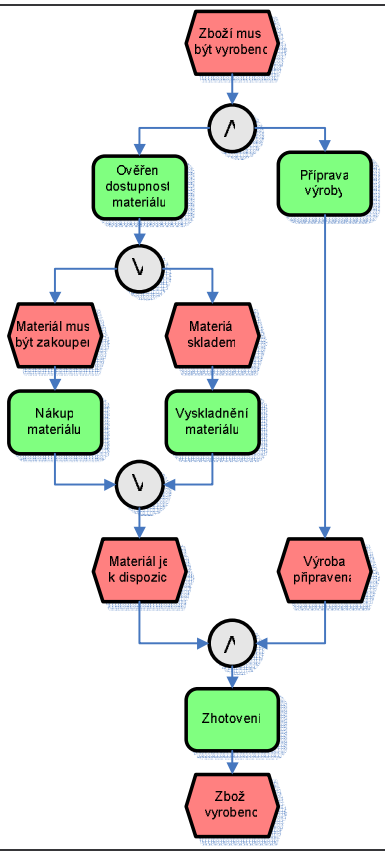
[5] Wil van der Aalst. Formalization and Verification of Event-driven Process Chains. Information and Software Technology, 41(10):639-650, 1999.

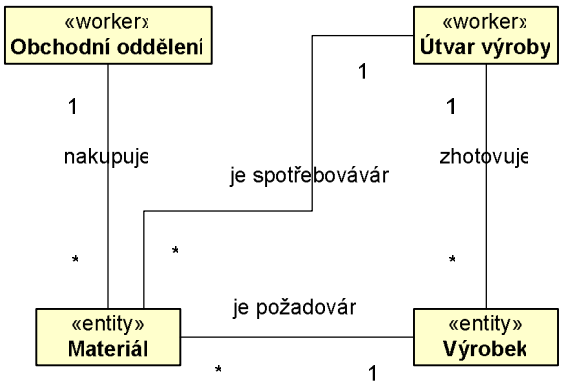
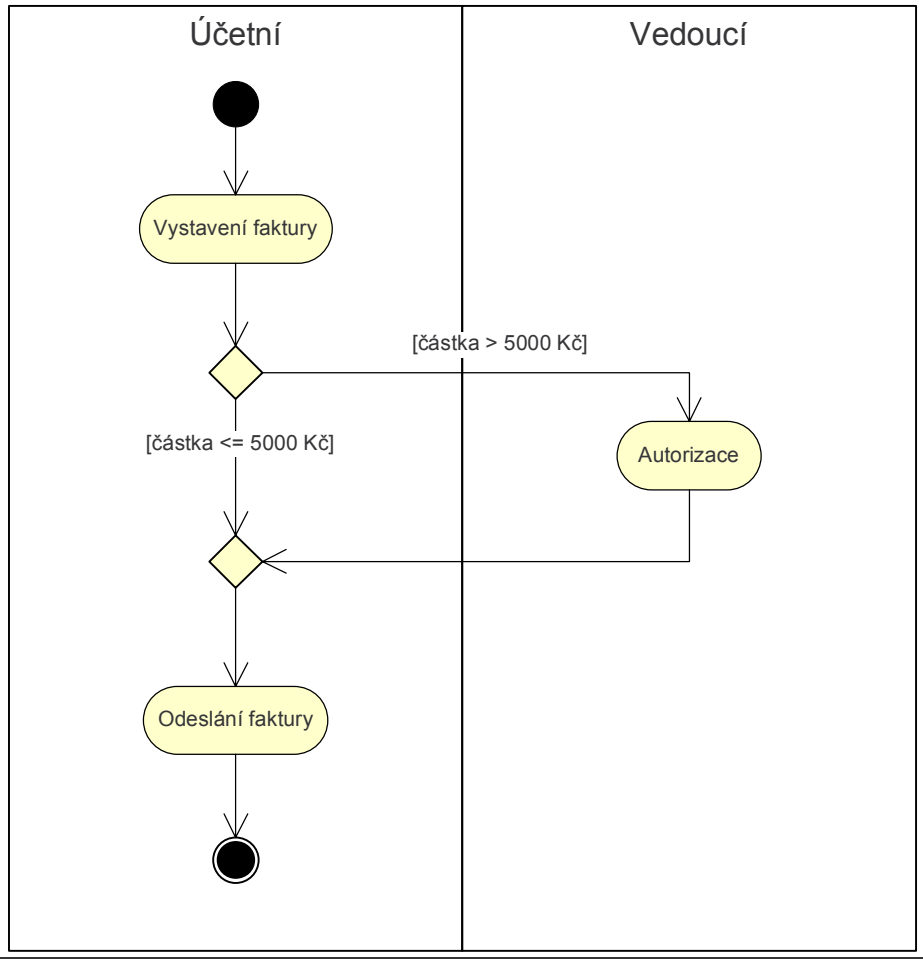
[6] Wil van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pages 161-183. Springer-Verlag, Berlin, 2000.

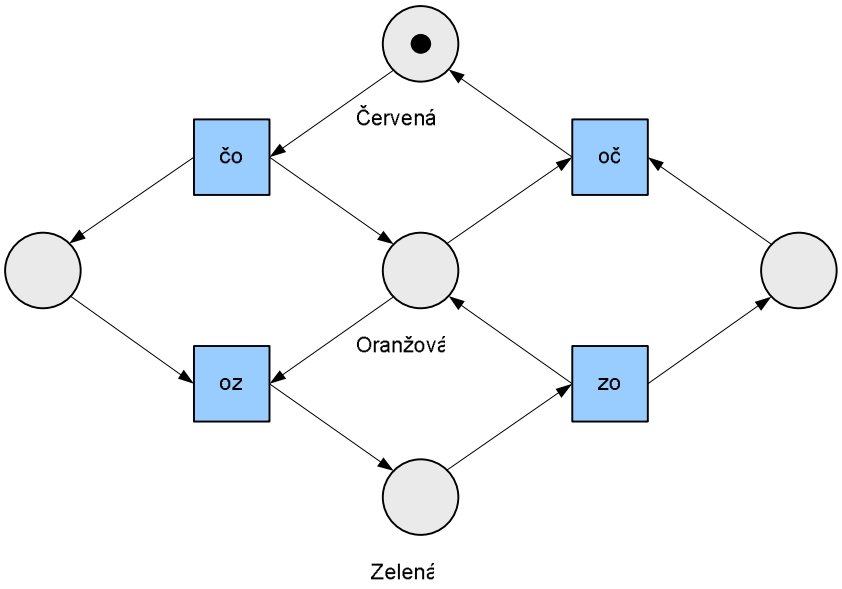
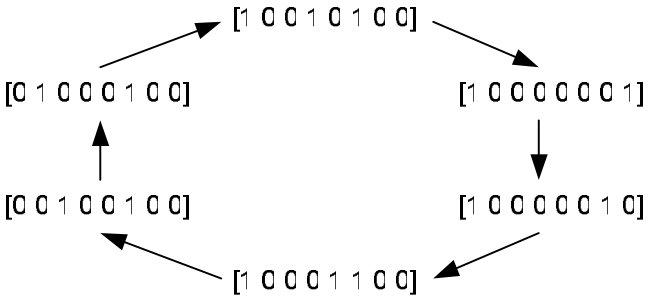
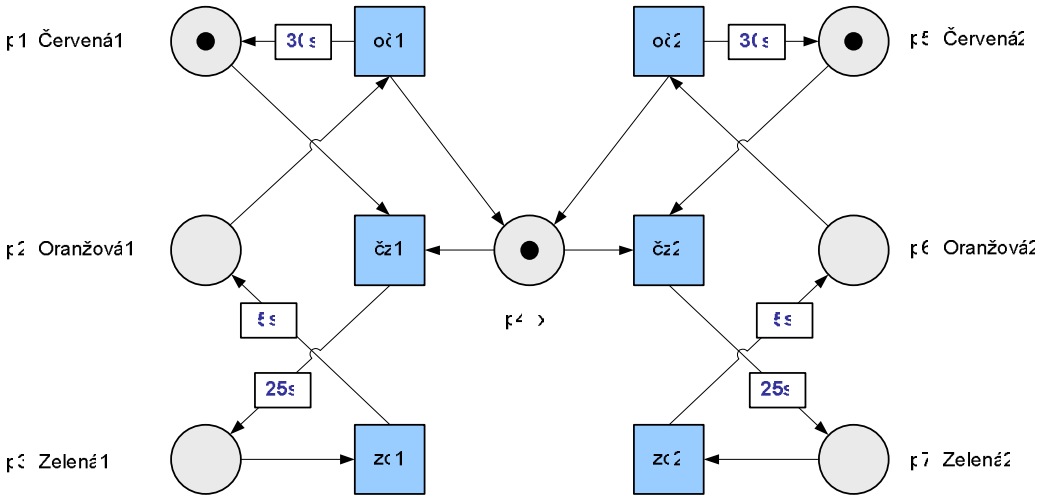
[7] Wil van der Aalst, Kees van Hee: Worklflow Management, Models, Methods, and Systems. MIT Press, 2002

[8] Češka, M.: Petriho sítě, Akademické nakladatelství CERM Brno, 1994

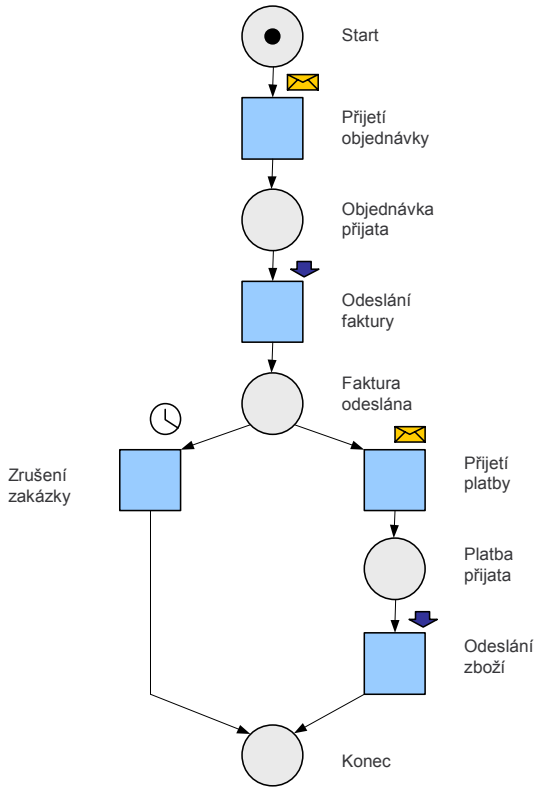
Klíč k řešení

Otázky	Odpovědi najdete vždy v předcházející kapitole.
Ú 2.2	
Ú 2.3	

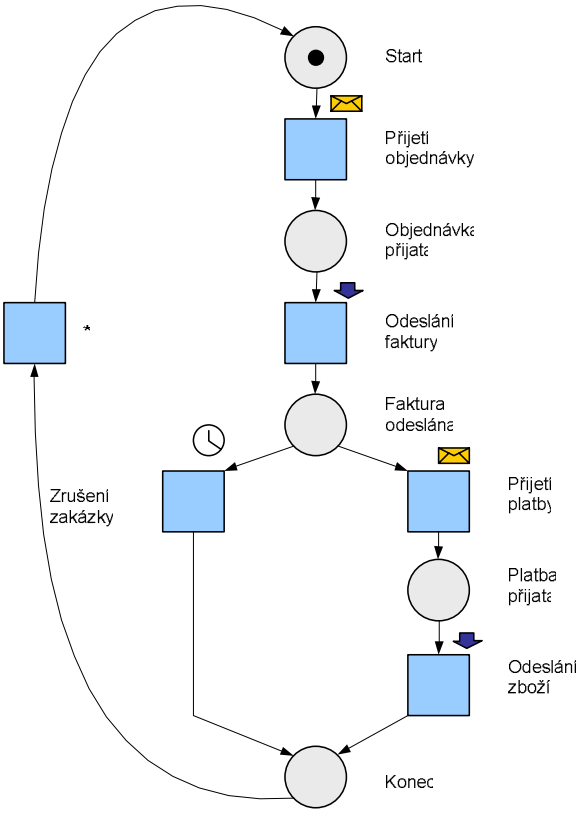
Ú 2.4	 «worker» Obchodní oddělení «worker» Útvar výroby 1 1 1 nakupuje zhotovuje je spotřebovávár * * * je požadovár «entity» Materiál «entity» Výrobek * 1
Ú 2.5	 Účetní Vedoucí Vystavení faktury [částka > 5000 Kč] [částka <= 5000 Kč] Autorizace Odeslání faktury

Ú 3.2/1	
Ú 3.2/2	
Ú 3.2/3	

Ú 3.3

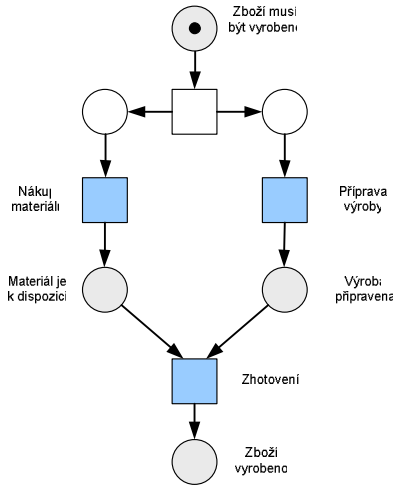


Ú 3.4



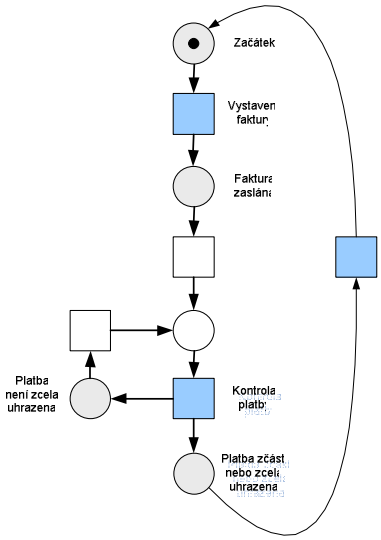
Nakrátko propojená WF-síť procesu zásilové služby je živá a omezená (bezpečná). Každý přechod lze provést obecně n-krát. Vzhledem k tomu, že WF-síť je živá a omezená, je také spolehlivá.

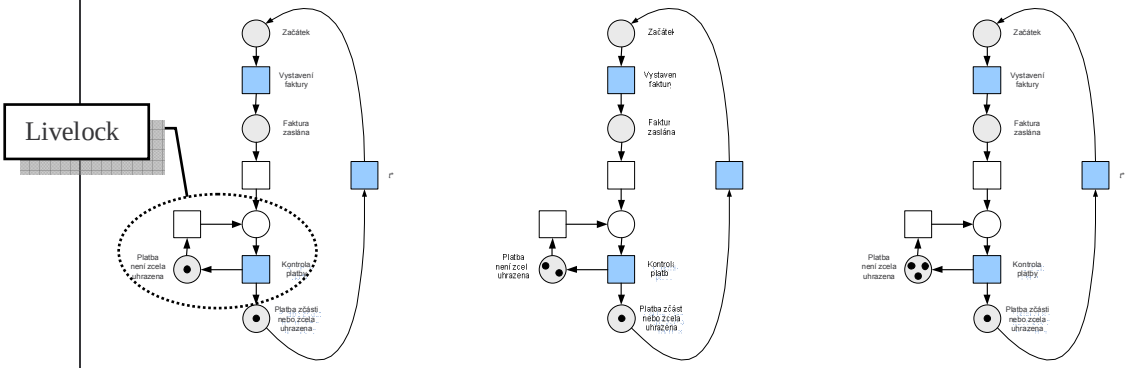
Ú 3.5/1



Petriho síť je WF-síť, protože splňuje všechny tři body z **definice 3.10** (*WF-síť*) a je zřejmě i spolehlivá, protože splňuje všechny požadavky z **definice 3.11** (*Spolehlivost*).

Ú 3.5/2





1.průchod 2. průchod 3. průchod

Nakrátko propojená Petriho síť musí být *živá* a *omezená*, aby odpovídající WF-síť byla *spolehlivá.* Poněvadž každý přechod může být proveden libovolně mnohokrát, je tato síť *živá.* Ze simulace několika průchodů však vyplývá, že dochází ke kumulaci značek na místu *Platba není zcela uhrazena.* Petriho síť tedy není *omezená* a tudíž ani proces není *spolehlivě* definovaný. Vzniklá situace se nazývá uvíznutí (*livelock*) v nekonečném cyklu .