Mendelova univerzita v Brně
Provozně ekonomická fakulta

Využití jazyka Object Constraint Language pro inovaci informačního systému

Diplomová práce

Vedoucí práce:
doc. Ing. Ivana Rábová, Ph.D.

Bc. Petr Slavíček

Brno 2016

Poděkování
Děkuji vedoucí práce doc. Ing. Ivaně Rábové, Ph.D. za metodické a cíleně orientované vedení při vypracování diplomové práce a také za cenné připomínky.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Využití jazyka Object Constraint Language pro inovaci informačního systému** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací*.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 19. května 2016 .....

Abstract

Bc. Slavíček, P. Using Object Constraint Language for inovation of information system. Diploma thesis. Brno: Mendel University, 2016.

Document proposes possibilities of using Object Constraint Language (OCL) in the process of inovation of information system. For inovation was choosen production plan of company Novibra Boskovice Ltd. In thesis is discussion about the type of production plan that company needs. Which data do they need to record and which do they need to gain from them. Production plan is elaborated into parts of analysis, proposal and also implementation of induvidual functional parts. In OCL part is introduced its short history, current state and also about possible future. What use has it, which alternatives exist, which programs use it for modelling, advantages and disadvantages of using it in this and other works.

Keywords: Thesis, diploma thesis, Object Constraint Language, information system, Java, UML, class diagram, business rules, production plan.

Abstrakt

Bc. Slavíček, P. Využití jazyka Object Constraint Language pro inovaci informačního systému. Diplomová práce. Brno: Mendelova univerzita v Brně, 2016.

Text se zabývá možnostmi využití Object Constraint Language (OCL) při inovaci informačního systému. Pro inovaci byl vybrán plán výroby společnosti Novibra Boskovice, s. r. o. V práci je pojednáno, jaký typ výrobního plánu společnost potřebuje. Jaké údaje v něm potřebuje evidovat a získávat z něj. Plán výroby je rozpracován do částí analýzy, návrhu a také implementace jednotlivých funkčních částí. U OCL je uvedena jeho krátká historie, současný stav a také to, jaká by mohla být jeho budoucnost. K čemu je OCL využíván, jaké jsou alternativy, které programy pro modelování jej využívají, výhody a nevýhody použití u této i jiných prací.

Klíčová slova: Závěrečná práce, diplomová práce, Object Constraint Language, informační systém, Java, UML, diagram tříd, podniková pravidla, plán výroby.

Obsah

Seznam obrázků

1 Úvod a cíl práce

1.1 Úvod

Při modelování informačních systémů a jejich částí návrhář dřív nebo později narazí na to, že samotné modely ostatním členům vývojového týmu nestačí. Naráží na skutečnosti, které do modelu nelze zařadit, ale je nutné na ně ostatní členy upozornit, aby s nimi v pozdějších fázích počítali. Část těchto nevyjádřitelných skutečností se týká omezení. Zjednodušeně by bylo možné omezení popsat jako soubor pravidel, která stanovují jaké prvky (hodnoty) do celé množiny prvků (do oboru hodnot) patří nebo naopak nepatří. Po zpracování omezení tak z této množiny možných hodnot část odstraníme a zůstává nám jen omezená část, jejíž hodnoty pro daný atribut můžeme použít.

Před vznikem Object Constraint Language existovala jediná možnost, jak tato omezení do návrhu systému (do modelů) dostat. A to bylo pomocí přirozeného jazyka. Tento způsob ale přinášel problémy. Tím prvním bylo to, že jedno jediné omezení bylo možné popsat mnoha způsoby s různou délkou textu. Přičemž všechny varianty mohly dosahovat naprosto stejné kvality, ale také nemusely. Tím druhým a zásadním problémem pak byla víceznačnost. Osoby, kterým tato omezení byla určena, je nemusely pochopit stejným způsobem jako jejich autor. Snadno se mohlo stát, že celý tým dané omezení pochopil. Až na jednoho, který mu porozuměl jinak. Vůbec to nemuselo být způsobeno jeho schopnostmi. Prostě se jen na vzniklý text podíval z jiného úhlu pohledu.

Pro snížení vlivu těchto problémů se postupně začaly používat již existující programovací jazyky. Z velké části odstraňovaly víceznačnost, ale byly to velké kusy kódu, které se už přímo mohly použít v části implementace. Nebo by se lehce upravily tak, aby se daly použít. Nebyly vytvořeny pro tvorbu omezujících pravidel, a proto jejich použití bylo v mnoha případech velice neefektivní. Bylo tedy nutné vytvořit něco zcela nového, co by bylo vytvořeny přímo k použití v UML diagramech pro popis omezení.

Vznikl tak Object Constraint Language (OCL), který je určen pro objektově orientované návrháře a analytiky pro modelování UML diagramů. OCL si vzal inspiraci z SQL a jazyka podobnému Javě (C++). Část podobnou k SQL využívá v operacích pro vyhledání prvků modelu. Důvodem proč se tvůrci inspirovali SQL, bude pravděpodobně v tom, že se k této činnosti hodí nejlépe. Programátoři jej znají a jeho kód není tak obsáhlý jako kódy řešící stejný problém zapsané v jiných jazycích. (Arlow a Neustadt, 2007)

Spousta programů pro medelování UML diagramů nenabízí pro OCL žádný specifický nástroj, ve kterém by byla zabudována kontrola správnosti (validita) OCL kódu. Většina řeší OCL pouze jako poznámky do diagramů. Bez kontroly správnosti sem uživatel může napsat cokoliv a nedozví se, že kód je chybný a další uživatelé jej tedy nemusí chápat. Jsou však ale výjimky, které validitu zvládnou a upozorní uživatele na chybu a případně i na to, jak ji odstranit (např. rozšíření Papyrus pro

program Eclipse). Tato rozšíření a programy teprve dávají návrhářovi opravdu silný nástroj, o který se mohou opřít, pokud chtějí sdělit dalším členům ve vývojářském řetězci omezení, které se daného modelu týkají. A nemusí se bát toho, že by kódy obsahovaly chyby. Ty úplně nejlepší nástroje pak mají možnost takto zkontrolovaný kód generovat do nějakého programovacího jazyka (většinou Javy).

Nevýhodou, kterou je potřeba si uvědomit, je to, že OCL není příliš rozšířený. Málo návrhářů jej ovládá a ještě méně jej používá. Což má za následek to, že nástrojů pracujících s OCL je velice málo. Nejsou často aktualizované a i vývoj samotného OCL je pomalý. Aktuální verze v době vzniku této práce je OCL 2.5, jenž byla vydaná v únoru roku 2014.

Nicméně v současnosti neexistuje žádný kvalitnější způsob, jak jiným způsobem omezení popsat v UML diagramech, než je OCL. Aby bylo možné zjistit výhody, nevýhody a problémy, které v praxi mohou nastat při použití OCL, bylo nutné vybrat nějaký praktický problém, který by se vyřešil od úplného začátku až téměř do konce. Jelikož jsem se při své praxi zabýval problémy související s plánem výroby, tak jsem se této oblasti chtěl věnovat i nadále a hlouběji.

Na praxi jsem měl na řešení omezený čas, a proto výsledek nemohl být dokonalý. Problém se mi sice podařilo vyřešit, ale už tehdy jsem věděl o lepším, ale časově náročnějším řešením, které tehdy uskutečnit nešlo. Z tohoto důvodu se zde chci k tomuto problému vrátit a vyřešit jej pomocí objektového programování v programovacím jazyce Java s použitím modelů, ve kterých využiji možnosti OCL.

1.2 Cíl a metodika práce

1.2.1 Cíl práce

Cílem této práce je použití specifického jazyka OCL pro vybraný problému z praxe. Bude vyvinuta aplikace pro podporu řízení výrobního plánu ve středně velké firmě v programovacím jazyce Java. Vývoj bude realizován podle vodopádového modelu, který bude členěn do fází analýzy problému, návrhu řešení, implementace a testování. V rámci analýzy bude diagram tříd UML rozšířen o integritní omezení zapsané pomocí OCL. Vyvíjená aplikace se bude týkat problému řízení výrobního plánu. Součástí práce je i zhodnocení výhod výsledné aplikace pro podnik, zhodnocení využití tohoto jazyka pro jednotlivé role vývoje softwaru a doporučení, zda je vhodné OCL u podobných projektů využívat.

1.2.2 Metodika práce

V první části práce se zaměřím na to, jak OCL vznikl, kdo stojí za jeho vznikem a k čemu se používá. Kdo se stará o jeho současný vývoj a kteří autoři se tématikou OCL zabývají. Jaké v současnosti existují nástroje na tvorbu jednotlivých integritních omezení v OCL, v čem se mezi sebou odlišují. Dále pak jestli a čím lze OCL nahradit. Součástí bude i krátký popis výrobního plánu. Jaká část informačních systémů se mu věnuje, jaké jsou jeho typické vlastnosti a v jakých aplikacích lze

s výrobním plánem pracovat. Ve zkratce se budu věnovat i programovacímu jazyku Java.

Další část bude o základní syntaxi OCL. Bude zde popsána metodika, kterou jsem zvolil pro vývoj aplikace. Ke každé z fází této metodiky bude stručně popsáno, čím se tato fáze zabývá, co je jejím cílem, jaké se v UML používají nástroje a jakým způsobem, co je pro ně typické, které z těchto nástrojů jsem využil a v jaké aplikaci.

V následující části popíši řešený problém. Vytvořím modely požadavků a vytvořím model případů užití. Ke každému případu užití následně vytvořím jeho specifikaci. Poté vytvořím analytický diagram tříd, případy užití rozepíši do sekvenčních diagramů a vyhledám veškerá omezení. Ty převedu do podoby OCL a přidám je do analytického diagramu. Následně vytvořím návrhový diagram tříd, podle kterého budu aplikaci implementovat. V části věnované implementaci uvedu, jakým způsobem byly OCL prvky převedeny do programovacího jazyka Java, a také zde budou rozebrány zajímavé vlastnosti a části výsledné aplikace.

V části věnované diskusi popíši výhody a nevýhody vzniklé aplikace a samotného používání OCL. V závěru zkontroluji splnění cíle, zhodnotím stav výsledné aplikace a také jak důležité bylo OCL pro její vývoj.

2 Přehled materiálů a základních informací k OCL a APS

2.1 OCL

2.1.1 Historie, současnost a možná budoucnost OCL

Object Constraint Language vznikl v roce 1995 ve firmě IBM jako „jazyk pro podnikové inženýrství“ („bussiness engineering language“). Za jeho autora je považován Jos Warmer. Warmer se při tvorbě OCL inspiroval v objektově orientované analýze a designové metodice Syntropy, který byl vytvořen Stevem Cookem, se kterým byl Warmer v kontaktu u IBM. Syntropy je modelovací technika, která dovoluje precizní specifikaci a oddělené oblasti zájmu. (Cook a Daniels, 1994)

Další verze (OCL v1.1) ale již nespadala pod IBM, ale ujalo se jí sdružení OMG, které jej uznalo jako formální specifikaci k UML v1. V té době UML neobsahovalo žádný nástroj pro precizní formulování pravidel, a tak sdružení sáhlo po této slibně vypadající novince. (Cabot a Gogolla, 2012) I dnes se o OCL stará toto sdružení a aktuální verze OCL je 2.4, která byla vydaná v únoru roku 2014. OMG ke každé nové verzi vydává na svých stránkách specifikaci, jak OCL používat (s názornými příklady). K němu vždy vydává i dokument s výčtem veškerých změn oproti starší verzi. Pod sdružení OMG samozřejmě spadá i samotné UML. Aktuální verze je 2.5 a byla vydaná v červnu 2015.

Ke konci letošního roku by měla vyjít nová verze s pořadovým označením 2.5. Měla by přinést importy, anonymní funkce, šablony, reflexi, typové konstruktory, vícenásobné návratové hodnoty, elseif a mnoho dalšího. (Willink, 2014)

V současnosti má OCL využití v UML při definici omezení, dobře definovatelných pravidel, derivačních pravidel, předpokladů, výstupních podmínek, transformaci modelů do jiných modelů a transformaci modelů do textu. (Cabot a Gogolla, 2012) Současná podoba byla významně ovlivněna setkáním významných autorů OCL během dvoudenního setkání v listopadu roku 1998 v Amsterodamu. (Cook et al., 2012) V něm byly vyřešeny problémy se sémantikou a syntaxí. Také se probíraly možná rozšíření OCL a celkově došlo k mnohým opravám původní verze jazyka. Po tomto setkání vznikl dokument, kde byly popsány veškeré dohodnuté změny, které byly doplněny o názorné příklady použití.

2.1.2 Další zdroje k OCL a UML

Jako jeden z hlavních zdrojů k informacím o OCL se dá považovat trojice knížek přímo od autora OCL. Jsou jimi „Object modeling with the OCL: The rationale behind the Object Constraint Language“ (Warmer a Clark, 2002), „The object constraint language: precise modeling with UML“ (Warmer a Kleppe, 1999) a „The object constraint language: getting your models ready for MDA“ (Warmer a Kleppe, 2003). Obsahují v sobě základní informace o syntaxi a sémantice OCL, návod na

používání v praxi, možnosti generování kódů do Javy, jak správně kombinovat OCL s UML a mnoho dalšího.

V českém jazyce vydaná kniha „UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky“ (Arlow a Neustadt, 2007) je pravděpodobně tím nejlepším dílem, který byl přeložen do češtiny, co se týče této problematiky. Z kostry tohoto díla jsem vycházel při zpracování této práce. Jsou v ní dopodrobna popsány veškeré fáze metodiky Unified Process, kterou jsem při vývoji této aplikace využil, a věnuje se i dopodrobna jednotlivým UML diagramům. Speciálně pak diagramu tříd, ke kterému v doplňkovém materiálu do detailu rozepisuje celou problematiku OCL. Jsou v ní popsány veškeré principy UML v2, praktické tipy a ukázky.

Dalším zdrojem je kniha „Enterprise modeling and computing with UML“ (Rittgen, 2007). Kniha se věnuje UML rozšířením pro modelování podniku. Část se věnuje pohledu na UML jako metajazyk. Popisuje použití UML a v jedné z částí se zabývá hodnotou UML modelů pro podnik. V jedné z částí se věnuje i OCL. Tato kniha je ve své podstatě sbírkou vědeckých prací různých autorů shrnutých do jedné celistvé práce popisující celý problém UML. Pohlíží na UML ve dvou oddělených pohledech. Modelování podniku a modelování informačních systémů.

V oblasti UML jsou známou postavou osoby Hanse-Erika Erikssona a Magnuse Penkera. A to především díky jejich diagramu, který nese jejich jméno. Jejich kniha „Business modeling with UML: business patterns at work“ se zaměřuje i na jiné typy diagramů a v souvislosti s diagramem tříd zmiňují i OCL. O něco mladší je pak kniha „UML 2 toolkit“ (Eriksson et al., 2004). Jde o příručku k UML a je v ní i krátká zmínka k OCL.

Veškeré zde uvedené zdroje starší roku 2014 je však nutné brát s rezervou ohledně OCL a UML, protože nezmiňují novinky v OCL v2.4 a UML v2.5, a tak v nich mohou být zastaralé informace. Nicméně jsou velice dobré pro základní pochopení problematiky OCL, která se s novými verzemi prakticky nemění. Dokumentace na stránkách OMG je sice aktuální, ale pro pochopení nemusí být dostatečná na rozdíl od těchto starších knih, které odkazují na starší verze OCL a UML. Z tohoto důvodu jsem k problematice čerpal z obou typů zdrojů.

Zdrojů k UML je obrovské množství, ale bohužel ty kvalitnější knihy nejsou obvykle aktualizovány tak často jako samotné UML. Málokterá kniha k UML se věnuje OCL kvalitně a dává mu větší prostor. Výjimkou je „UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky“ (Arlow a Neustadt, 2007), která OCL věnuje opravdu velkou část knihy. Je tak prakticky jediným zdrojem, který kvalitně obsáhne obě problematiky. U ostatních zdrojů je vždy nutné zvolit nějakou kombinaci.

2.1.3 Aplikace a nástroje pracující s OCL

Většina OCL nástrojů má nějakou spojitost s Javou. Buď jsou v ní napsaná, umí generovat kód do Javy nebo jsou rozšířeními do programů pro vývoj Java aplikací (jako jsou Eclipse či NetBeans). Mezi nejznámější nástroje patří:

- Dresden OCL,
- Eclipse MDT/OCL,
- HOL-OCL,
- Oclarity,
- OCLE,
- SimpleOCL,
- UMLtoCSP,
- Visual OCL,
- Octopus
- a několik dalších.

Problémem většiny těchto nástrojů je, že je tvoří jednotlivci a nejsou často aktualizované. Většina z nich vznikla již před rokem 2010, a tak aktuální verzi UML a OCL obsahuje málokterá. Ty lepší obsahují generování kódu do programovacího jazyka a to jak ze vzniklých UML diagramů (vytvoření tříd, jejich atributů, hlaviček metod), tak i samotných OCL. Bohužel se ale nedá říci, že jde o až tak velkou výhodu, protože jde většinou jen o generování kódu do programovacího jazyka Java. Lze ale nalézt i rozšíření, která generování do jiných kódů umožní. Například do rozšíření Eclipse s názvem Papyrus lze přidat další rozšíření, které generuje kód do C++.

V této práci jsem využil nástroj Papyrus, který se přidává jako rozšíření do Eclipse. Výhodou tohoto a jemu podobných nástrojů je, že OCL je propojené s UML diagramem, a tak zde existuje kontrola správnosti nejen podle syntaxe, ale i podle sémantiky. Pokud uživatel udělá v OCL chybu (např. odkáže na neexistující atribut dané třídy), tak je na ni upozorněn a je mu nabídnuto i vyřešení problému. U nástroje Papyrus jsem vyzkoušel nabízené generování kódu, ale nebyl jsem spokojen s výsledným kódem. Kód byl pro mě velice nepřehledný. Byl rozházený do různých tříd a balíčků a přišel mi i hodně obecný. Pro jeho využití bych jej musel upravit, abych se v něm trochu vyznal. Z těchto důvodů jsem se rozhodl tento vygenerovaný kód nepoužít a převod OCL prvků do Javy udělat úplně celý ručně. Jen tak získám kód, který pro mě bude přehledný, a navíc mohu narazit na problémy, které s tímto převodem souvisí.

2.1.4 Alternativy k OCL

K OCL existují alternativy. Ale žádná se OCL plně nevyrovná. Mezi ně patří:

- přirozený jazyk,
- First-Order Logic (FOL),
- Alloy,
- SQL,
- doménově specifické jazyky (DSL),
- Semantics of Business Vocabulary and Rules (SBVR),
- Z, objektový Z,
- Vienna Development Method (VDM).

Přirozený jazyk člověku nejvíce vyhovuje. Jeho problémem ale je, že mu počítače zatím dostatečně nerozumí a je víceznačný. Při přenosu mezi dvěma subjekty (zdrojem a cílem), nemusí mít na konci přenosu oba stejnou informaci, protože přenášená data přirozeného jazyka cíl pochopil jinak (měla pro něj jiný význam). Alloy a Z jsou postaveny na logice a teorii množin. Z má bohatší matematickou notaci a širší využití než Alloy. VDM je množinou technik pro vývoj počítačových systémů a je vůbec jednou z prvních formálních metod. Avšak ještě dnes se používá a to zejména v průmyslovém odvětví. DSL jsou jazyky určené primárně ke komunikaci. Nejsou obecné, ale zaměřují se na konkrétní problémovou doménu (jako např. awk). SBVR je standard sdružení OMG, který definuje slovník a pravidla pro popis sémantiky faktů a pravidel. FOL je v Česku znám spíše pod označením predikátová logika. SQL je strukturovaným dotazovacím jazykem v relačních databázích.

2.2 APS

2.2.1 Co je to APS

Systémy výrobního plánu jsou ve světě známy pod zkratkou APS („advanced planning and scheduling“ nebo také „advanced planning system“). Zabývají se plánováním nákupu materiálu nutného k výrobě produktů (polotovarů), plány výrobní haly a plánováním samotné výroby. Z této definice vyplývá, že je výrobní plán pouze podmnožinou APS. Ale dalo by se říci, že je tou nejdůležitější částí. Moderní APS dokáží optimalizovat výrobu podniku tak, aby byl vyroben produkt v požadované kvalitě a včas podle obdržených zakázek s optimálním využitím výrobních kapacit. APS se obvykle dělí do těchto částí:

- nákup materiálu,
- obchodní (prodejní),

- výrobní (plán výroby),
- managerská (řídící).

Plánování výroby je jednou z částí produkčního cyklu. Jde o přetváření zdrojů (materiálu, lidské práce a energie) do výrobků. Součástí hodnoty vyrobených výrobků se stává i část strojů, na kterých výrobky vznikají (hodnota opotřebení strojů).

Ve své práci jsem se zaměřil na část výrobní (výrobní plán). Ta ve větších a kvalitních systémech bere informace i z dalších částí (materiálové a obchodní) a vybírá výrobu takovou, aby materiál, ze kterého se produkt vyrábí, byl na skladě a zároveň pro něj existoval zákazník. Jelikož účelem této práce nebyla tvorba částí s nákupem materiálu a prodejem výrobků, nechávám výrobu na rozhodnutí samotných zaměstnanců. Sami si určí, co se bude vyrábět a v jakém pořadí. U podniků zabývajících se zakázkovou výrobou by to mohlo být pro zaměstnance příliš náročné, ale u výroby na sklad v sériové výrobě to až takovým problémem být nemusí. Plán výroby také může poskytovat přehled pracovišť a jednotlivých strojů. Kolik se má čeho vyrábět a kdy. Často je obsažena i informace o tom, jak by vypadala celková výroba, kdyby se přidaly další směny, přidaly stroje, zvýšila kapacita současných strojů, atd. Zvládá reagovat na změny, které proběhnou. Například se zruší některá ze zakázek nebo se některé zvýší priorita. Umí nalézt úzká místa výroby a dokáže je optimalizovat.

Výrobním plánem se zabývá spousta autorů. Ať už samostatně, v rámci produkčního cyklu nebo výroby. Z publikací mohu zmínit například „Manufacturing planning and control systems for supply chain management“ (Vollmann, 2005), „Factory planning manual: situation-driven production facility planning“ (Schenk, Wirth a Muller, 2010) a „Planning, scheduling and constraint satisfaction: from theory to practice“ (Castillo, 2005).

2.2.2 Aplikace pro výrobní plán

Podle SystemOnLine.cz existuje přibližně 45 APS dostupných v Česku. Novibra Boskovice využívá informační systém SAP, ve kterém existuje modul pro „plánování výroby a detailní rozvrhování“. Je znám pod zkratkou SAP APO (SAP Advanced Planner and Optimizer). Umí kontrolovat stav materiálu a přidávat dávky do plánu ihned po přijetí zakázky. Součástí je také výrobní optimalizace. Ale schází mu ta nejdůležitější vlastnost, která je tímto podnikem vyžadována. A tou je výpočet koeficientů výroby, který se kvůli tomu musí nějak počítat ručně. K tomuto účelu používá podnik aplikaci Microsoft Excel. I při použití Excelu ale SAP APO obsahuje ještě jeden problém. A to je přehlednost (doklikání se k potřebným informacím). I z tohoto důvodu je pro analýzu výroby v tomto podniku používán především dokument Excel. Veškerá data potřebná k výpočtu jsou viditelná na jedné obrazovce. Bez nějakého složitého hledání.

Tyto problémy postihují i jiné systémy a aplikace. Proto následující seznam systémů a aplikací pro výrobní plány neberte jako alternativy pro řešení tohoto

problému, protože jimi jednoduše nejsou. Mohou být použity v podnicích, kde tyto vlastnosti nejsou vyžadovány. Mezi tyto systémy a aplikace patří především:

- Excel,
- SAP APO,
- APS ONE,
- MPlan,
- Microsoft Dynamics NAV, AX,
- BYZNYS ERP - Výroba,
- K2,
- Plantune,
- QI.

Podnik může i nadále používat kombinaci SAP APO a Excelu. Nepřineslo by to žádné další náklady oproti jiným zmíněným systémů a aplikacím. SAP APO ale nemá potřebný výstup a Excel není kvalitním řešením. U ostatních jsou problémem jak cena, tak výstup. Výstup ale lze u některých aplikací upravit. Příkladem může být QI a jeho QI Builder. V něm by bylo možné informačnímu systému přidat funkcionalitu s výpočtem koeficientů a tudíž by splňoval základní požadavek.

2.3 Java

Výsledný program této práce byl naprogramovaný v jazyce Java. Výhodou programovacího jazyku Java je jeho nezávislost na architektuře. Výsledný program tak lze spouštět na různých operačních systémech. Oproti C nebo C++ má trochu zjednodušenou syntaxi. Pro mě je přehlednější, lépe se mi s ním pracuje než s ostatními jazyky a mám v něm více zkušeností. Nevýhodou je naopak to, že je jeho spuštění pomalejší. Což ale pro vznikající aplikaci věnované výrobnímu plánu není takovým problémem. Požadavky na aplikaci odpovídají tomu, co Java zvládne. Ať už jde o desktopovou aplikaci, přístup ke vzdáleným souborům, jejich synchronizaci pro přístup více uživatelů, autorizaci i autentifikaci.

K Javě existuje mnoho publikací, ze kterých se dá čerpat. Zmíním zde alespoň dvě. Tou první je „Java 7: výukový kurz“ (Schildt, 2012). Jde o do češtiny přeloženou příručku popisující vše podstatné okolo Javy s názornými příklady. Často se aktualizuje a v době psaní této práce vychází v češtině již 8. verze této oblíbené knihy. Tou druhou je pak opět kniha vydána Herbertem Schildtem s názvem „Java the complete reference“ (Schildt, 2011), která je v podstatě rozšířenou verzí předchozí knihy s dvojnásobným počtem stran, ale bohužel není překládána do češtiny.

2.4 Ostatní zdroje

Kniha „Softwarové inženýrství“ (Sommerville, 2013) se, jak již název napovídá, zabývá všemi důležitými znalostmi do softwarového inženýrství. Celým procesem softwarového inženýrství, agilními metodami, vývojem systémů, modelování pomocí UML, architekturou a designem. Je určena pro profesionály z této oblasti, ale svojí hodnotu má i pro ostatní. Druhou důležitou knihou je „Procesně řízená organizace“ (Řepa, 2012), která se zaměřuje na procesy organizace ve vztahu k informačnímu systému organizace, modelování informací organizace, jak zavést procesní řízení do organizace, organizace podniku a mnoho dalšího.

3 Syntaxe OCL a metodika vývoje software

3.1 Syntaxe OCL

OCL je jazyk, který upřesňuje modely o další informace. Je standardem, který v současnosti spravuje sdružení OMG. Využívá se pro definování omezení a vyhledávacích funkcí. U většiny aplikací určených k modelování se přidává jako prostý text do objektů typu poznámka. Má složitou syntaxi, která připomíná mix SQL a Javy nebo C++. Celá problematika OCL je docela rozsáhlá, takže zde zmíním pouze základní syntaxi. Na stránkách OMG je ke stažení manuál k současné verzi OCL a jeho použití o 264 stranách.

První částí OCL je kontext balíčku. Pokud třídy, na které se chceme odkazovat, jsou uvnitř nějakého balíčku, tak je potřeba tuto skutečnost zmínit pomocí následujícího kódu. To samé platí, pokud jen chceme odkazovat na atributy nějakého objektu třídy nebo její funkce. Jde o nepovinný kontext. Pokud tedy třídy v žádném balíčku nejsou, tak se nepoužije. Zbytek OCL kódu se vyskytuje mezi oběma klíčovými slovy.

```
package NazevBalicku
...
endpackage
```

První povinnou částí OCL je pak kontext výrazu. V něm sdělujeme, které třídy se dané omezení týká. Pokud jde o omezení nějakého atributu třídy, necháváme pouze název třídy. Pokud jde o omezení nebo vyhledávání akci spadající pod konkrétní funkci, zapisujeme k názvu třídy také název funkce se všemi názvy parametrů a jejich typem společně s návratovým typem této funkce. Tak jako je to znázorněno na následujícím příkladě.

```
package NazevBalicku
context Trida::nejakaFunkce(par1:Integer):boolean
...
endpackage
```

Následuje typ výrazu, který bude použit. Existují následující typy:

- inv: pro vždy platná omezení,
- pre: pro podmínku, která musí platit před OCL výrazem,
- post: pro podmínku, která musí platit po OCL výrazu,
- body: po kterém následuje vyhledávací funkce,
- init: pro inicializaci hodnoty,
- derive: pro odvozovací pravidla,

- let: pro definici dočasné proměnné, která může být použita ve zbylém kódu OCL,
- def: pro definici trvalé proměnné, která může být použita i u jiných OCL.

Po uvedení typu může za mezerou následovat název OCL výrazu. Poté je uvedena vždy dvojtečka, která je následována samotným kódem.

```
context Trida::nejakaFunkce(par1:Integer):boolean
inv nazevVyrazu: ...
```

V OCL se používá zástupné slovo *self*. Jím se nahrazuje libovolná instance třídy uvedené v kontextu. Přes tečku se z ní přistupuje k jednotlivým hodnotám atributů instance nebo výsledkům funkcí. Pomocí názvu vazby pak i k instancím jiné třídy, které jsou ve vazbě s touto třídou, a k jejich hodnotám atributů a výsledkům funkcí. Atributy pak lze jednoduše omezovat pomocí výrazů. Například následující kód upravuje, aby žádná osoba nemohla mít nulovou nebo zápornou výšku. A to nikdy za běhu programu.

```
context Osoba
inv vyskaOsoby: self.vyska >= 1
```

V případě funkcí pak lze pomocí klíčového slova *result* zvolit jaký výsledek funkce bude vracet. OCL obsahuje spoustu klíčových slov k funkcím, které pracují se samotnými atributy. Například funkce *sum*() sečte veškeré hodnoty dodané kolekce. Následující příklad tak sečte veškeré hodnoty příjmů (pole, ve kterém jsou uloženy veškeré příjmy na účtu) a od nich odečte veškeré výdaje. Výsledek tohoto rozdílu je pak návratovou hodnotou funkce, která zjišťuje stav účtu.

```
context Ucet::stavUctu():Integer
post: result = self.prijmy->sum() - self.vydaje->sum()
```

Někdy je potřeba aktualizovat hodnotu atributu. V takovém případě je nutné v OCL říci, že chceme předchozí stav měněného atributu. Toho dosáhneme pomocí klíčového slova @*pre*, které se uvede za název atributu.

```
context Terc::zasahTerce()
post: pocetZasahu = pocetZasahu@pre + 1
```

Následující OCL kód zjišťuje, zda jsou v podniku všichni zaměstnanci do 40 let. Třída Oddělení má vazbu na třídu Osoba s názvem zaměstnanec. Po zavolání funkce, která zjišťuje, jestli je kolektiv pracovníků daného oddělení mladý, se projdou veškeří zaměstnanci a u každého se zkontroluje jeho věk. Pokud ji jediný zaměstnanec poruší (je starší), tak je jako výsledek funkce vrácena nepravda.

```
context Oddeleni::jeMladyKolektiv():boolean
post: result = self.zamestnanec->forAll( vek <= 40 )
```

OCL není záležitostí pouze diagramů tříd, ale dá se použít i u diagramů interakce, aktivit a stavových automatů.

3.2 Metodika vývoje aplikace a její fáze

Pro vývoj aplikace jsem zvolil metodiku UP (Unified Process). Ta je založena na dvou metodách. Ericsson approach a Rational objectory process. Tvůrcem této metodiky je Ivar Jacobson. Jacobson je duchovním otcem i metodiky RUP. Rozdílem těchto metodik je to, že RUP je produktem firmy Rational, která dnes spadá pod firmu IBM, zatímco UP je otevřeným standardem. Metodika UP pracuje v iteracích, kdy se při každé iteraci věnuje jen malé části z celého systému. V každé iteraci existuje celkem pět fází (pracovních postupů). Jsou jimi:

- požadavky,
- analýza,
- návrh,
- implementace,
- testování.

Samotný projekt tvorby celého systému má pak 4 fáze. Mezi ně patří:

- zahájení (plánování projektu),
- rozpracování (architektura projektu),
- konstrukce (provozuschopnost),
- zavedení (nasazení).

Přechody mezi jednotlivými fázemi tvorby projektu jsou milníky. Milníky označují určité cíle, kterých musí být dosaženo, aby bylo možné považovat předcházející fázi a milník za dokončený. Milníky jsou:

- záměry životního cyklu,
- architektura životního cyklu,
- funkční varianta produktu,
- nasazení produktu.

Dosažením posledního milníku je procesu vývoje aplikace (systému) fakticky ukončen. (Arlow a Neustadt, 2007)

3.2.1 Požadavky

Jsou prvním pracovním postupem při vývoji systému (aplikace). V této fázi se vytváří softwarové požadavky (metamodel). Metamodel je složen ze dvou částí. Z modelu požadavků a modelu případů užití. Model požadavků je možné dělit na model funkčních požadavků a model nefunkčních požadavků. Funkční požadavky je dále možné dělit na vlastnosti systému, podniková pravidla a uživatelské rozhraní. Funkční požadavky představují, co bude systém umět (jaké funkce bude nabízet), zatímco nefunkční požadavky představují, jaká omezení bude systém mít (např. maximální množství uživatelů, rychlost zpracování, atd.). Požadavky lze získat z konzultací s budoucími uživateli systému nebo jiných zainteresovaných osob. Dále pak z jiných systémů (softwaru) nebo hardwaru, se kterými bude systém v kontaktu. Z právních, regulačních nebo interních předpisů a také z obchodní strategie podniku. Při tvorbě požadavků se vychází i z dokumentu se základním popisem problému, ve kterém je ve zkratce uvedeno, co má systém umět a jaký je jeho přínos pro společnost. (Arlow a Neustadt, 2007)

Ve své práci jsem využil především konzultací s uživateli systému a další informace jsem získával z rozboru současného softwaru, který je používán na řešení tohoto problému (dokumentu Excel). Modely požadavků jsem vytvořil v programu Enterprise Architect.

Další důležitou součástí počáteční fáze vývoje systému jsou modely případů užití (modely use case). Ty jsou tvořeny aktéry a případy užití. Aktéři jsou role, které entity vůči systému představují. Případy užití jsou posloupnosti činností, které spouští aktor. Pro vytvoření modelu případů užití je nutné nalézt aktory, případy užití a vzájemné vztahy mezi nimi. Podkladem pro hledání jsou modely požadavků, informace od zainteresovaných osob, ze softwaru či hardwaru. K vytvoření modelu případů užití jsem použil program Enterprise Architect.

Jelikož je každý případ užití tvořen několika po sobě jdoucími činnostmi (kroky), vytváří se specifikace případu užití. Někdy se jim také říká scénáře. Většinou se graficky vyjadřují jako tabulka, která se skládá z několika částí. Názvu případu užití, identifikátoru, stručného popisu, co daný případ užití vykonává, seznamu aktérů (s rozdělením na toho, kdo jej spouští a koho dalšího mohou ovlivnit), vstupních podmínek, posloupnosti kroků, kterými případ užití prochází (s možností větvení, cyklů, skoků), výstupních podmínek a alternativních případů užití. Veškeré vstupní podmínky musí být splněny před samotným spuštěním případu užití, jinak jej nelze spustit. Výstupní pak musí být splněny po uskutečnění posledního kroku, ať už se prochází kteroukoliv větví. Modelování případů užití se nedoporučuje vždy. Například pokud je jen jeden typ uživatelů (jedna role, kterou uživatel může mít). Nebo pokud převládají nefunkční požadavky a těch funkčních je příliš málo (tudíž by i případů užití bylo málo).

Aktory je možné generalizovat. Podobně jako je tomu u programovacích jazyků při generalizaci tříd. Generalizovaná třída je zobecněním svých potomků. Každý z potomků je tak možné označit jako předka, pokud si odmyslíme vlastnosti,

které předek nemá. Stejným způsobem existuje i generalizace pro samotné případy užití. Mezi případy užití existují dva typy vazeb. Zahrnující (include) a rozšiřující (exclude). U zahrnující vazby se po zavolání zdrojového případu užití vždy zavolá i cílový případ užití. U rozšiřující je pak po zavolání cílového případu užití možné zavolat (za určitých podmínek) zdrojový případ užití. (Arlow a Neustadt, 2007)

Scénáře jsem vytvořil v Microsoftu Word. Enterprise Architect sice nabízí také možnost vytvářet scénáře, ale preferuji tento způsob, který je snadnější na tvorbu, je přehledný a zároveň snadno modifikovatelný pro druhé osoby, které s Enterprise Architect nemají zkušenosti. Pro export tabulek z Wordu do obrázků se dá využít doplněk s názevm „Kutools for Word“.

3.2.2 Analýza

Po sestavení požadavků na systém, případů užití a jejich specifikací následuje analýza. V té se realizují případy užití a tvoří analytický diagram tříd. Analytický diagram tříd je složen ze tříd a vazeb mezi nimi. Třídy se skládají z názvu, atributů a funkcí. V některých programovacích jazycích se rozlišují funkce a procedury, kdy funkce mají návratovou hodnotu, zatímco procedury ji nemají (případně vrací návratový typ volání). Třídy slouží jako předpis pro objekty (instance třídy). Jednotlivé instance si mezi sebou mohou předávat zprávy pomocí funkcí a měnit stavy svých atributů. Atributy mají tři typy viditelnosti. Soukromý (značící se $-$), veřejný ($+$) a chráněný ($\#$). V Javě existuje ještě čtvrtý typ, který některé jazyky nepoužívají, a nazývá se implicitní. V UML i na tuto viditelnost mysleli a označují ji za viditelnost balíčku ($\sim$), která je čtvrtým typem viditelnosti. Atributy jsou určitého datového typu. V UML existuje 5 základních datových typů. Pro celá čísla (Integer), přirozená čísla (UnlimitedNatural), řetězce (String), pravdivostní hodnoty (Boolean) a reálná čísla (Real). Je však možné tyto typy nepoužít a místo nich psát datové typy jazyka, který se použije pro implementaci, a vytvořit tak model závislý na konkrétním programovacím jazyce. Pro diagramy tříd jsem tedy zvolil datové typy, které se používají v Javě. Komunikace mezi jednotlivými objekty je prováděna pomocí relací. V diagramu tříd existují relace typu asociace, agregace a kompozice, které se mezi sebou odlišují sílou spojení. Čtvrtou často používanou vazbou mezi třídami je dědičnost (generalizace, zobecnění) a má podobnou funkci jako v případě aktérů. Potomek třídy získává veškeré atributy a funkce předka. Zároveň může obsahovat další atributy a funkce, které předek nemá, nebo některé funkce překrýt, aby konaly jinou akci než u předka.

Realizace případů užití se provádí pomocí diagramů interakce. Ty znázorňují spolupráci jednotlivých objektů k dosažení určitého chování. Typicky se zobrazuje jeden případ užití do jednoho diagramu interakce. Diagramy interakce se dělí na sekvenční diagramy, komunikační diagramy, diagramy zjednodušené interakce a diagramy časování. Sekvenční diagramy a analytické diagramy tříd vychází ze stejných zdrojů informací. Takže nezáleží jestli se po sestavení případů užití začnou tvořit sekvenční diagramy a teprve poté analytické diagramy nebo naopak. Údaje v nich

obsažené jsou však vzájemně úzce propojené. Funkce u konkrétní čáry života v sekvenčním diagramu musí být uvedena i v analytickém diagramu tříd u stejné třídy, která odpovídá této čáře života.

V analytické části existuje spousta dalších diagramů, které ještě více upřesňují návrhovou podobu projektu. Pro tuto práci jsem ale zvolil pouze analytický diagram tříd a sekvenční diagramy. Diagram tříd jsem namodeloval v programu Visual Paradigm. Výhodou tvorby diagramu tříd pod Visual Paradigm je možnost si vygenerovat Java kód dané struktury. Nestane se tak, že by programátor zapomněl na některou funkci. Pro sekvenční diagramy jsem využil Enterprise Architect.

Teprve po namodelování analytického diagramu tříd a sekvenčních diagramů je možné začít s tvorbou OCL kódů, které zpřesňují vztahy uvnitř analytického diagramu tříd. Ty jsem tvořil ve dvou verzích. Ta první byla pomocí rozšíření Papyrus pod programem Eclipse. Rozšíření Papyrus obsahuje nástroje pro medelování UML diagramů. Zvládá modelování diagramu tříd společně s OCL prvky. Výhodou tvorby v Papyru je možnost generovat Java kód ze vzniklého diagramu tříd společně s namodelovanými OCL prvky. Tuto možnost jsem vyzkoušel, ale vygenerovaný kód pro mě nebyl vyhovující. Abych se v něm vyznal natolik, abych mohl naprogramovat i zbytek aplikace, musel bych udělat příliš mnoho úprav. Proto jsem raději celý tento proces provedl ručně. Naopak nevýhodou pro modelování OCL v Papyru je to, že OCL kódy neobsahují všechny části. Kontext je nahrazen čarou k určité třídě, což není až takový problém (spíše naopak). Hlavním problémem to, že v diagramu nejsou viditelná klíčová slova body, precondition, postcondition, invariant, atd. Schází mi zde také možnost krátkého slovního popisu pravidla a vadí mi povinnost definovat název. Z tohoto důvodu jsem do názvů OCL zapisoval krátké poznámky o tom, co tento kód znamená. Celý diagram je tak srozumitelnější i pro ty, kteří s OCL nemají tolik zkušeností. Druhou verzi OCL jsem vytvořil jednoduše v textovém editoru tak, jak mají správně vypadat.

3.2.3 Návrh

V návrhové části se vytváří především návrhový diagram tříd. Ten je oproti analytickému zaměřen na naprogramovatelnost problému. Funkce a atributy, které byly potřeba pro základní popis problému, se rozšiřují o další funkce a atributy (nebo i rovnou celé předělávají), aby aplikace po naprogramování v identické struktuře mohla být funkční. Typické pro návrhový diagram tříd je, že se zde objevují veškeré funkce upravující přístup k soukromým atributům (get, set metody). Analytický diagram tříd jsem vytvořil v programu Visual Paradigm.

3.2.4 Implementace a testování

Pro implementaci jsem vybral programovací jazyk Java s využitím programu Eclipse. Při implementaci jsem vycházel ze všech výše zmíněných diagramů a modelů.

Pro převod OCL do programovacího jazyka se dá využít různých struktur. Nejčastěji jde o naprogramování do bloku try-catch-finally. Vyzkouší se kus kódu

a pokud neprojde, tak se vyhodí výjimka, které na porušení OCL upozorní a změnu v datech neprovede. Programovací jazyky bez tohoto bloku mohou využít jednoduché testování pomocí podmínek, kdy se v jedné z větví změna provede a ve druhé ošetří porušení omezení. V precondition OCL stačí před provedením změn vyhledat, co je nutné, a pokud není nalezen problém, tak je možné provést akci. U postcondition OCL je nutné vyzkoušet, co se provedením této akce stane, a teprve pokud nedojde k porušení omezení, tak tuto akci provést. U invariant OCL by se správně mělo kontrolovat obojí, ale je možné tento problém zjednodušit na typ postcondition. A to tak, že uděláme kontrolu po veškerých akcích, které by porušení těchto omezení mohly vyvolat. Pokud tedy začínáme s čistou aplikací bez dat (nebo s již dříve takto zkontrolovanými daty), k porušení již nikdy nedojde. Pokud by data dříve zkontrolována nebyla, musela by se před importem nějakou funkcí překontrolovat.

4 Vývoj aplikace

4.1 Požadavky

4.1.1 Aktuální situace a základní popis problému

Na praxi jsem byl v podniku Novibra Boskovice, s. r. o. Podnik Novibra původně pochází ze Stuttgartu, kde byl založen v roce 1920. Teprve až roku 1992 byl přestěhován do Boskovic. V roce 2001 se pak stal členem Rieter Group. Jednotliví členové této skupiny se zabývají stroji a příslušenstvím na spřádání příze. Novibra se specializuje na výrobu vřeten. Její vřetena jsou ve světě dopřádání známá pro svoji kvalitu. Je možné je používat za vysokých otáček a zároveň mají i delší dobu životnosti oproti konkurenci.

V Novibře používají informační systém SAP, ale pro plánování výroby využívají navíc dokumenty Microsoft Excel, ve kterém počítají týdenní koeficienty výroby jednotlivých strojů a celé výroby podniku, což v samotném SAP není umožněno. Tyto výpočty původně prováděli ručně, což pro přiděleného pracovníka prakticky znamenalo ztrátu téměř jednoho dne v týdnu, kdy by se mohl věnovat jiným věcem. Na praxi v tomto podniku jsem byl požádán, jestli by nebylo možné práci pro tohoto zaměstnance nějakým způsobem zjednodušit. Struktura dokumentů naštěstí byla vždy stejná. Lišit se mohl počet sloupců, který znázorňoval počet směn (respektive počet týdnů). První sloupec s daty byla vždy pondělní ranní směna a data vždy končila sloupcem představující nedělní noční směnu. V dokumentech se mohl lišit i počet strojů. Podnik mohl během období zavádět do výroby nové stroje nebo je vyřazovat. Problémem byla i samotná data. Musel jsem brát v úvahu nevyplněná pole, textové poznámky místo číselných hodnot a také se musela z dat zjišťovat výroba jednotlivých směn, jelikož se data týkala jen konečného stavu skladu po uskutečnění směny. K těmto datům pak bylo nutné vytvořit další tabulku s materiálem, do které se ukládala data, která v tabulce s výrobou nebyla, a tak je pro výpočet musel zaměstnanec ručně vyhledávat v SAP. Pomocí několika na sebe navazujících funkcí v Excelu se mi podařilo výpočet koeficientů zjednodušit natolik, že se po uživatelovi chtělo jen, aby nahrál do dokumentu data o výrobě na nový list, zadal do formuláře název tohoto listu a číslo týdne, pro který chce hodnoty vypočítat, a udržoval tabulku s materiálem aktuální. Pokud nastal problém ve výpočtu, tak dostal informaci, proč tato chyba nastala (a jak ji má opravit). Například o neznámém materiálu, které musí vyhledat ručně v SAP přidat do tabulky s materiálem, nebo že se v datech nachází poznámka, kterou musí převést do číselné podoby (nebo alespoň smazat).

Výpočet týdenního koeficientu stroje má následující podobu, kde tvs je týdenní výrobou stroje a sdd je sumou dílců všech dávek vyráběných tento týden:

$$tks = \frac{tvs}{sdd}$$

Toto řešení problému bylo dostačující, ale záviselo na tom, že se nikdy nezmění struktura. Pokud by se nějak změnila, automatické počítání koeficientů by přestalo

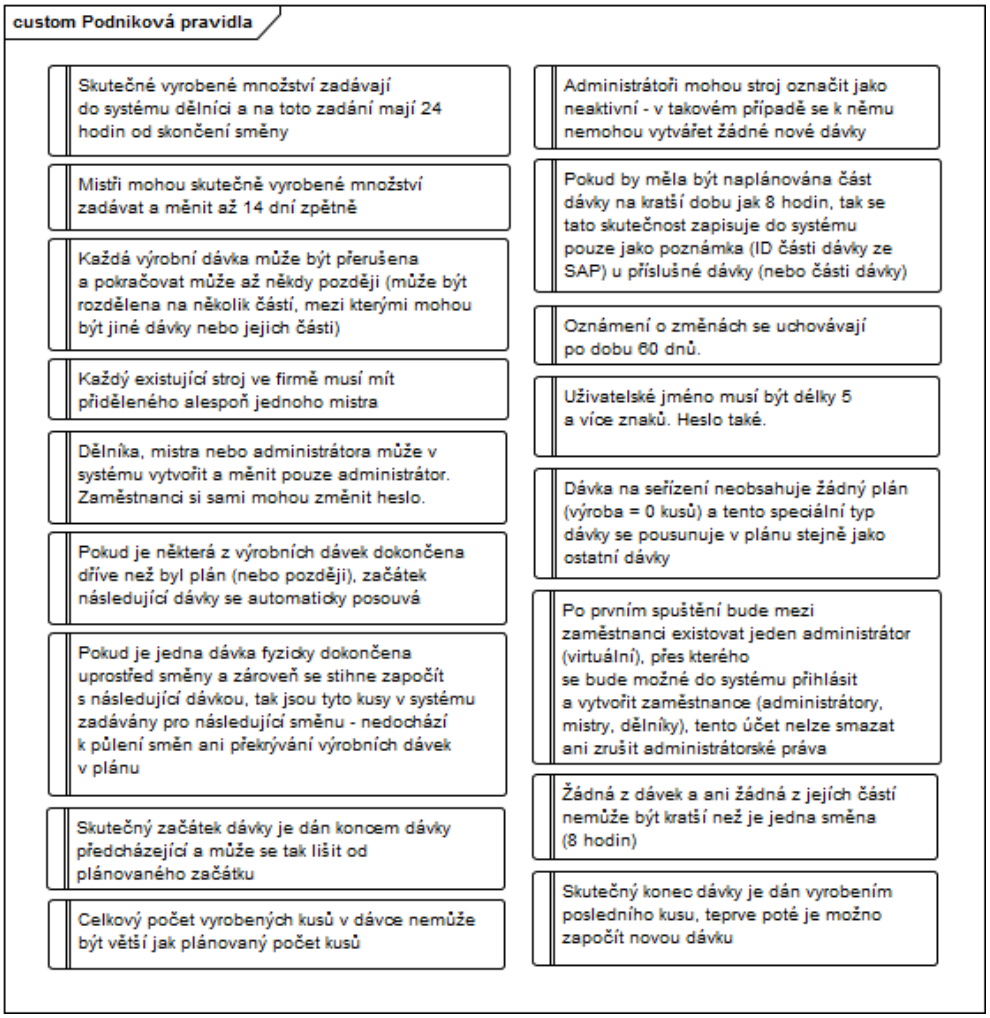
fungovat správně, dokud by se neupravily vzorce na tuto novou strukturu. To mě přivedlo na myšlenku převést tento problém do objektové podoby a vyzkoušet na něm možnosti využití OCL. V Novibře využívají síťový disk, na kterém je kopie tohoto souboru, kterou si někteří ze zaměstnanců mohou stáhnout a provést na něm nějaké výpočty, případně jej využít pro přehled toho, jak je na tom nyní společnost s výrobou. Problém ale nastává, pokud je soubor nutné editovat. To může v jednu chvíli pouze jeden uživatel, pokud nechtějí přijít o změny, které provedl ten druhý. To samé platí i se zápisem počtu vyrobených kusů na jednotlivých strojích. Není možné, aby na konci směny zaměstnanci, kteří tyto výrobky skutečně vyrobily, si jej otevřeli a zapsali výrobku, ale musí to provádět přes nějakého prostředníka, kterému tyto údaje sdělí. Ten je pak následně zapíše do tohoto dokumentu.

Výsledná aplikace musí umět evidovat stroje. Na každém stroji se vyrábí dávky. Dávky jsou vždy plánovány na určité množství výrobků (polotovarů), jejichž výroba trvá určité množství směn. Plán jednotlivých směn dávky se může lišit. Zaměstnanci pak mají možnost zapisovat skutečné vyrobené množství. Pokud je dávka dokončena dříve než byl plánovaný konec, začne následující plánovaná dávka dříve. Pokud je dokončena později, posouvá se začátek následující dávky také na pozdější dobu. Do aplikace se bude možné přihlašovat pomocí tří typů rolí. Dělník bude mít možnost pouze zapisovat skutečnou výrobu. Mistr bude moci vytvářet dávky, bude mít přístup ke koeficientům a stejně jako dělník může zapisovat a měnit výrobu. Administrátor pak bude moci vytvářet stroje, zaměstnance a bude mít dostupné i veškeré další možnosti, které může provádět mistr. Veškeré důležité změny na strojích, dávkách nebo zaměstnancích budou oznamovány zainteresovaným uživatelům. Půjde o klientskou aplikaci bez serverové části. Do aplikace se bude zaměstnanec přihlašovat pomocí loginu a hesla. Veškerá data budou ukládána mimo aplikaci na síťovém disku, aby z něj mohlo číst a zároveň do něj zapisovat více uživatelů zároveň.

4.1.2 Model požadavků

Z popisu problému jsem vytvořil následující modely s požadavky na aplikaci. Funkční požadavky jsem rozdělil na podniková pravidla (obr. č. 1), vlastnosti systému (obr. č. 2) a požadavky na uživatelské rozhraní (obr. č. 3). Při jejich vytváření jsem bral v potaz současné používané řešení (tj. dokument Excel) a také vlastnosti, které by zaměstnanci při práci mohli ocenit nebo jsou nezbytné pro fungování jiných částí, které si přáli.

Z pravidel jsou zajímavé informace o časových možnostech zápisu skutečné výroby. Dělníci mají na zápis nejkratší čas. Pokud neopraví chyby nebo zapomenou zapsat výrobu včas (do 24 hodin od ukončení směny), má možnost tak učinit ještě mistr. Je zde uvedeno také výše zmíněné zkracování a natahování dávek v kalendáři. S tím souvisí i možnost, že se skutečný začátek dávky může lišit od toho, který byl stanoven v době tvorby dané dávky. V používaném dokumentu Excel byly nejmenší dávky na dobu 8 hodin (jedné směny). Žádná dávka nemohla být kratší, protože na každou směnu byla jen jedna buňka. Hodnoty obou dávek se v ní tak museli



Obrázek 1: Model funkčních požadavků: podniková pravidla

sečítat a označení té méně důležité bylo uvedeno vedle hlavní. Toto pravidlo jsem se rozhodl zachovat z toho důvodu, že je to takto pro uživatele mnohem přehlednější, než kdyby směny byly rozdělené ještě na hodiny nebo dokonce minuty, desítky minut. Není takto zatěžován informacemi o malých (hodinových) dávkách, které jsou jednoduše sloučeny do větších dávek se společným produktem.

Z vlastností systému se ta nejdůležitější informace týká seřizovacích dávek. Ty se posouvají stejně jako ostatní dávky, ale na rozdíl od nich se automaticky nezkracují a neprodlužují. Vyznačují se tím, že se v nich nevyrábí žádné výrobky a jde pouze o přípravu na další dávky. Kromě nich existují ještě volné směny. Ty se naopak posouvat nedají, protože jejich termín je vždy svázán s konkrétním datem a časem. Protože se zaměstnanci zapomínají odhlašovat již ze současných systémů, bylo nutné přidat funkci na automatické odhlašování po uplynutí určité doby, kdy byl uživatel neaktivní v této aplikaci (neprovedl žádnou akci).

Role administrátora je určena především pro zaměstnance z kanceláří (např.

Obrázek 2: Model funkčních požadavků: vlastnosti systému

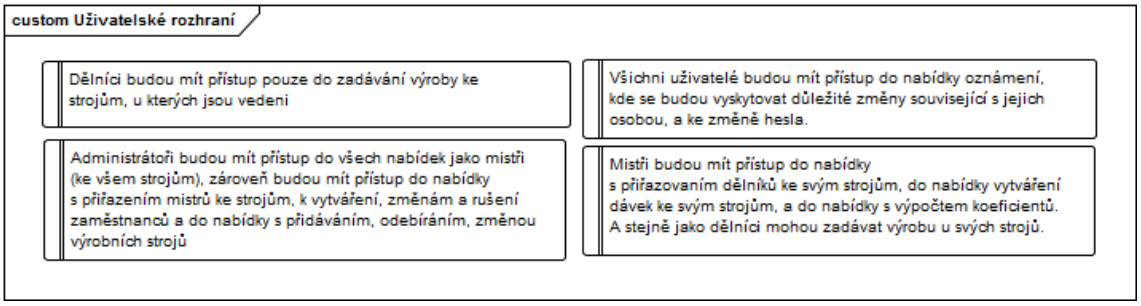
z IT), ale může být přidělena i nějakému mistrovi z výrobní haly, pokud je nutné, aby měl i přístup k vytváření zaměstnanců nebo strojů.

S nefunkčními požadavky (obr. č. 4) pak souvisí vlastnosti aplikace, které jsou „nad aplikací“. Jde především o to, že půjde o desktopovou aplikaci vytvořenou v Javě, o uložení dat mimo aplikaci, aby k datům mohli přistupovat zaměstnanci z různých počítačů zároveň, o rolích zaměstnanců ve výrobní hale a o jejich přihlašování do aplikace.

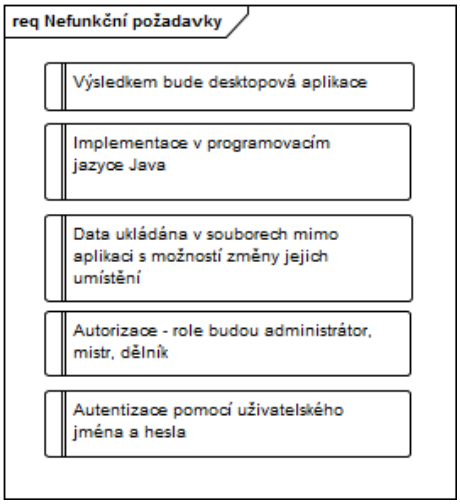
K systému budou přistupovat tři typy uživatelů. Dělníci, kteří budou mít svá práva v aplikaci nejvíce omezeny. Z výroby budou mít přístup pouze k zadávání skutečné výroby. Mistři, kteří budou moci navíc vytvářet, měnit a rušit dávky a také zobrazovat si výpis týdenních koeficientů výroby. Administrátoři pak budou mít přístup ke všem volbám, které v aplikaci existují. Zavádění strojů do výroby, jejich vyřazování, vytváření zaměstnanců a jejich přihlašovacích údajů, přiřazování zaměstnanců ke strojům a samozřejmě veškeré tyto záznamy měnit a rušit. Rozdělení rolí (a zároveň i aktérů systému) je znázorněno na obr. č. 5.

Z požadavků a dalších získaných informací jsem vytvořil model případů užití (obr. č. 6), který ukazuje rozdíly v přístupu aktorů k jednotlivým případům užití.

Ke každému případu užití jsem následně vytvořil scénář. Celkem jich je 23



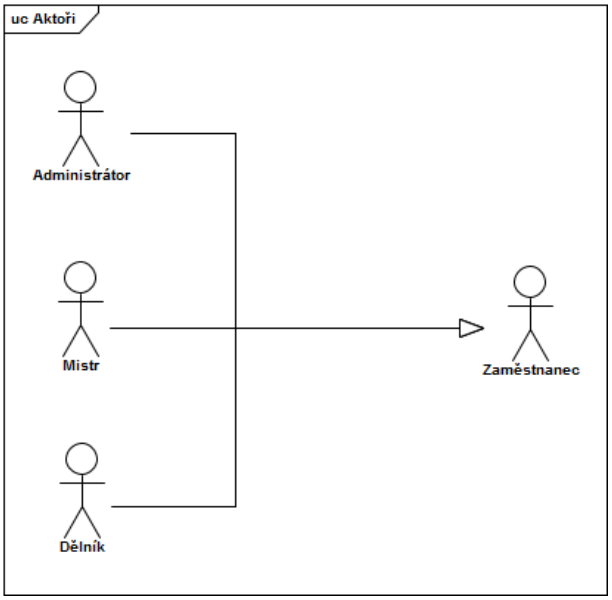
Obrázek 3: Model funkčních požadavků: uživatelská rozhraní



Obrázek 4: Model nefunkčních požadavků

a jsou uvedeny v příloze. Blíže přiblížím pouze dva z nich. Vybral jsem případy užití, které mají největší vztah k výrobnímu plánu. První z nich se zabývá vytvářením výrobních dávek (obr. č. 22). Tento případ užití vyvolává administrátor nebo mistr a v aplikaci vytvoří u konkrétního stroje výrobní dávku s určitým plánem jednotlivých směn. Nemá žádné podmínky před spuštěním a ani po něm. Uživatel vybere v aplikaci stroj, klikne na příslušné tlačítko a zobrazí se mu formulář pro tvorbu dávek. V něm vyplní veškeré potřebné údaje a vybere způsob rozpočítání množství mezi směny. Na výběr má volbu ručního přiřazení, při které se mu zobrazí další formulář, ve kterém vyplní plán ke každé směně podle svého vlastního uvážení. Po zadání hodnot pak systém zkontroluje, zda součet zadaných hodnot odpovídá celkovému množství. Jestli neodpovídá, tak ho systém na tuto skutečnost upozorní a nechá ho údaje opravit. Pokud zvolí automatické rozpočítání, tak systém celkové množství rovnoměrně rozpočítá mezi směny sám. Následně systém dávku vytvoří a vyhodí uživateli hlášku o úspěšném vytvoření dávky. Jako poslední krok systém vytvoří všem zainteresovaným uživatelům zprávu, že proběhla tato změna.

Druhý případ užití se věnuje zápisu skutečně vyrobených kusů výrobků (polotovarů). Tento případ užití může spustit dělník, mistr nebo administrátor. Tedy



Obrázek 5: Model aktérů

libovolná role v aplikaci. Uživatel vybere směnu a stroj, ke kterému výroba náleží, zadá počet vyrobených kusů a potvrdí akci. Systém pak tuto výrobu zapíše a informuje uživatele o úspěšném zadání údajů. Jako poslední akce proběhne vytvoření zpráv o této akci všem zainteresovaným osobám. Těmi jsou mistři, kteří mají tento stroj na starost, a dělníci, kteří na něm pracují. Případ užití nemá žádné vstupní a výstupní podmínky.

4.2 Analýza a návrh řešení

4.2.1 Analytický diagram tříd

Celkem bude potřeba 7 tříd (obr. č. 9). Třída přihlášení se bude starat o přihlašování a odhlašování ze systému, o načítání dat a bude mít funkce na vytváření vyskakovacích oken (okno pro upozornění, potvrzení, informování, atd.). Výroba bude představovat výrobní halu. V ní budou do polí uloženy stroje a zaměstnanci, takže se bude starat o jejich přidávání a zároveň i odstraňování. Bude mít funkce na výpočet koeficientů výroby pro jednotlivé stroje a celý podnik. U zaměstnanců budou vedeny jejich přihlašovací údaje a funkce, kterými jim budou přidělovány. Dále pak role pro přihlášení (administrátor, mistr nebo dělník) a další osobní údaje. Ke každému zaměstnanci bude veden seznam oznámení (zpráv). Ty budou mít atribut s datum vytvoření a text samotné zprávy. Starší zprávy se budou ze systému mazat, proto je u zaměstnance vedena funkce na jejich mazání.

Stroje mají vždy přiřazeného alespoň jednoho zaměstnance jako mistra a libovolný počet dělníků, proto jsou u stroje funkce na jejich přidělování a odstraňování. Ke každému stroji budou navíc vedeny dávky (výrobní a nevýrobní) a stejně tak

i zde musí být funkce na jejich přidělování a odstraňování z pořadníku (pole). Také je nutné mít funkce, které budou posouvat dávky v tomto pořadníku dopředu a dozadu. Stroj může být aktivní (lze na něm vytvářet dávky) a naopak neaktivní (nelze přidávat dávky). Bude se u něho evidovat defaultní nastavení směn, ve kterých je obvykle používán. Každý stroj má své vlastní směny volna, kdy na něm nikdo nepracuje (např. neděle, svátky, dovolená), i když spadají do defaultních směn.

V dávkách se budou evidovat tři seřazené seznamy o stejném množství prvků. Data směn, plánovaná výroba směny a skutečná výroba směny. Dále se bude evidovat datum začátku a konce dávky (původní, plánované a skutečné). Původní data se nastavují pouze při vytváření dávky, zatímco plánované se mění s posunem dávky a skutečné se nastavují až po zapsání výroby. Důležité jsou atributy pro zápis celkového množství, které musí být shodné se sumou všech plánů směn. Dále pak seřízení, které určuje zda se jedná o výrobní nebo nevýrobní (seřizovací) dávku, a označení části, které se nastavuje pouze pokud tato dávka vznikla rozdělením jiné dávky. Důležité jsou funkce na ruční nastavení plánu směn a na rovnoměrné rozložení. Výrobky (polotovary) z dávky jsou vyráběny vždy z jednoho materiálu o určitém množství kusů (vycházím z informací uvedených v dokumentu Excel).

4.2.2 Sekvenční diagramy

Ke každému případu užití jsem vytvořil sekvenční diagram. Celkem tak vzniklo 23 sekvenčních diagramu. Na tomto místě uvedu a blíže přiblížím pouze dva z nich. Všechny ostatní lze nalézt v příloze.

Prvním je přiřazování dělníků ke stroji (obr. č. 10). Jde o velice podobný případ jako přiřazení mistrů ke stroji. Rozdíl je pouze v roli, kterou přidělovaný zaměstnanec zastává. Uživatel, v tomto případě mistr nebo administrátor, si nejprve nechá vypsat seznam všech dostupných strojů. U některého z nich si zavolá funkci, která mu zobrazí formulář s přiřazováním dělníků k tomuto stroji. Ten je předvyplněn současným stavem stroje, který může uživatel následně změnit. Uživatel vybere jednoho zaměstnance a zavolá funkci na jeho přiřazení ke stroji. Nakonec vznikne pro všechny zainteresované osoby upozornění o této akci.

Druhým je sekvenční diagram o změně přihlašovacích údajů (obr. č. 11). Tu provádí vždy jen uživatel, jehož role je administrátorem. Ten se spouští v případě ztráty přihlašovacích údajů, změny příjmení (a tudíž i loginu, pokud je login vytvářen z příjmení) a v dalších závažných případech. Administrátor si otevře formulář se seznamem zaměstnanců. V nich vybere konkrétního zaměstnance a vybere funkci pro změnu přihlašovacích údajů. Systém mu vrátí formulář se současnými údaji uživatele. Zde si administrátor může vybrat, zda pouze vygeneruje nové heslo nebo udělá i změnu loginu. Pro první volbu zde bude tlačítko s funkcí na generování nového hesla. Ve druhém případě může přepsat obdržené řetězce pro osobní údaje a login. Po stisknutí druhého tlačítka dojde ke kontrole, zda byly změněny osobní údaje. Pokud ano, tak dojde k zavolání funkce na jejich změnu, jinak se tato funkce vynechává. Následně se zavolá funkce na změnu loginu, jejímž parametrem bude

hodnota z příslušného pole. Se změnou loginu se automaticky generuje i nové heslo, které je zobrazeno administrátorovi, aby je mohl předat danému uživateli. Pokud došlo ke změně loginu, tak se na úplném konci vytváří zpráva pro zainteresované osoby.

4.3 OCL

Po těchto krocích jsem nalezl několik pravidel, které by bylo možné převést na OCL a následně i naprogramovat. V příloze jsou k dispozici v klasické podobě. Na obr. č. 12 pak ve zjednodušené podobě vytvořené v programu Eclipse s rozšířením Papyrus. Z analytického diagramu tříd jsem odstranil prvky, kterých se omezení netýkala, aby byl výsledek přehlednější. Mezi těmito omezeními jsou:

1. délka hesla a loginu musí být delší jak 5 znaků,
2. číslo zaměstnance, login, SAP označení stroje a materiálu, číslo stroje a označení dávky musí být jedinečné,
3. počet směn u dávky a počet kusů materiálu musí být větší jak 1,
4. v případě že jde o seřizovací dávku, musí být počet kusů roven 0,
5. vždy musí existovat jeden administrátor v aplikaci,
6. k neaktivnímu stroji nelze přidávat dávky,
7. funkce zobrazující kolik zbývá vyrobit,
8. celkový počet vyrobených kusů nemůže být větší jak plánovaný,
9. každý stroj musí mít alespoň jednoho mistra,
10. hodnoty jsou vždy zadané (např. SAP označení a počet kusů materiálu),
11. get/set metody (např. getPocet(), setPocet()).

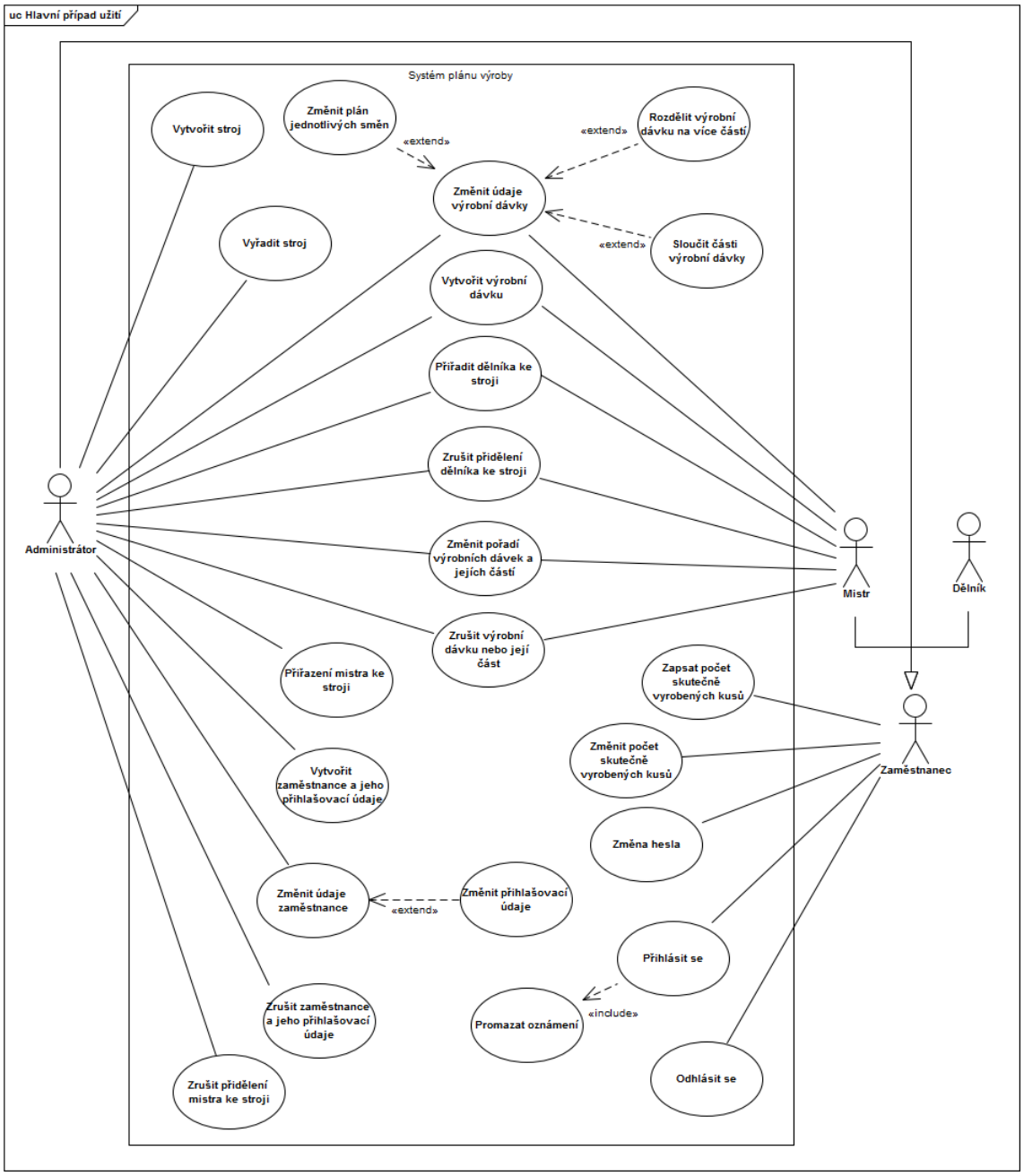
Posledních dvou případů OCL se v mé práci nachází hned několik a výsledný kód OCL je mezi sebou natolik podobný, že jsem se rozhodl ke každé skupině napsat jen několik příkladů.

Jelikož nové zaměstnance a změnu role zaměstnance mohou provádět pouze administrátoři, tak po smazání posledního administrátora by se aplikace stávala prakticky nepoužitelná a muselo by se začít opět od nuly. Proto zde existuje omezení na počet administrátorů. Toto omezení se dá řešit dvěma způsoby. Buď se skutečně před každým smazáním a změnou role administrátora bude kontrolovat, zda ještě nějaký jiný administrátor existuje, nebo se určí jeden administrátor jako nesmazatelný a bez možnosti změnit roli. Rozhodl jsem se pro druhou možnost, a tak administrátor s indexem 0 nelze smazat, ani mu změnit roli. Heslo však u něho změnit lze jako u ostatních uživatelů.

Každý stroj musí mít vždy alespoň jednoho mistra. Souvisí to s uspořádáním výrobní haly v podniku, kde každá sekce a stroje v ní umístěné spadají pod nějakého pracovníka (mistra). Toto pravidlo platí i pro stroje, které nejsou zařazeny do užívání. Je ale přípustné, aby mistra stroje dělal zaměstnanec s rolí administrátora. Toho by se využilo například v případě, kdy má reálný mistr administrátorská práva (např. z důvodu, aby mohl vytvářet účty zaměstnancům), nebo když ve výrobě žádný mistr neexistuje.

4.4 Návrh řešení

V návrhové části jsem vytvořil dva diagramy tříd. První z nich byl předběžný, ze kterého jsem následně vycházel pro implementaci. Při samotné implementaci jsem však v tomto původním diagramu udělal spoustu změn, a proto jsem k práci připojil ještě konečný diagram tříd, který odpovídá tomu, co je přesně naprogramováno. Některé funkce jsem rozdělil na více částí, aby byl program přehlednější, využíval jsem statické proměnné a funkce, uživatelské rozhraní jsem tvořil pomocí pojmenovaných tříd a mnoho dalšího. Oba diagramy jsou příliš rozsáhlé na to, aby se mohly umístit zde na stránku. Proto jsou v této práci k nalezení jen v elektronické podobě v příloze společně se zdrojovým kódem.



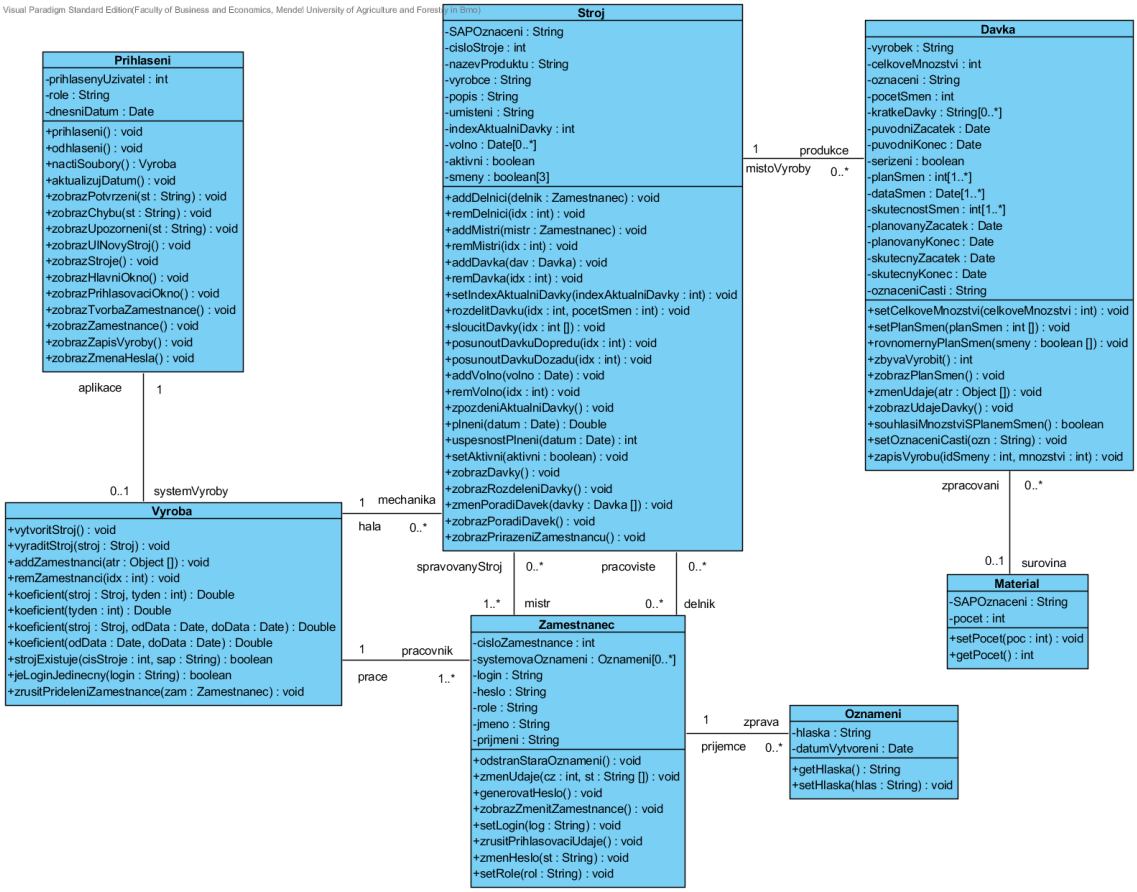
Obrázek 6: Model případů užití

Případ užití: Vytvořit výrobní dávku
ID: 5
Stručný popis: Vytvoří u zvoleného stroje výrobní dávku (co se bude vyrábět, v jakém množství, označení dávky, kolik dní je na výrobu (nebo počet směn), ID části dávek kratších jak 8 hodin, jejichž plnění se v systému sledovat nebude, ID materiálu). Uživatel pak může ručně zadat plán na jednotlivé směny nebo může nechat systém, ať toto množství rovnoměrně rozloží.
Hlavní aktéři: Mistr, Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere stroj, u kterého chce vytvořit výrobní dávku, a zvolí tuto akci 2. Systém zobrazí formulář s tvorbou dávek 3. Mistr nebo administrátor vyplní veškeré položky 4. Mistr nebo administrátor zvolí formu přidělení plánu na směny (ručně nebo automaticky) 5. Pokud bylo zvoleno ruční přidělování, pak: 5.1 Systém zobrazí formulář s plánem směn 5.2 Mistr nebo administrátor zadají plán výroby pro jednotlivé směny a potvrdí ukončení zadávání 5.3 Systém zkontroluje plánované množství, pokud sedí, pak: 5.3.1 Systém vytvoří novou dávku v systému a u daného stroje ji zařadí za veškeré dříve vytvořené dávky 5.3.2 Systém informuje mistra nebo administrátora o úspěšném vytvoření výrobní dávky 5.4 Nebo: 5.4.1 Systém upozorní uživatele na chybu a umožní mu vrátit se do bodu 5.1 6. Nebo: 6.1 Systém rovnoměrně rozpočítá plán dávky mezi směny 6.2 Systém informuje mistra nebo administrátora o úspěšném vytvoření výrobní dávky 7. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

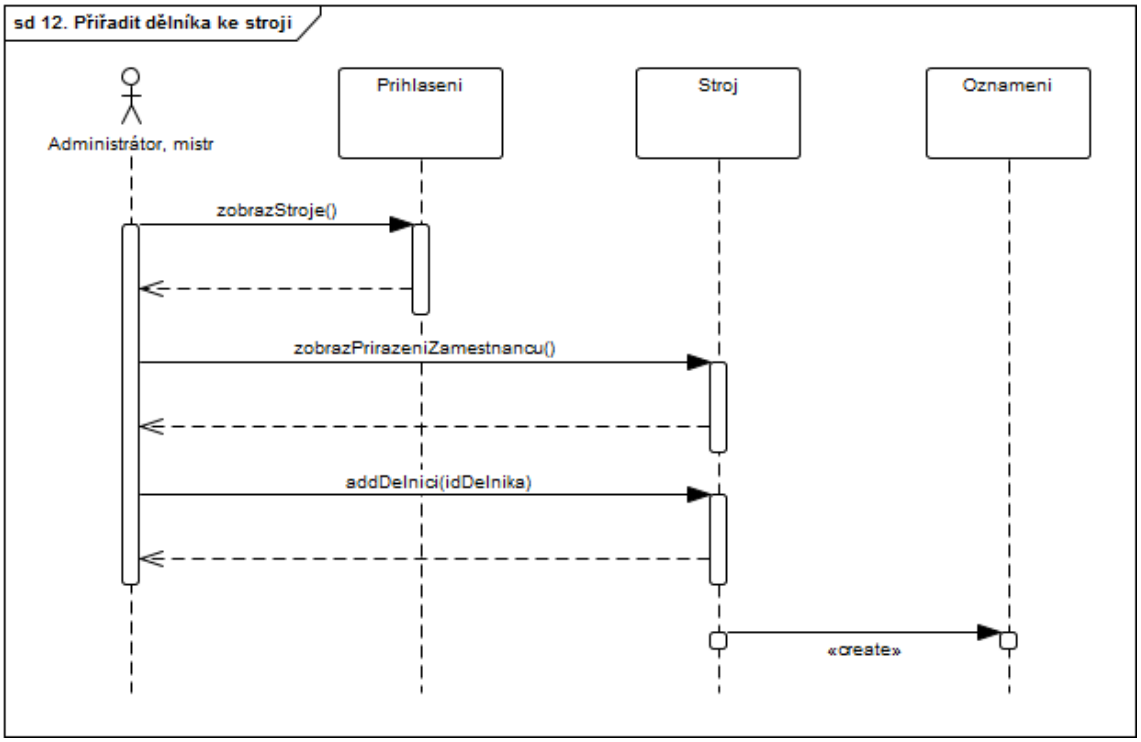
Obrázek 7: Vytvořit výrobní dávku

Případ užití: Zapsat počet skutečně vyrobených kusů
ID: 18
Stručný popis: Uživatel systému může zadat, kolik toho za směnu vyrobil. Dělník může vyplňovat údaje pouze ke strojům, ke kterým je přiřazen. Mistr nebo administrátor může v případě nutnosti zadat údaje za dělníka.
Hlavní aktéři: Uživatel
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel vybere stroj a směnu, kterou chce vyplnit 2. Uživatel vyplní údaje o výrobě a potvrdí zadávání 3. Systém informuje uživatele o úspěšném zadání výroby 4. Systém přidá do oznamovací části mistrů a dělníků přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

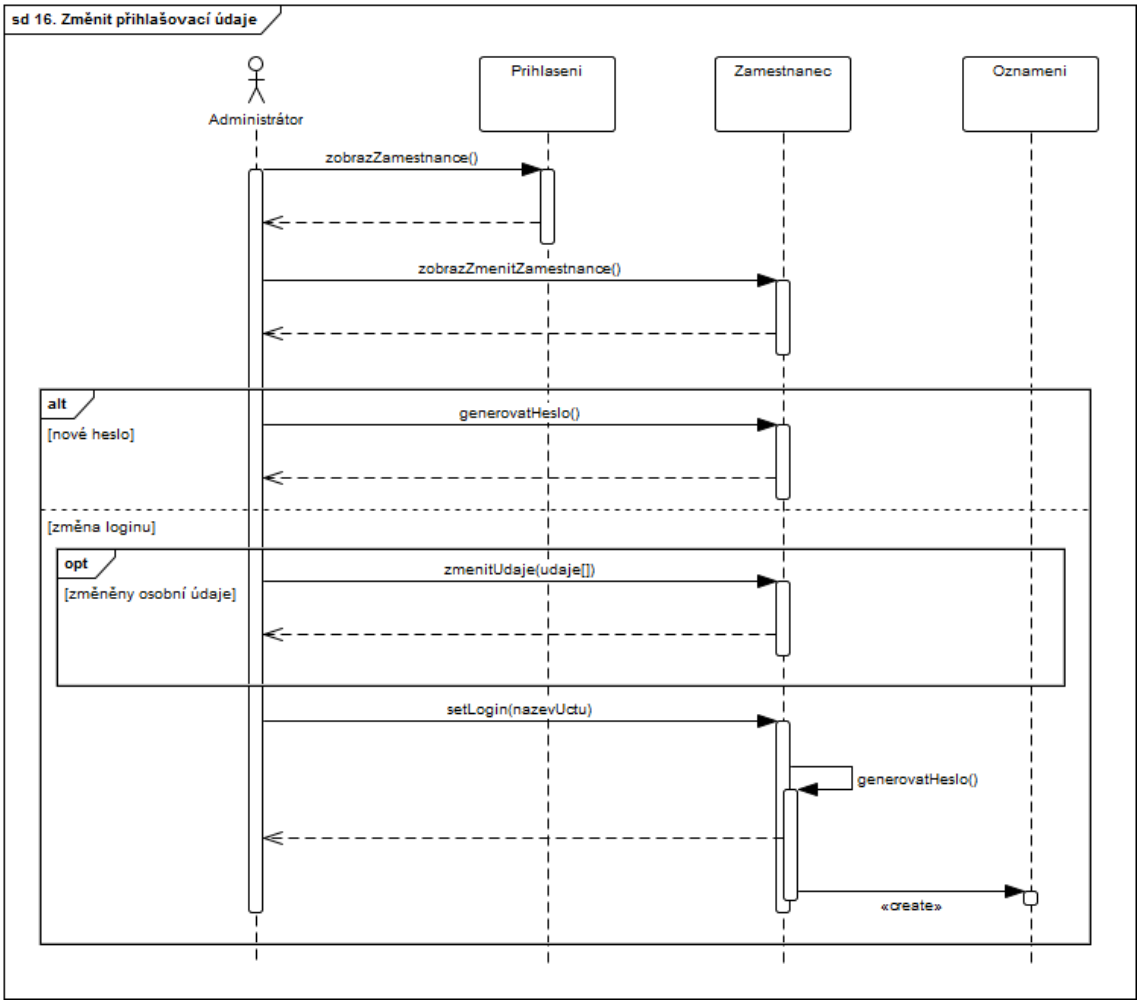
Obrázek 8: Zapsat počet skutečně vyrobených kusů



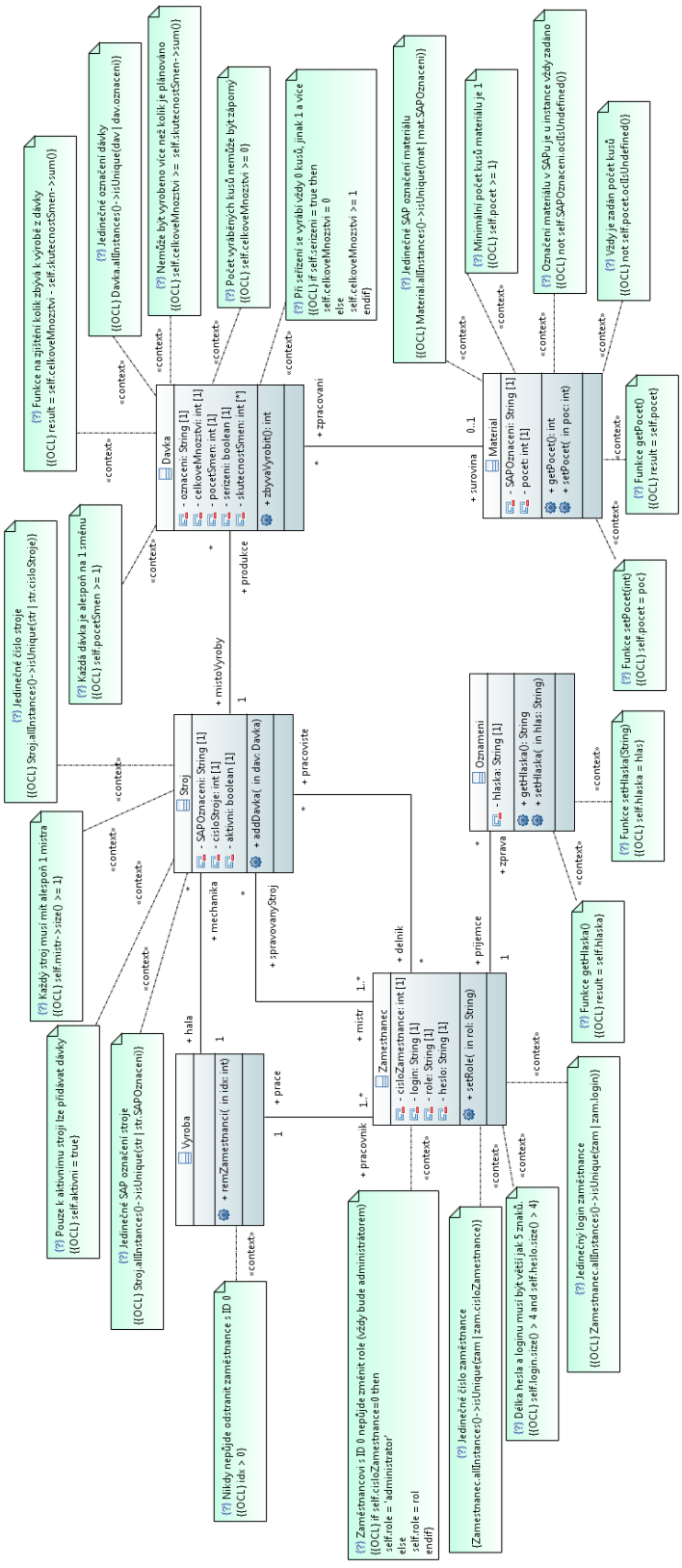
Obrázek 9: Analytický diagram tříd



Obrázek 10: Sekvenční diagram: přiřadit dělníka ke stroji



Obrázek 11: Sekvenční diagram: změnit přihlašovací údaje



Obrázek 12: Zjednodušený diagram tříd s OCL prvky

5 Implementace aplikace

5.1 OCL

Veškeré OCL prvky uvedené v předcházející části jsem převedl do programovacího jazyka Java. Všechny je možné dohledat ve zdrojových souborech. Zmíním zde proto jen některé z nich.

Omezení č. 7 věnované funkci na výpočet zbývajícího počtu výrobků je možné převést z OCL kódu na kód Javy pomocí cyklu, ve kterém se projdou veškeré položky věnované zápisu skutečné výroby a jejich hodnoty sečtou. Následně se tento výsledek odečte od hodnoty atributu této dávky s názvem celkoveMnozstvi, který slouží k uložení celkového vyráběného množství dávky. Výsledný kód tak vypadá následovně.

```
public int zbyvaVyrobit() {
  int celkemVyrobeno = 0;
  for(int i~= 0; i~< pocetSmen; i++){
    celkemVyrobeno += skutecnostSmen.get(i);
  }
  return celkoveMnozstvi - celkemVyrobeno;
}
```

Omezení č. 9, které se věnuje tomu, že každý stroj musí mít přiděleného alespoň jednoho mistra lze vyřešit následujícím způsobem.

```
public Stroj(Zamestnanec mistr) {
  mistri.add(mistr);
}
public void remMistri(int idx) throws Exception {
  if(mistri.size()>1){
    mistri.remove(idx);
  }else{
    throw new Exception("Mistra nelze odstranit od stroje,
    protoze je jedinym mistrem tohoto stroje!");
  }
}
```

Nejprve je nutné zajistit konstruktor, který musí být pouze parametrický. Takto zajistíme, aby při vytváření nového stroje bylo vynuceno zadání nějakého zaměstnance. Dále je ve funkci pro odstranění mistra od stroje kontrolováno, zda jsou v množině pro ukládání mistrů alespoň dva mistři. Pokud ano, tak je možné jednoho smazat. Jinak je vyhozena výjimka nadřazené funkci, která odstranění mistra zavolala. V ní je pak možné na tuto výjimku nějak reagovat. Nejsnadnějším způsobem je upozornit na tuto skutečnost uživatele vykreslením okna s upozorněním o porušení tohoto omezení.

Následující část kódu pak ukazuje převod omezení č. 2 a 3, která se zabývají počtem kusů materiálu a jedinečným označením materiálu. Oba atributy jsou vynucovány konstruktorem. Nejprve se testuje počet kusů materiálu. Ten nesmí být nulový nebo záporný. Pokud se tak stane, je vyhozena výjimka metodě, která tento konstruktor vyvolá. Pokud projde, tak se testuje jedinečné označení materiálu. Je nutné projít veškeré instance materiálu, který se ve výrobě nachází, dokud se nenarazí na některý, který by jedinečnost porušoval. V tomto případě je opět vyhozena výjimka nadřazené metodě, která tento konstruktor vyvolala, a cyklus je přerušen. V opačném případě je vytvořen objekt s oběma atributy. Stejná kontrola je nutná i u případných metod sloužících ke změně daných parametrů. U jedinečného označení se ale musí kód lehce upravit kvůli případu, kdy uživatel mění označení na tu samou hodnotu. V takovém případě by uživatel obdržel vždy chybu, že dané označení nejde změnit, protože již existuje. Což by jej mohlo zmást. Proto je nutné měněný prvek z kontrolované množiny vyloučit nebo jej přeskočit.

```
public Material(String sap, int poc) throws Exception {
  if(poc<1){
    throw new Exception("Dílců musí být alespoň 1 kus!");
  }else{
    boolean jedinecneOznaceni = true;
    for(int i~= 0; i~< Vyroba.material.size(); i++){
      if(Vyroba.material.get(i).getOznaceni().equals(sap)){
        jedinecneOznaceni = false;
        break;
      }
    }
    if(!jedinecneOznaceni){
      throw new Exception("Oznaceni materialu neni jedinecne!");
    }else{
      SAPOznaceni = sap;
      pocet = poc;
    }
  }
}
```

5.2 Odhlášení po 30 minutách neaktivity

V programu bylo nutné vytvořit automatické odhlášení po 30 minutách neaktivity, kdy uživatel nic neprovedl v aplikaci. Nešel jsem cestou, kdy se aktivitou myslí pohyb myší, stisknutí tlačítka klávesnice nebo jiného ovládacího prvku kdekoliv v operačním systému. Chtěl jsem, aby šlo skutečně jen o akce uvnitř programu. Po přihlášení do aplikace se spustí následující kód, který vytvoří časovač na 30 minut. Po uplynutí této doby je zavolána funkce na odhlášení, která uživatele vrátí do

přihlašovací obrazovky, kde se musí znovu přihlásit. Ke každému posluchači akcí v programu jsem následně přidal restart instance tohoto časovače. Kdykoliv tedy uživatel klikne na nějaké tlačítko nebo záložku, časovač začne počítat opět od nuly.

```
public static void spustCasovac(){
  int delay = 1800000;
  ActionListener taskPerformer = new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      casovac.stop();
      Prihlaseni.odhlaseni();
    }
  };
  casovac = new Timer(delay, taskPerformer);
  casovac.start();
}
```

5.3 Aktuální datum

Ke správnému fungování aplikace je nutné, aby si aplikace zjistila správné datum a čas. Existují dva způsoby jak toho dosáhnout. Buď odeslat požadavek na aktuální datum některému veřejnému časovému serveru (např. nist.gov) a nebo si datum vezme z operačního systému. Při stahování času ze serveru může být problémem to, že bude server nedostupný nebo počítač nebude mít přístup do sítě mimo podnik. Při testování tohoto řešení jsem narazil i na problém, kdy server záměrně neodpovídá na požadavky vzniklé několik sekund po sobě. Pravděpodobně jde o ochranu před útoky. Pokud se ale použije metoda s převzetím systémového času, tak naopak můžeme narazit na špatné nastavení v počítači. Proto jsem zvolil dvojí zjišťování. Aplikace si nejprve vezme čas systémový a poté se ještě pokusí spojit s časovým serverem a tento čas přepsat. Pro zjištění času ze serveru jsem využil třídy z knihovny „Commons Lang“ a výsledný kód vypadá následovně.

```
dnesniDatum = Calendar.getInstance();
String TIME_SERVER = "time-b.nist.gov";
NTPUDPClient timeClient = new NTPUDPClient();
InetAddress inetAddress = InetAddress.getByName(TIME_SERVER);
TimeInfo timeInfo = timeClient.getTime(inetAddress);
long returnTime =
timeInfo.getMessage().getTransmitTimeStamp().getTime();
dnesniDatum.setTime(new Date(returnTime));
```

5.4 Vzdálené soubory s daty a přihlašovacími údaji

K datům je potřeba přistupovat z více míst. Proto bylo nutné veškerá zpracovávaná data uložit mimo programovou strukturu. K tomuto účelu jsem vyčlenil 2 soubory.

Jeden, do kterého se ukládají data o výrobě, a druhý s přihlašovacími údaji, který taktéž musí být uložen mimo aplikaci.

Soubor s daty je ukládán v objektovém formátu. Ze třídy výroba jsou v něm ukládány atributy stroje, materiál a zaměstnanci. Pod stroji se skrývá celá výrobní struktura s přiřazenými dávkami, mistry, dělníky, směnami a přiřazeným materiálem. Materiál a zaměstnanci jsou oproti tomu jednoduché seznamy toho, co existuje ve firmě. Ve výrobní struktuře se ale tyto instance objevit nemusí. Například dělník nemusí být uveden u žádného stroje a stejně tak i materiál u žádné dávky. Pro správné fungování je nutné oba soubory synchronizovat. K tomu jsem využil třídu „RandomAccessFile“ a „FileDescriptor“. Pokud uživatel změní nějaký atribut a zavolá funkci na uložení změn, tak tato funkce způsobí, že se soubor zablokuje pro ostatní uživatele. Před samotnou změnou uložených se ještě musí stáhnout aktuální verze dat. Kdyby se tak neučinilo, tak by mohly být ztraceny změny, které učinili uživatelé během doby, kdy si uživatel naposledy stáhl data a kdy provedl tyto změny. Stejně tak je nutné vždy ještě zkontrolovat, zda změny provedené uživateli je možné vůbec uskutečnit. Zda daný prvek nebyl v mezičase odstraněn už jiným uživatelem nebo neproběhly jiné změny, které by tuto změnu znemožnily.

```
RandomAccessFile fileData =
new RandomAccessFile(Prihlaseni.cestaData, "rw");
FileDescriptor fdData = fileData.getFD();
...
fileData.close();
```

Soubor s uživateli a jejich hesly jsem udělal odděleně oproti ostatním datům. Data jsou v něm ukládána do struktury souboru s vlastnostmi (properities file). V Javě existuje pro tento typ souborů speciální třída s názvem „Properties“. Ta umožňuje ukládat data ve formátu klíč a údaj. V tomto případě tedy název účtu (login) a hash kód hesla tohoto účtu. Pro výpočet hash kódů z hesel jsem využil externí knihovnu „Jasypt“. Knihovna nabízí dvě síly zabezpečení hesla pomocí algoritmu SHA (lehčí a silnější). K dispozici je i třetí varianta s individuálním nastavením parametrů. Lehčí potřebuje pro výpočet méně prostředků a jelikož data v aplikaci nejsou pro podnik až tak cenná, tak by toto zabezpečení mohlo stačit. Proto jsem vybral lehčí typ zabezpečení.

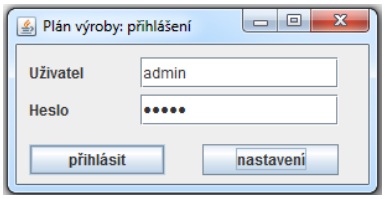
K oběma souborům je možné změnit cestu, kde jsou uloženy. A to v přihlašovací nabídce pod tlačítkem nastavení.

5.5 Použité externí knihovny

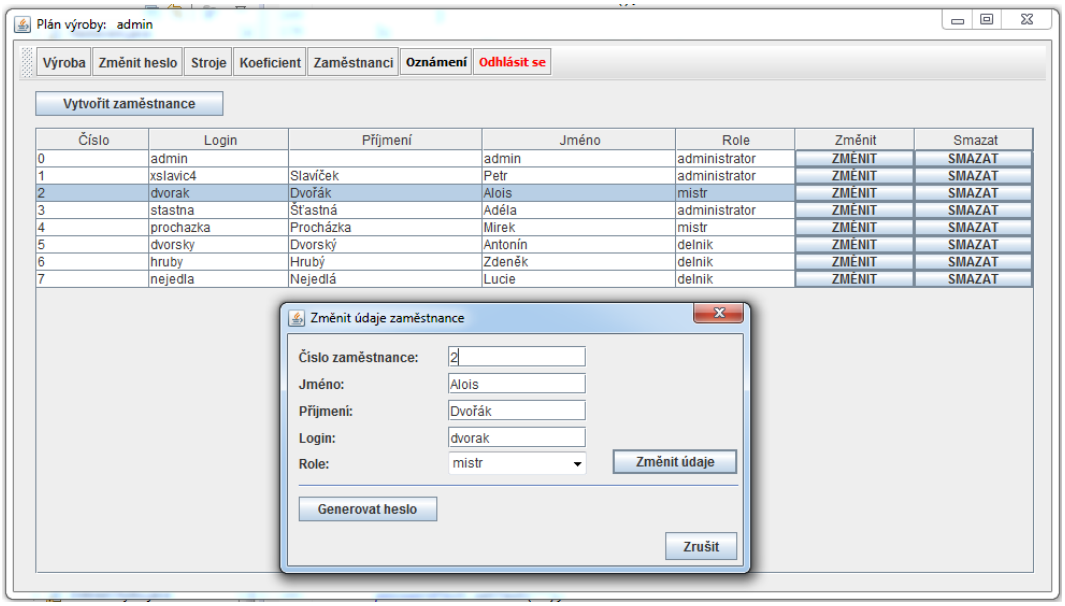
Pro hesla jsem využil knihovnu Jasypt. Ta má své využití pro šifrování a dešifrování hesel, řetězců, čísel i bitů. Pro kontaktování časového serveru knihovnu Commons Lang. Tato knihovna v podstatě jen rozšiřuje možnosti java.lang. Využívá se pro manipulaci s řetězci, k serializaci, reflexi, souběžnosti a mnohým dalším věcem.

Poslední je pak knihovna JDatePicker. Tato knihovna je navržená pro jednoduché přidání kalendáře a výběr data do rámců aplikace.

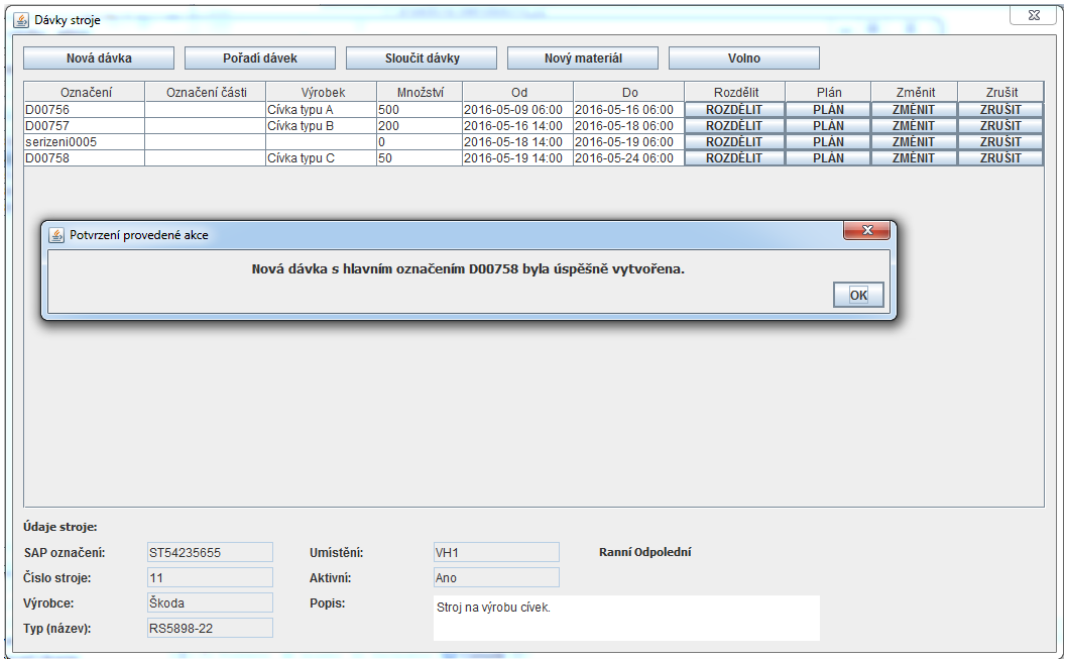
5.6 Ukázky z uživatelského rozhraní



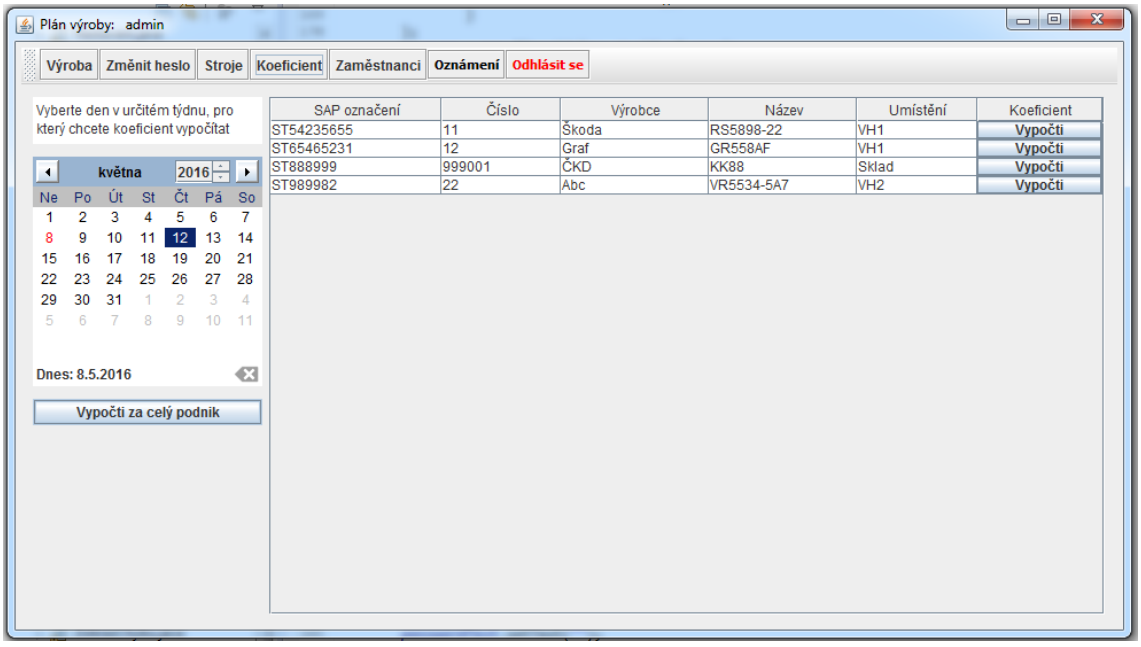
Obrázek 13: Uživatelské rozhraní: přihlašovací obrazovka



Obrázek 14: Uživatelské rozhraní: záložka zaměstnanci



Obrázek 15: Uživatelské rozhraní: dávky stroje



Obrázek 16: Uživatelské rozhraní: záložka koeficient

6 Diskuse a využití pro praxi

6.1 Aplikace výrobního plánu

Podnik v současnosti provádí výpočet řešených koeficientů výroby pomocí výpočtů v Excelu. Není mi známo, že by existoval informační systém, který by v základu nabízel u výrobního plánu výpočet řešeného koeficientu výroby. ERP systém SAP tento výpočet nenabízí, takže podniky využívající tento systém, které řeší podobný problém, jej musí řešit svépomocí. Řešením by byla úprava systému pomocí programové nástavby SAP, jakým je například QI Builder, kde by se mohl výpočet výrobních koeficientů naprogramovat přímo do QI. Jiným řešením by byl přechod ze SAP na jiný systém. Například zmíněný QI s QI Builderem. Taková změna by pro podnik mohla být složitá. Například by bylo nutné přeškolit zaměstnance a je zde riziko, že by mohlo dojít ke ztrátě některých využívaných funkcionalit, které by nový systém nenabízel. Domnívám se, že jediným správným řešením je tak integrace výpočtu do SAP nebo vytvoření nějaké jednoduché aplikace namísto používání Excelu.

Úprava SAP by byla pro podnik lepším řešením než vytváření nové aplikace. Uživatelé již se SAP umí pracovat, a tak by je jen stačilo naučit s touto novou částí. Avšak nepříjemnou vlastností by byla kontrola, kdo údaje o vyrobených výrobcích do systému zadal. Podnik nemá dostatek licencí SAP, aby mohl mít každý zaměstnanec svůj vlastní přístup, a tak jsou některé účtu sdíleny mezi více zaměstnanci. Mé řešení umožňuje každému zaměstnanci vytvořit účet a je tak možné sledovat, kdo udělal jakou změnu. Problémem úpravy SAP by mohla být i jeho cena. Z tohoto hlediska je vzniklá aplikace lepší.

Nevýhodou vzniklé aplikace je objem přenášených dat. Datové soubory zabírají přibližně méně jak dvojnásobek toho co dokument Excel. To je způsobeno především tím, že se v něm ukládají i jiné údaje. Každá změna, která se v datech provede, musí být provedena u klienta na celém objemu dat. Klient tak musí stáhnout data, která vůbec nepotřebuje, aby u nich mohl změnit jen některou úzkou část, a následně pak všechny data odeslat zpět na síťový disk. Čím déle se bude aplikace využívat, tím větší bude datový soubor. Podobný problém byl i u dosavadního řešení s dokumenty Excel. Uživatel si je nejprve musel stáhnout k sobě, pozměnit a následně odeslat zpět. Zlepšit situaci by mohlo, kdyby uživatel mohl provést více změn najednou. Toto vylepšení by bylo možné přidat i do této aplikace. Během práce uživatele by se jen ukládaly veškeré provedené změny do nějakého seznamu a po určité době (automaticky nebo ručně) by došlo k jednorázové změně v datových souborech. Nevýhodou by bylo, že by mohlo být větší množství těchto akcí stornováno, protože některý jiný uživatel provedl v mezičase změny, se kterými tyto nejsou slučitelné.

Jiným řešením by bylo datový soubor „ořezávat“. Datový soubor by byl rozdělen na více částí, kdy se každá bude věnovat například jiné skupině týdnů. Aplikace by si pak stahovala jen to časové rozmezí, které potřebuje. Podobným způsobem je rozdělen i současný dokument. Problémem tohoto řešení je, že by se v každém

souboru musela opakovat data nezávislá na týdnu (např. informace o strojích nebo zaměstnancích). Proto by bylo nutné tuto variantu rozšířit i o věcné dělení, kde by tato data měla vlastní soubory. Avšak tím úplně nejlepším by bylo vytvořit aplikaci pro serverovou část, která by se starala o komunikaci s jednotlivými aplikacemi. Zasílala by jim jen ta data, která aktuálně potřebují, a zpět by dostávala jen informace, co má u sebe změnit a jakým způsobem. Nevýhodou tohoto řešení je, že musí být zajištěno místo, kde tato aplikace poběží nonstop v cloudu.

Výhody implementované aplikace:

- možnost přístupu více uživatelů k datům zároveň, než jak je tomu v současnosti u Excelu, kde musí být zápis výroby prováděn přes prostředníka,
- každý zaměstnanec výroby může mít svůj vlastní účet k přístupu do aplikace,
- automatické přizpůsobení plánu podle aktuální situace ve výrobě, pokud se u dávky výrobky nestihnou vyrobit před plánovaným koncem (nebo naopak dokončí rychleji), tak se automaticky prodlouží (zkrátí) její plánovaná doba a dávkám následujícím po ní se posune datum začátku a konce výroby,
- možnost přidávání doby na seřízení,
- přidávání směn s volnem,
- jednoduché sledování změn v datech (zprávy uživatelům),
- snadný výpočet koeficientů výroby, který není závislý na struktuře dat (jako u Excelu).

Výhodou, ale i nevýhodou je pak to, že je tato vzniklá aplikace šitá na míru podniku Novibra Boskovice a jejímu problému.

6.2 Vývoj softwaru s OCL

Hlavním přínosem OCL ve vývoji softwaru je zlepšení komunikace mezi analytikem, návrhářem a programátorem. OCL dokáže přesně specifikovat na UML modelu integritní omezení. Interpretace omezení v přirozeném jazyce není jednoznačná. Druhá strana komunikace mu nemusí rozumět tím způsobem, jakým bylo zamýšleno. Jasně stanovená pravidla OCL zajistí, aby se tyto důležité myšlenky pro vývoj dostaly od analytika až k programátorovi v naprosto stejném smyslu. Mohou jej ale využít i jiné role vývoje, ke kterým se dostane po vytvoření. Testerům může například prozradit, co je v aplikaci skutečně chybou a co bylo naopak záměrem. Designérům velikosti grafických prvků (např. pokud je omezena velikost atributu), ale také jim mohou napomoci vybrat ten správný typ prvku, který se pro zobrazení tohoto objektu hodí, a i jeho obsah (např. vypisovat jen seznam s jedinečnými hodnotami nebo tabulku s více sloupci). Také může sdělit koučům, kteří zaučují do používání softwaru, jaké hodnoty tito zaměstnanci nemají do aplikace zadávat.

Pro vývoj softwaru je nutné zvolit vhodnou aplikaci, která s OCL umí pracovat. Nebo alespoň takovou aplikaci, ke které existuje vhodné rozšíření s OCL. Enterprise Architect například dokáže vkládat OCL do diagramů. Umí dokonce i kontrolovat syntaxi, ale nezvládá validaci a neumí generovat z OCL kód. Dobrý nástroj k OCL by validaci měl zvládnout, protože nikdo není neomylný a OCL nepatří k jednoduchým jazykům. Generování kódu už je věcí názoru. Roli může hrát to, jestli programátorovi vyhovuje struktura výstupního kódu z generátoru, která se u různých aplikací může lišit.

6.3 OCL

Problémem OCL je, že veškeré články vývojového řetězce pracující s omezeními, především pak analytici, návrháři a vývojáři, mu musí rozumět. Jinak jeho použití přestává mít význam. Pokud porovnáme prvky OCL s jejich převedenou podobou do programovacího jazyka, zjistíme další výhodu a tou je, že se sestávají z mnohem méně znaků. Je tedy lepší používat OCL než rovnou programovat kousky kódu v nějakém jazyce, které by se ve výsledku pravděpodobně stejně použily v úplně jiné podobě.

Nevýhodou OCL je to, že nezvládne vše. Není možné v něm definovat změnu hodnoty některého prvku modelu. Neumí vymezit žádné jiné funkce než vyhledávací. Umí pouze přistupovat k prvkům modelu, nastavovat jejich omezení a vyhledat prvky s nějakým omezením. Možná kdyby tyto záležitosti uměl, tak by se stal rozšířenějším. A to je jeho další nevýhodou. OCL není moc rozšířený a nepoužívá se tak často. Při hledání informací k OCL jsem narazil na trochu nadnesené porovnání k rozšířenosti OCL od neznámého autora, že jen 1 ze 100 vývojářů ovládá OCL, ale jen 1 ze 100 těch, co jej ovládají, jej skutečně v praxi používá.

OCL používají převážně vývojáři z Javy. U ostatních jazyků se až tolik nepoužívá. S tím souvisí i množství rozšíření a aplikací dávajících dohromady OCL a převod do programovacího jazyka Java. Zajímavá by mohla být představa, kdyby někdo dal oba Jazyky dohromady a vytvořil programovací jazyk s velice krátkým zápisem omezení a vyhledávacích funkcí. A to je vlastně i další nevýhodou. OCL se musí převádět do daného programovacího jazyka (ručně nebo pomocí různých generátorů kódu).

Podle množství publikací a programů to totiž vypadá, že OCL má svůj vrchol již za sebou a nějaká změna by ho mohla dostat opět do popředí. Nejvíce se o OCL mluvilo kolem roku 2010. Nyní už zájem o něj opadl. Důvodů může být více, ale pravděpodobně si uživatelé po několika odzkoušení uvědomili, že nevýhody převažují nad výhodami.

7 Závěr

Splnil jsem veškeré stanovené cíle. Program byl rozpracován do fází analýzy problému, návrhu řešení, implementace a testování. Nalezl jsem omezení, která jsem následně převedl do OCL a implementoval v programovacím jazyce Java. Vytvořené řešení by mohlo podniku pomoci řešit problém s výpočtem výrobních koeficientů. Podnik by ušetřil čas zaměstnance, který se výpočtům tohoto koeficientu zabývá, a nemusel by si pořizovat další licence jiných systémů, ani investovat do toho současného.

Před zavedením do podniku je nutné počítat s dalším testováním, zda vše funguje tak jak má, a případně vyřešit nevýhody této aplikace jedním z navrhnutých řešení uvedených v předchozí kapitole. Také by se mohla rozšířit o nové funkcionality. Například vytvoření panelu s přehledem, který by vypadal a fungoval podobně jako dosavadní dokument v Excelu, a který by mohl zaměstnancům usnadnit přechod mezi aplikacemi. Volné směny, které by bylo možné nastavit pro všechny stroje ve výrobě jedním tlačítkem (např. u celozávodní dovolené). Možnost slučovat dávky, které nejsou u sebe v kalendáři. Převedení již dokončené dávky na nedokončenou (např. pokud zaměstnanec zadal větší výrobu než měl a dávka má tedy ještě pokračovat). A mnoho dalšího. Aplikace by se mohla rozšířit o část věnovanou zásobování a objednávkám, které by se s výrobním plánem mohly propojit.

Pokud bych se nyní u stejného projektu rozhodoval, zda OCL použít nebo ne, rozhodně bych OCL nevytvářel. Pokud se o vývoj stará jediná osoba, tak použití OCL nemá žádný smysl. Vyplatilo by se to jedině, pokud bych na něco důležitého, co jsem si dobře promyslel, nechtěl zapomenout. Ale i v tomto případě bych si raději vytvořil obyčejné poznámky, ze kterých bych mohl čerpat informace. Pokud bych tento zápis pochopil jiným způsobem, tak to není žádný problém. Prostě jsem jen změnil názor po provedených krocích, které proběhly mezi těmito událostmi, a vyřešil bych ho jiným způsobem.

Dalo by se říci, že čím více osob je ve vývojářském týmu, tím preciznější by mělo být stanovení omezujících podmínek. Svádí to k tomu, že by se u velkých projektů mělo OCL používat. Ale problémem je, že všichni v týmu pak musí OCL zvládat. A tomu tak často nebývá. I kdyby všichni s OCL uměli, tak stále existuje omezení, které by od užívání OCL mohlo odrazovat. A tím je nutnost převodu OCL kódů do jiného programovacího jazyka. Kdyby se převádět nemusel nebo existovaly opravdu solidní nástroje pro snadné generování kódů z OCL do libovolného jazyka, tak bych jej doporučit mohl. Jinak ale nemohu.

8 Reference

Arlow, J. a I. Neustadt. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

Cabot, J. a M. Gogolla. *Object Constraint Language (OCL): A Definitive Guide.* Formal Methods for Model-Driven Engineering [online]. 2012, 58-90 [cit. 2016-04-24]. DOI: 10.1007/978-3-642-30982-3_3. Dostupné z: http://link.springer.com/10.1007/978-3-642-30982-3_3.

Castillo, L. *Planning, scheduling and constraint satisfaction: from theory to practice.* Washington, DC: IOS Press, 2005. ISBN 1586034847.

Cook, S. a J. Daniels. *Designing object systems: object-oriented modelling with Syntropy.* New York: Prentice Hall, 1994. Prentice Hall object-oriented series. ISBN 0-13-203860-9.

Cook, S. et al. *The Amsterdam Manifesto on OCL.* [online]. s. 115-149 [cit. 2016-04-24]. DOI: 10.1007/3-540-45669-4_7. Dostupné z: http://link.springer.com/10.1007/3-540-45669-4_7.

Documents Associated With Object Constraint Language™ (OCL™), Version 2.4. *Object Management Group* [online]. OMG [cit. 2016-04-24]. Dostupné z: http://www.omg.org/spec/OCL/2.4/.

Documents Associated With Unified Modeling Language™ (UML®), Version 2.5 *Object Management Group* [online]. OMG [cit. 2016-04-24]. Dostupné z: http://www.omg.org/spec/UML/2.5/.

Eriksson, H. a M. Penker. *Business modeling with UML: business patterns at work.* New York: John Wiley & Sons, c2000. ISBN 0-471-29551-5.

Eriksson, H. et al. *UML 2 toolkit.* Indianapolis: Wiley Pub., c2004. ISBN 04-714-6361-2.

Novibra [online]. Novibra Boskovice, s. r. o. [cit. 2016-04-24]. Dostupné z: https://www.novibra.com/cs/.

Rittgen, P. *Enterprise modeling and computing with UML.* Hershey, Philadelphia: Idea Group Pub., c2007. ISBN 1599041766.

Řepa, V. *Procesně řízená organizace.* 1. vyd. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4128-4.

Schenk, M., Wirth, S. a E. Muller. *Factory planning manual: situation-driven production facility planning.* New York: Springer, c2010. ISBN 364203635X.

SCHILDT, H. *Java 7: výukový kurz.* 1. vyd. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2.

SCHILDT, H. *Java the complete reference.* 8th ed. New York: McGraw-Hill, 2011. ISBN 9780071606318.

SOMMERVILLE, I. *Softwarové inženýrství.* 1. vyd. Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

SystemOnLine.cz - ekonomické a informační systémy v praxi [online]. CCB, 2016 [cit. 2016-04-29]. Dostupné z: http://www.systemonline.cz/.

VOLLMANN, T. *Manufacturing planning and control systems for supply chain management.* 5th ed. New York: McGraw-Hill, c2005. ISBN 007144033X.

WARMER, J. B. a T. CLARK. *Object modeling with the OCL: The rationale behind the Object Constraint Language.* New York: Springer, c2002. ISBN 978-3-540-43169-5.

WARMER, J. B. a A. G. KLEPPE. *The object constraint language: getting your models ready for MDA.* 2nd ed. Boston: Addison-Wesley, 2003. ISBN 978-0321179364.

WARMER, J. B. a A. G. KLEPPE. *The object constraint language: precise modeling with UML.* Mass: Addison Wesley Longman, c1999. ISBN 978-0201379402.

WILLINK, E. *OCL 2.5 Plans.* Valencia: 14th International Workshop on OCL and Textual Modeling Applications and Case Studies (OCL 2014) [online prezentace]. 2014, [cit. 2016-04-24]. Dostupné z: http://goo.gl/JdYCJr.

Přílohy

A Scénáře případů užití

Případ užití: Vytvořit stroj
ID: 1
Stručný popis: Zavede do systému nový stroj s označením stroje v SAP, číslem stroje, názvem produktu, výrobcem tohoto produktu, stručným popisem, jeho umístění (označení budovy a místnosti), při jakých směnách se obvykle používá a zda je v provozu a je možné k němu přiřazovat nové dávky (zda je aktivní).
Hlavní aktéři: Administrátor
Vedlejší aktéři: Mistr
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor klikne na tlačítko s vytvářením nového stroje 2. Systém zobrazí formulář s vytvářením nového stroje 3. Administrátor zadá veškeré údaje nového stroje 4. Administrátor potvrdí údaje 5. Pokud v systému neexistuje stroj se stejným označením, pak 5.1 Systém napíše uživateli potvrzení o vytvoření nového stroje 6. nebo 6.1 Systém napíše uživateli, že stroj s daným označením již existuje 7. Systém přidá do oznamovací části všem administrátorům a mistrovi oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 17: Scénář: vytvořit stroj

Případ užití: Vyřadit stroj
ID: 2
Stručný popis: Administrátor stroj nenávratně odstraní ze systému i se všemi jeho výrobními dávkami.
Hlavní aktéři: Administrátor
Vedlejší aktéři: Mistr
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere konkrétní stroj a vyvolá akci na zrušení stroje 2. Systém vygeneruje upozornění 3. Systém vyhledá veškeré dávky a jejich součásti, které byly přiřazeny k tomuto stroji, a odstraní je 4. Systém odstraní veškerá přiřazení k tomuto stroji (mistrů a dělníků) 5. Systém odstraní záznam o tomto stroji 6. Systém potvrdí uživateli, že je akce u konce 7. Systém přidá do oznamovací části administrátorů a mistrovi tohoto stroje oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 18: Scénář: vyřadit stroj

Případ užití: Přihlásit se
ID: 3
Stručný popis: Uživatel se přihlásí do systému s příslušnými právy a zobrazí se mu nabídka odpovídající jeho přístupovým právům
Hlavní aktéři: Uživatel (administrátor, mistr, dělník)
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel spustí aplikaci a zadá svoje přihlašovací údaje (jméno a heslo) 2. Systém ověří, zda se zadané hodnoty shodují s některým uživatelem. Pokud ano, pak: 2.1 Otevře aplikaci s nastavením pro roli, kterou uživatel má 3. Nebo: 3.1 Upozorní uživatele na neexistujícího uživatele nebo špatně zadané heslo
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 19: Scénář: přihlásit se

Případ užití: Odhlásit se
ID: 4
Stručný popis: Odhlásí uživatele ze systému a umožní tak přihlášení jiného uživatele
Hlavní aktéři: Uživatel (administrátor, mistr, dělník)
Vedlejší aktéři: žádní
Vstupní podmínky: Uživatel je přihlášen
Hlavní scénář: 1. Uživatel klikne na tlačítko pro odhlášení 2. Systém zavře aplikaci otevřenou pro konkrétního uživatele 3. Systém zobrazí přihlašovací obrazovku
Výstupní podmínky: žádné
Alternativní scénáře: Automatické odhlášení

Obrázek 20: Scénář: odhlásit se

Alternativní případ užití: Odhlásit se:Automatické odhlášení
ID: 4.1
Stručný popis: Odhlásí uživatele ze systému a umožní tak přihlášení jiného uživatele
Hlavní aktéři: Čas
Vedlejší aktéři: Uživatel (administrátor, mistr, dělník)
Vstupní podmínky: Uživatel je přihlášen a byl delší dobu nečinný
Hlavní scénář: 1. Alternativní scénář se spouští po uplynutí doby, po které uživatel nic nevykonal (pohyb myší) 2. Alternativní scénář pokračuje krokem 2 hlavního scénáře
Výstupní podmínky: žádné

Obrázek 21: Scénář: automatické odhlášení

Případ užití: Vytvořit výrobní dávku
ID: 5
Stručný popis: Vytvoří u zvoleného stroje výrobní dávku (co se bude vyrábět, v jakém množství, označení dávky, kolik dní je na výrobu (nebo počet směn), ID části dávek kratších jak 8 hodin, jejichž plnění se v systému sledovat nebude, ID materiálu). Uživatel pak může ručně zadat plán na jednotlivé směny nebo může nechat systém, ať toto množství rovnoměrně rozloží.
Hlavní aktéři: Mistr, Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere stroj, u kterého chce vytvořit výrobní dávku, a zvolí tuto akci 2. Systém zobrazí formulář s tvorbou dávek 3. Mistr nebo administrátor vyplní veškeré položky 4. Mistr nebo administrátor zvolí formu přidělení plánu na směny (ručně nebo automaticky) 5. Pokud bylo zvoleno ruční přidělování, pak: 5.1 Systém zobrazí formulář s plánem směn 5.2 Mistr nebo administrátor zadají plán výroby pro jednotlivé směny a potvrdí ukončení zadávání 5.3 Systém zkontroluje plánované množství, pokud sedí, pak: 5.3.1 Systém vytvoří novou dávku v systému a u daného stroje ji zařadí za veškeré dříve vytvořené dávky 5.3.2 Systém informuje mistra nebo administrátora o úspěšném vytvoření výrobní dávky 5.4 Nebo: 5.4.1 Systém upozorní uživatele na chybu a umožní mu vrátit se do bodu 5.1 6. Nebo: 6.1 Systém rovnoměrně rozpočítá plán dávky mezi směny 6.2 Systém informuje mistra nebo administrátora o úspěšném vytvoření výrobní dávky 7. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 22: Scénář: vytvořit výrobní dávku

Případ užití: Zrušit výrobní dávku nebo její část
ID: 6
Stručný popis: Zruší ze systému výrobní dávku nebo část výrobní dávky. Odstraněná dávka nebo její část automaticky posouvá veškeré dávky a části dávek, které se nacházely za tímto blokem.
Hlavní aktéři: Administrátor, Mistr
Vedlejší aktéři: žádní
Vstupní podmínky: Vybraná dávka nebo její část ještě nebyla zcela dokončena.
Hlavní scénář: 1. Administrátor nebo mistr vyhledají konkrétní dávku nebo její část a stisknou tlačítko na odstranění 2. Systém vyzve k potvrzení smazání 3. Administrátor nebo mistr potvrdí smazání 4. Pokud jde o smazání části dávky, pak: 4.1 Systém smaže část zvolené dávky, která ještě neproběhla (nebyly zadány skutečně vyrobené kusy), a informuje uživatele o úspěšném provedení akce 5. Nebo: 5.1 Systém smaže danou dávku (pouze tu část, která ještě neproběhla = nebyly zadány skutečně vyrobené kusy) a informuje uživatele o úspěšném provedení akce 6. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 23: Scénář: zrušit výrobní dávku nebo její část

Případ užití: Změnit údaje výrobní dávky
ID: 7
Stručný popis: V systému se změní údaje o dávce (co se bude vyrábět, v jakém množství, označení dávky, kolik dní je na výrobu (nebo počet směn), ID části dávek kratších jak 8 hodin, ID materiálu). Změna množství se znovu rozpočítá mezi dny, které ještě nebyly dokončeny. Změna v počtu dní na výrobu pak přidá nebo odebere směny od konce dávky a posouvá začátek dávek následujících po této.
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: Část dávky ještě nesmí být dokončena.
Hlavní scénář: 1. Mistr nebo administrátor vybere dávku a klikne na volbu úprav 2. Systém zobrazí formulář, ve kterém uživatel může provádět změny na původních hodnotách 3. Mistr nebo administrátor provede dané změny a potvrdí je 4. Pokud uživatel změní množství, pak: 4.1 Systém se zeptá, zda rovnoměrně rozložit plán mezi všechny směny nebo změnu množství nechat na uživateli 4.2 Mistr nebo administrátor zvolí jednu z voleb 4.3 Pokud zvolí ruční, pak: 4.3.1 Systém zobrazí formulář s plánem pro jednotlivé směny s původními hodnotami 4.3.2 Mistr nebo administrátor změní údaje, které chce 4.3.3 Systém zkontroluje, zda suma hodnot odpovídá celkovému plánovanému množství. Pokud ne, tak upozorní uživatele a umožní mu návrat do kroku 4.3.1 4.4 Nebo: 4.4.1 Systém rovnoměrně rozpočítá hodnoty mezi směny, které ještě nebyly uskutečněny (nebyla k nim vydána informace o skutečně vyrobeném množství) 5. Systém informuje mistra, administrátora o úspěšném provedení změn 6. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 24: Scénář: změnit údaje výrobní dávky

Případ užití: Změnit plán jednotlivých směn
ID: 8
Stručný popis: Změní plán jednotlivých směn
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere dávku a zvolí možnost na změnu plánu směn 2. Systém zobrazí příslušný formulář pro danou dávku s původními hodnotami 3. Mistr nebo administrátor změní plán pro jednotlivé směny 4. Mistr nebo administrátor potvrdí změny 5. Systém zkontroluje sumu plánů jednotlivých směn se sumou celé dávky, pokud se liší, pak: 5.1 Systém upozorní uživatele, že hodnoty nesedí a vrátí ho do kroku 3 6. Systém potvrdí uživateli, že změny byly úspěšně provedeny 7. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 25: Scénář: změnit plán jednotlivých směn

Případ užití: Rozdělit výrobní dávku na více částí
ID: 9
Stručný popis: Rozdělí výrobní dávku na několik samostatných částí, které se mohou chovat jako výrobní dávky, ale se zachováním údajů původní dávky a propojením zpět na ní. Jediným parametrem navíc je ID části dávky.
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vyberou výrobní dávku a zvolí možnost rozdělení výrobní dávky 2. Systém zobrazí příslušný formulář s aktuálním rozložením směn 3. Mistr nebo administrátor určí rozmezí 2 směn, kde se dávka rozdělí 4. Systém vyzve k zadání ID pro jednotlivé části 5. Mistr nebo administrátor zadá ID pro každou část 6. Systém rozdělí dávku na 2 nové části 7. Systém zobrazí uživateli potvrzení o úspěšném provedení změn 8. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 26: Scénář: rozdělit výrobní dávku na více částí

Případ užití: Sloučit části výrobní dávky
ID: 10
Stručný popis: Sloučí části jedné výrobní dávky zpět dohromady.
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere konkrétní dávku a vybere možnost se sloučením částí 2. Systém vrátí formulář s aktuálním rozdělením celé dávky 3. Mistr nebo administrátor vybere části, které chce spojit a potvrdí akci 4. Systém zkontroluje, zda jsou části v pořadí u sebe (nejsou přerušeny jinou dávkou nebo částí), pokud ano, pak: 4.1 Upozorní uživatele na tuto skutečnost a nabídne mu možnost vrátit se zpět ke kroku 2 5. Systém sloučí části 6. Uživatel zadá ID nově vzniklé části 7. Systém informuje mistra nebo administrátora o úspěšném sloučení skupin 8. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 27: Scénář: sloučit části výrobní dávky

Případ užití: Změnit pořadí výrobních dávek a jejich částí
ID: 11
Stručný popis: Přesune výrobní dávku nebo její část v pořadí, v jakém čekají na výrobu na daném stroji. Přesunout lze až k dávce nebo části, která ještě není zpracovávána (nebyl v ní vyroben žádný výrobek). Pokud je nutné výrobu na stroji náhle ukončit uprostřed dávky nebo její části (výjimečná situace), musí uživatel tuto dávku nebo její část rozdělit na 2 části, aby následně bylo možné přesun provést.
Hlavní aktéři: Mistr, Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere stroj a zvolí možnost změny pořadí 2. Systém zobrazí formulář, ve kterém budou seřazeny jednotlivé dávky a nezapočaté části dávek, u kterých ještě nebyla započata výroba 3. Uživatel změní pořadí dávek a potvrdí změny 4. Systém přesune pořadí daných dávek 5. Systém potvrdí uživateli změnu 6. Systém přidá do oznamovací části mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 28: Scénář: změnit pořadí dávek a jejich částí

Případ užití: Přiřadit dělníka ke stroji
ID: 12
Stručný popis: Přiřadí k určitému stroji dělníka, který na něm bude vykonávat práci, aby mohl k tomuto stroji zapisovat jím vyrobené množství.
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: Dělník
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere konkrétní stroj a zvolí volbu na změnu přiřazení pracovníků 2. Systém zobrazí formulář pro přidělení pracovníků ke stroji 3. Mistr nebo administrátor vybere pracovníka ze skupiny nepřidělených k tomuto stroji a klikne na přidělit ke stroji 4. Systém provede přidělení pracovníka ke stroji 5. Systém potvrdí mistrovi nebo administrátorovi, že akce byla úspěšně dokončena 6. Systém přidá do oznamovací části dělníka a mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 29: Scénář: přiřadit dělníka ke stroji

Případ užití: Zrušit přidělení dělníka ke stroji
ID: 13
Stručný popis: Odstraní přiřazení dělníka k určitému stroji
Hlavní aktéři: Mistr, administrátor
Vedlejší aktéři: Dělník
Vstupní podmínky: žádné
Hlavní scénář: 1. Mistr nebo administrátor vybere konkrétní stroj a zvolí volbu na změnu přiřazení pracovníků 2. Systém zobrazí formulář pro odstranění pracovníků od stroje 3. Mistr nebo administrátor vybere pracovníka ze skupiny přidělených k tomuto stroji a klikne na odebrat 4. Systém provede odstranění přiřazení pracovníka k tomuto stroji 5. Systém potvrdí mistrovi nebo administrátorovi, že akce byla úspěšně dokončena 6. Systém přidá do oznamovací části dělníka a mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 30: Scénář: zrušit přidělení dělníka ke stroji

Případ užití: Vytvořit zaměstnance a jeho přihlašovací údaje
ID: 14
Stručný popis: Vytvoří v systému nového zaměstnance s atributy jméno, příjmení, pracovní zařazení
Hlavní aktéři: Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor otevře formulář pro tvorbu nových zaměstnanců 2. Systém vrátí tento formulář 3. Administrátor zadá pozici zaměstnance (dělník, mistr, administrátor) 4. Administrátor zadá informace o zaměstnanci 5. Administrátor vymyslí uživateli jméno, vygeneruje heslo a potvrdí zadané údaje 6. Systém zkontroluje jedinečnost uživatelského jména v systému, pokud není použito, pak: 6.1 Systém vytvoří v systému zaměstnanci a vytvoří mu přihlašovací účet 6.2 Systém potvrdí administrátorovi vytvoření uživatele spolu s jeho heslem k přihlášení 6.3 Administrátor předá uživateli přihlašovací údaje 7. Jinak: 7.1 Systém upozorní administrátora na špatné uživatelské jméno a vrátí jej do kroku 4 8. Systém přidá do oznamovací části administrátorů oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 31: Scénář: vytvořit zaměstnance a jeho přihlašovací údaje

Případ užití: Změnit údaje zaměstnance
ID: 15
Stručný popis: Administrátor může provést změnu údajů u zaměstnance, které se předtím zadávali u jeho vytváření. Typicky jeho pozice, pracovní zařazení, příjmení
Hlavní aktéři: Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere zaměstnance a klikne na provádění změn uživatele 2. Systém zobrazí formulář s původním nastavením daného zaměstnance 3. Administrátor pozmění příslušné údaje a potvrdí změny 4. Systém provede změny 5. Systém potvrdí uživateli úspěšné dokončení změn 6. Systém přidá do oznamovací části administrátorů oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 32: Scénář: změnit údaje zaměstnance

Případ užití: Změnit přihlašovací údaje
ID: 16
Stručný popis: Administrátor může změnit přihlašovací jméno nebo vygenerovat nové heslo
Hlavní aktéři: Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere zaměstnance a klikne na změnu přihlašovacích údajů 2. Systém zobrazí formulář s původními uživatelskými údaji 3. Administrátor změní uživatelské jméno a zároveň vygeneruje nové heslo nebo pouze vygeneruje nové heslo 4. Administrátor potvrdí změny 5. Systém provede změny a informuje administrátora o úspěšném provedení změn 6. Administrátor sdělí uživateli nové údaje 7. Systém přidá do oznamovací části administrátorů oznámení o změně přihlašovacího jména
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 33: Scénář: změnit přihlašovací údaje

Případ užití: Zrušit zaměstnance
ID: 17
Stručný popis: Odstraní ze systému zaměstnance a jeho přihlašovací údaje
Hlavní aktéři: Administrátor
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere zaměstnance a zvolí nabídku zrušení zaměstnance 2. Systém zobrazí upozornění, zda tuto akci má skutečně provést 3. Pokud administrátor akci potvrdí, pak: 3.1 Systém zruší veškerá přiřazení zaměstnance ke strojům 3.2 Systém zruší ze systému přihlašovací údaje 3.3 Systém zruší ze systému zaměstnance 3.4 Systém potvrdí uživateli provedení akce
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 34: Scénář: zrušit zaměstnance

Případ užití: Zapsat počet skutečně vyrobených kusů
ID: 18
Stručný popis: Uživatel systému může zadat, kolik toho za směnu vyrobil. Dělník může vyplňovat údaje pouze ke strojům, ke kterým je přiřazen. Mistr nebo administrátor může v případě nutnosti zadat údaje za dělníka.
Hlavní aktéři: Uživatel
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel vybere stroj a směnu, kterou chce vyplnit 2. Uživatel vyplní údaje o výrobě a potvrdí zadávání 3. Systém informuje uživatele o úspěšném zadání výroby 4. Systém přidá do oznamovací části mistrů a dělníků přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 35: Scénář: zapsat počet skutečně vyrobených kusů

Případ užití: Změnit počet skutečně vyrobených kusů
ID: 19
Stručný popis: Umožňuje změnit dříve zadaný počet kusů jiným zaměstnancem.
Hlavní aktéři: Uživatel
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel vybere směnu, u které chce změnit vyrobené množství 2. Uživatel změní množství vyrobených výrobků a potvrdí změnu 3. Systém provede změny a informuje uživatele 4. Systém přidá do oznamovací části mistrů a dělníků přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 36: Scénář: změnit počet skutečně vyrobených kusů

Případ užití: Přiřazení mistra ke stroji
ID: 20
Stručný popis: Administrátor přiřadí mistra ke konkrétnímu stroji, který bude mít na starost.
Hlavní aktéři: Administrátor
Vedlejší aktéři: Mistr
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere konkrétní stroj a zvolí u něj možnost změnit mistra stroje 2. Systém zobrazí formulář se všemi existujícími mistry a mistry přiřazených ke stroji 3. Administrátor přesune konkrétního mistra ke stroji 4. Administrátor potvrdí změny 5. Systém provede změnu přiřazení mistrů u daného stroje a informuje uživatele o dokončení akce 6. Systém přidá do oznamovací části administrátorů, mistrů a dělníků přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 37: Scénář: přiřazení mistra ke stroji

Případ užití: Promazat oznámení
ID: 21
Stručný popis: Po přihlášení do aplikaci se promažou veškerá oznámení starší 60 dnů.
Hlavní aktéři: Zaměstnanec
Vedlejší aktéři: Čas
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel se přihlásí do aplikace 2. Systém zjistí aktuální datum a smaže všechna oznámení starší 60 dnů
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 38: Scénář: promazat oznámení

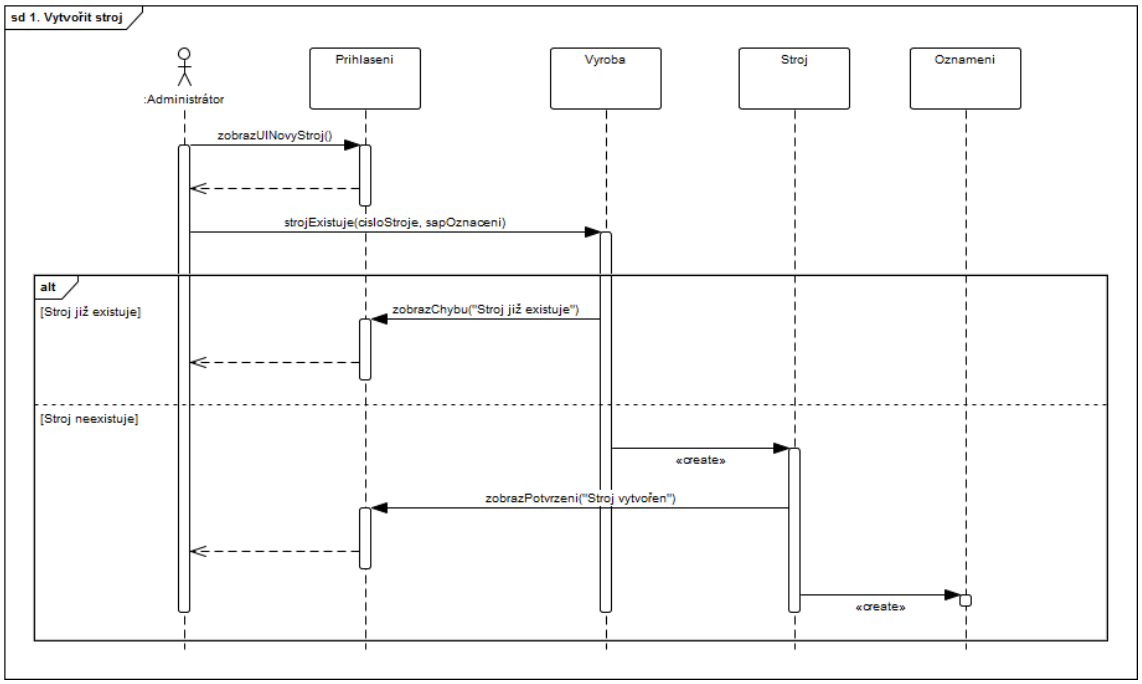
Případ užití: Změna hesla
ID: 22
Stručný popis: Uživatel si po přihlášení do aplikace může změnit svoje heslo za jiné.
Hlavní aktéři: Uživatel
Vedlejší aktéři: žádní
Vstupní podmínky: žádné
Hlavní scénář: 1. Uživatel si otevře nabídku se změnou hesla 2. Systém zobrazí formulář se změnou hesla 3. Uživatel zadá staré heslo, nové heslo, a nové heslo ještě jednou a potvrdí zadání 4. Systém zkontroluje, zda heslo vyhovuje podmínkám a je v obou políčkách stejné a zda je původní heslo správné. Pokud ano, pak: 4.1 Systém změní přístupové heslo 4.2 Systém oznámí uživateli, že změna hesla proběhla úspěšně 4.3 Systém odhlásí uživatele 5. Jinak: 5.1 Systém oznámí uživateli, že heslo neodpovídá pravidlům a vrátí ho ke kroku 2
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 39: Scénář: změna hesla

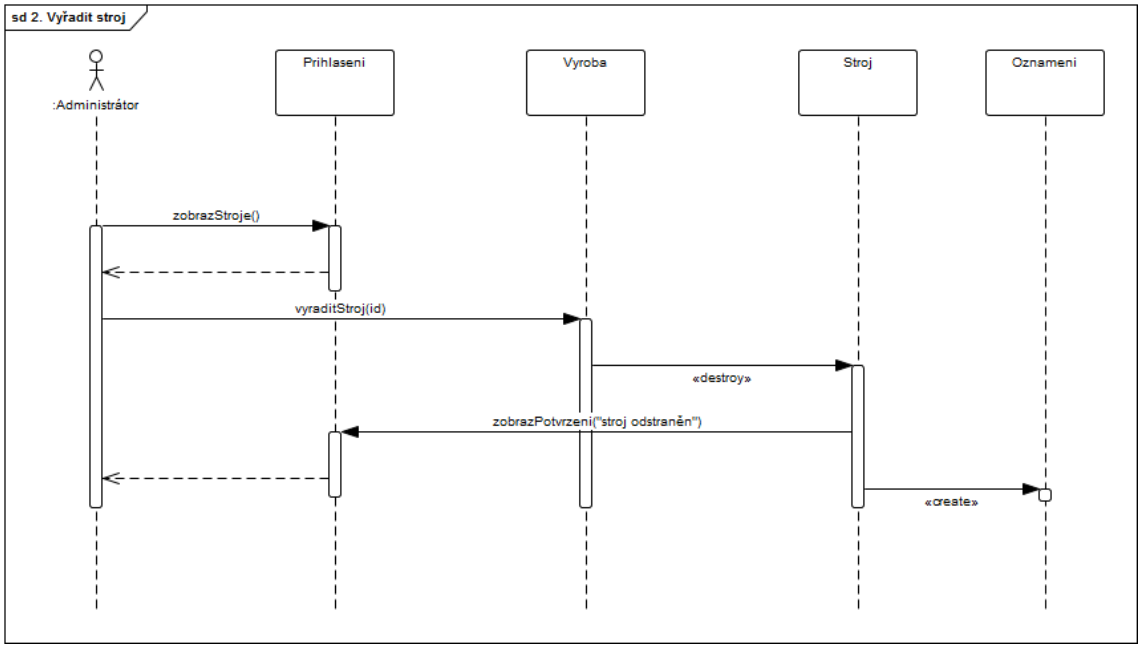
Případ užití: Zrušit přidělení mistra ke stroji
ID: 23
Stručný popis: Odstraní přiřazení mistra ke stroji
Hlavní aktéři: Administrátor
Vedlejší aktéři: Mistr
Vstupní podmínky: žádné
Hlavní scénář: 1. Administrátor vybere konkrétní stroj a zvolí volbu na změnu přiřazení mistrů 2. Systém zobrazí formulář pro odstranění mistrů od stroje 3. Administrátor vybere mistra ze skupiny přidělených k tomuto stroji a klikne na odebrat 4. Systém provede odstranění přiřazení mistra k tomuto stroji 5. Systém potvrdí administrátorovi, že akce byla úspěšně dokončena 6. Systém přidá do oznamovací části administrátorů, dělníků a mistrů přiřazených k tomuto stroji oznámení o této akci
Výstupní podmínky: žádné
Alternativní scénáře: žádné

Obrázek 40: Scénář: zrušit přidělení mistra ke stroji

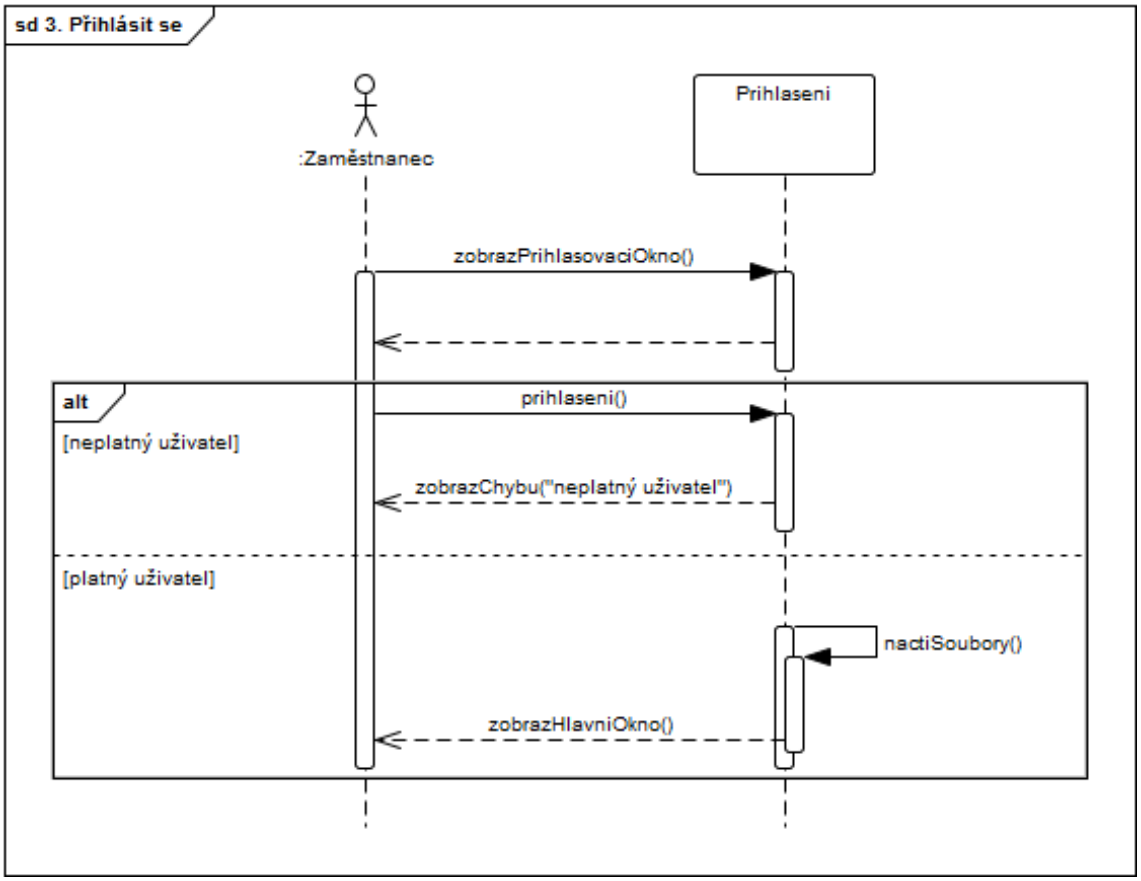
B Sekvenční diagramy



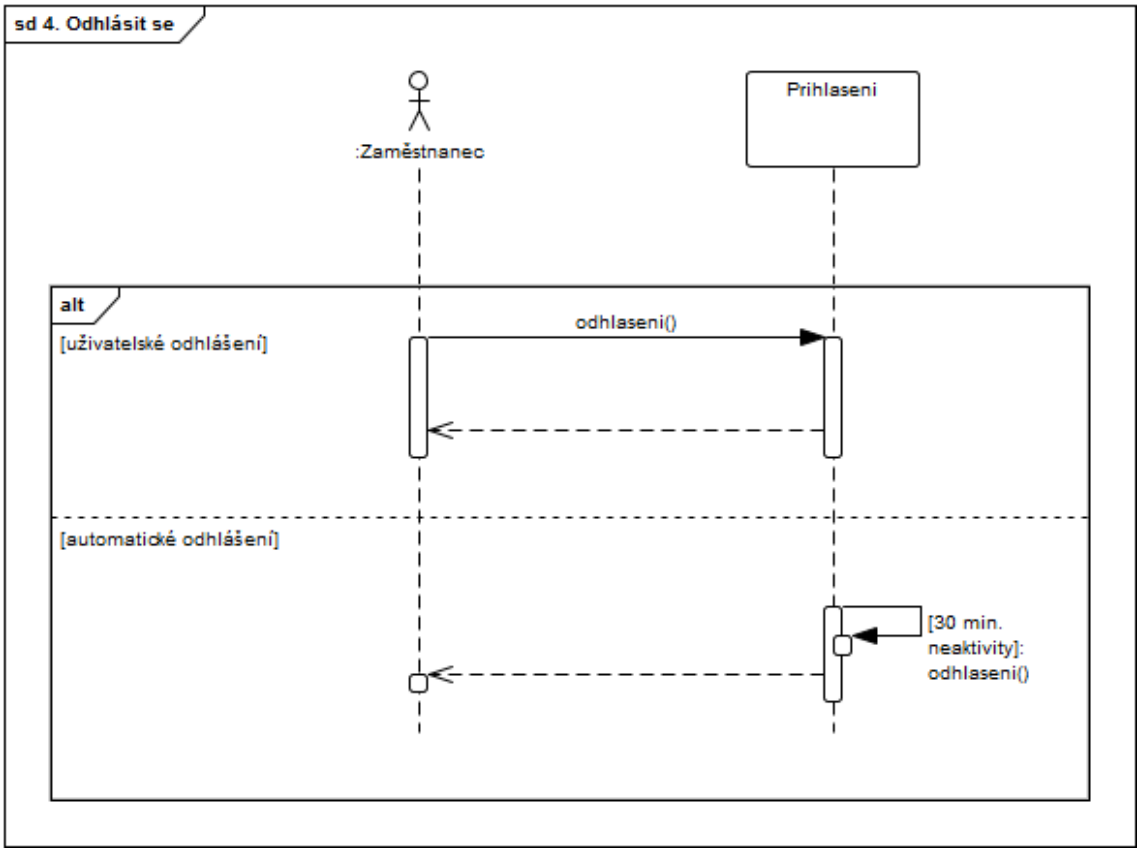
Obrázek 41: Sekvenční diagram: vytvořit stroj



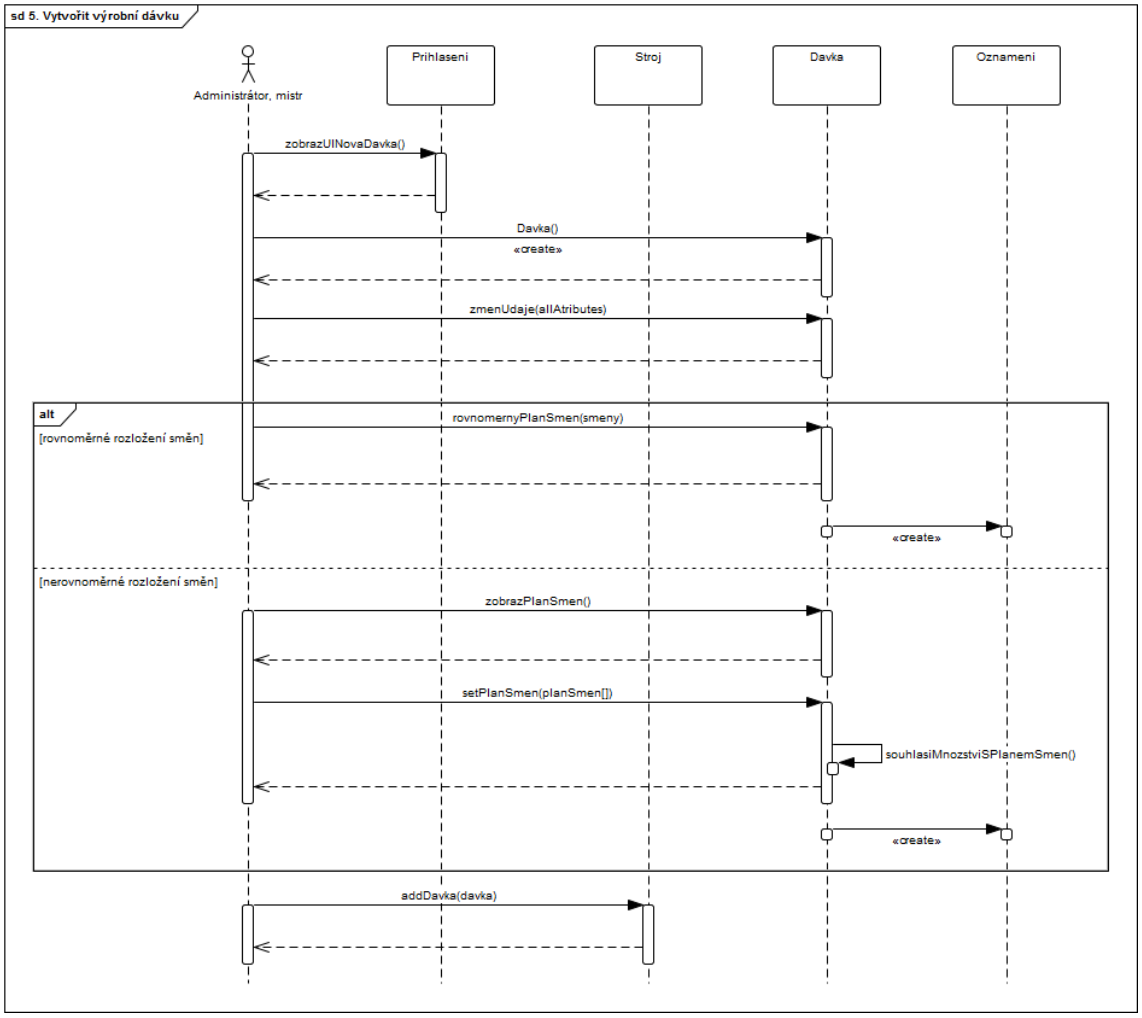
Obrázek 42: Sekvenční diagram: vyřadit stroj



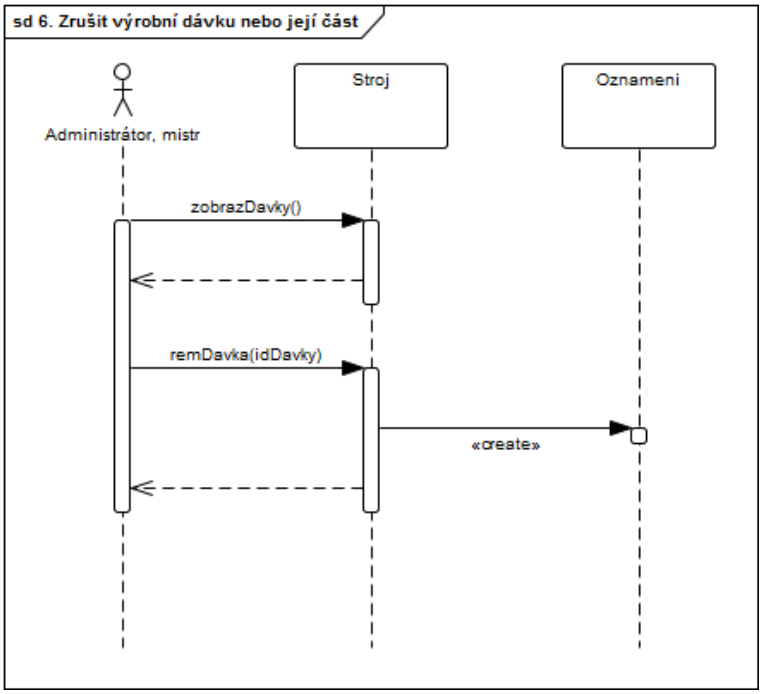
Obrázek 43: Sekvenční diagram: přihlásit se



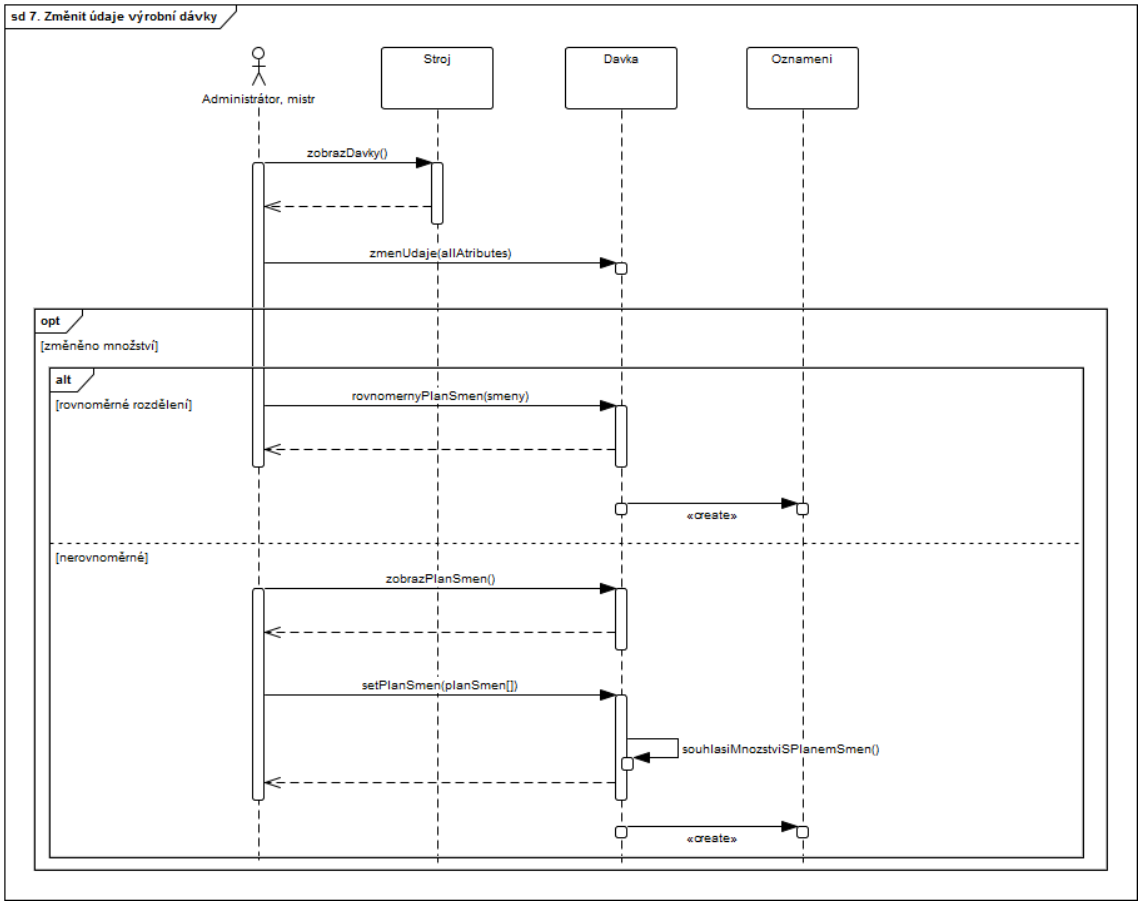
Obrázek 44: Sekvenční diagram: odhlásit se



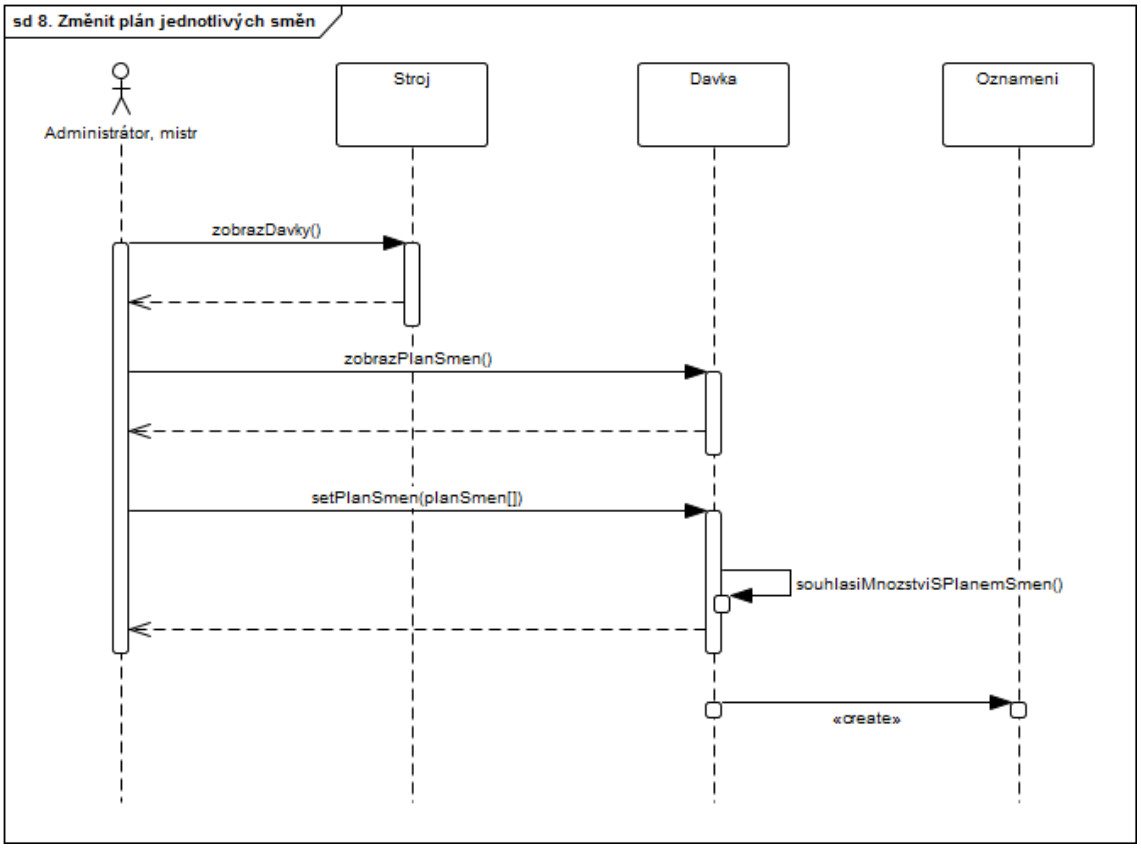
Obrázek 45: Sekvenční diagram: vytvořit výrobní dávku



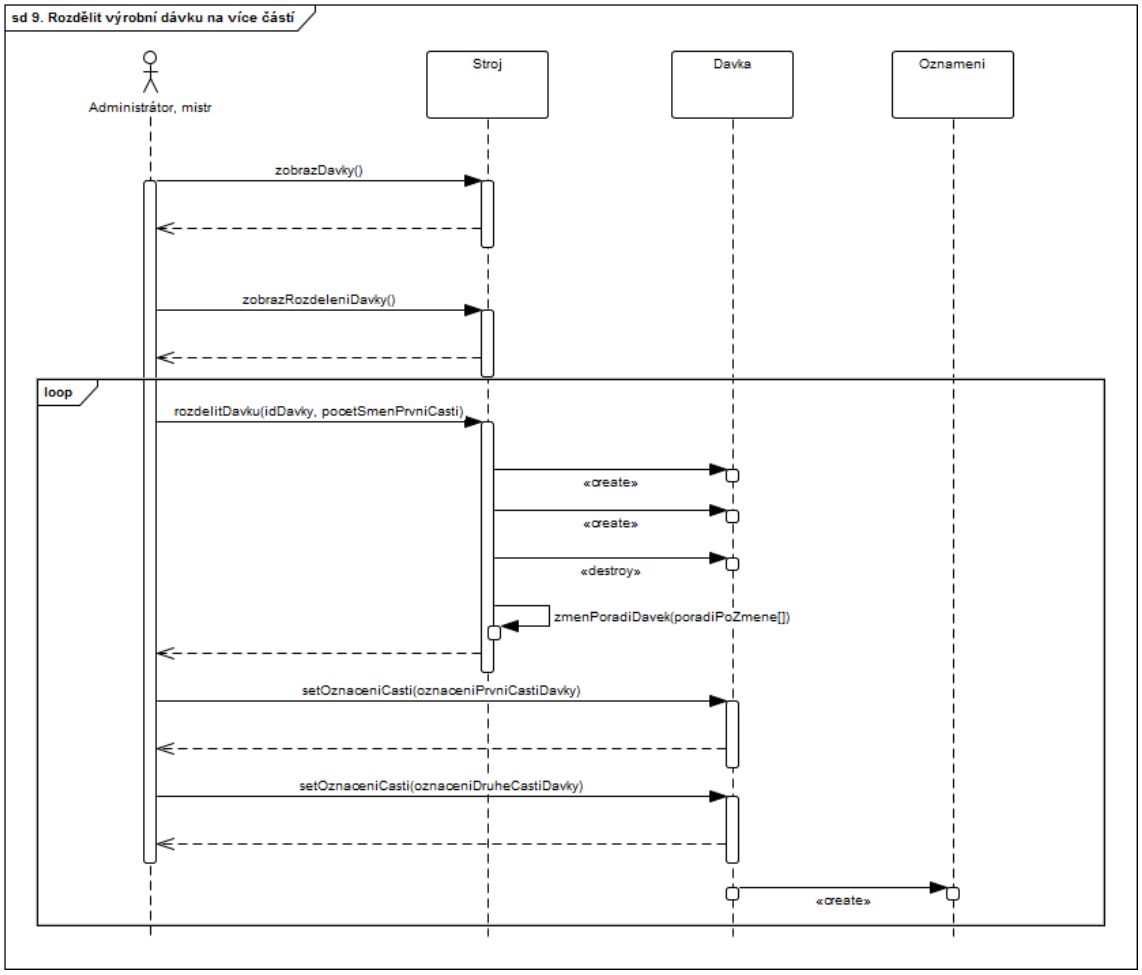
Obrázek 46: Sekvenční diagram: zrušit výrobní dávku nebo její část



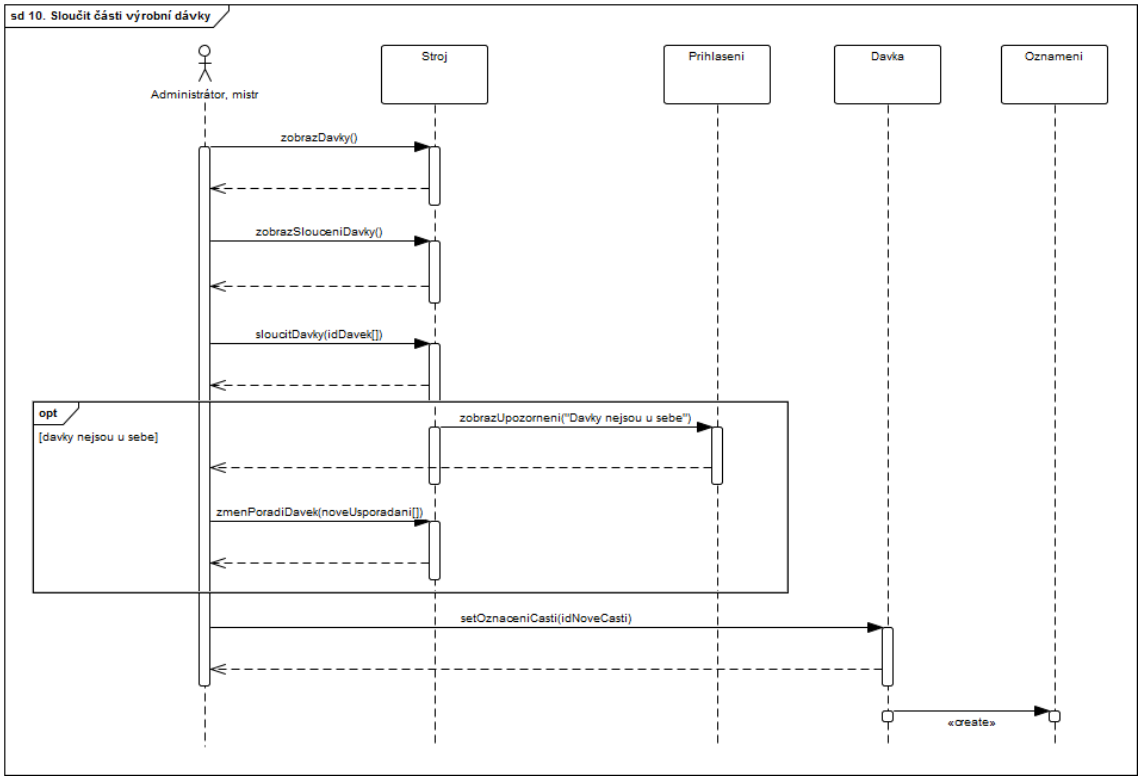
Obrázek 47: Sekvenční diagram: změnit údaje výrobní dávky



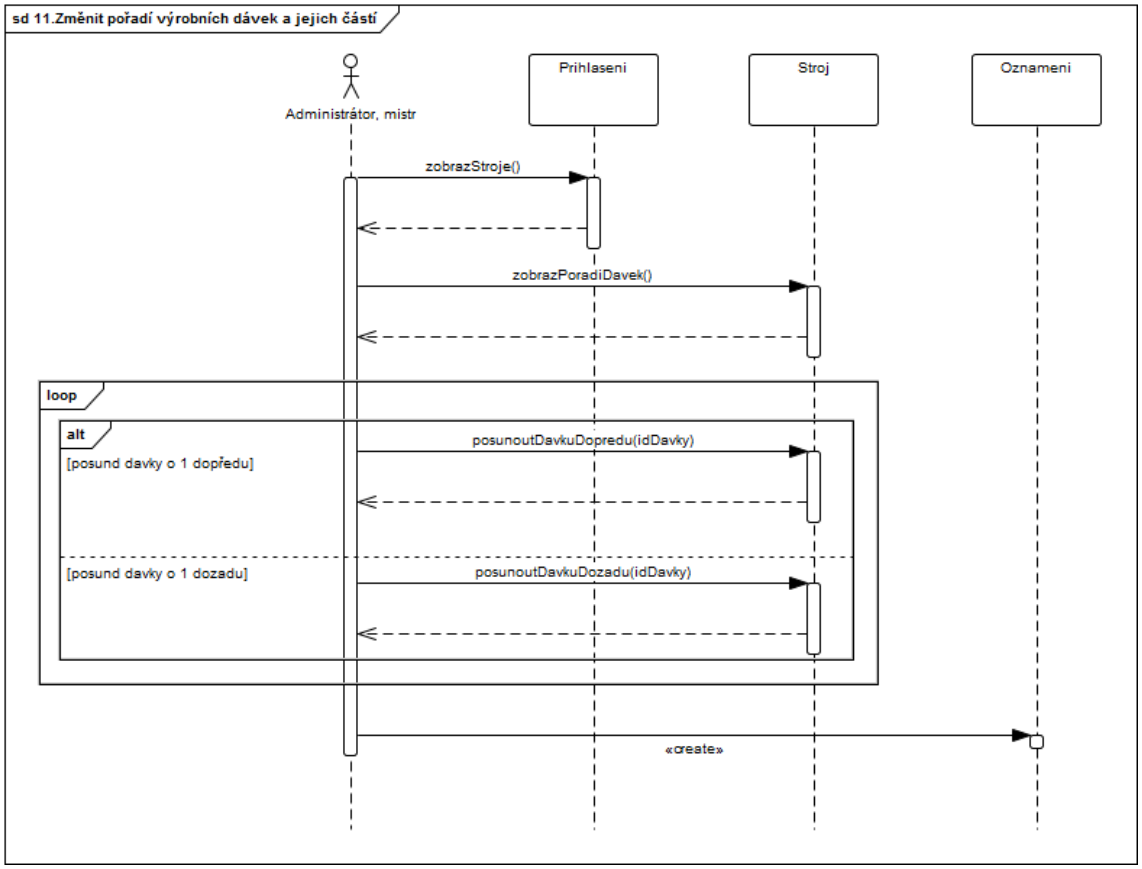
Obrázek 48: Sekvenční diagram: změnit plán jednotlivých směn



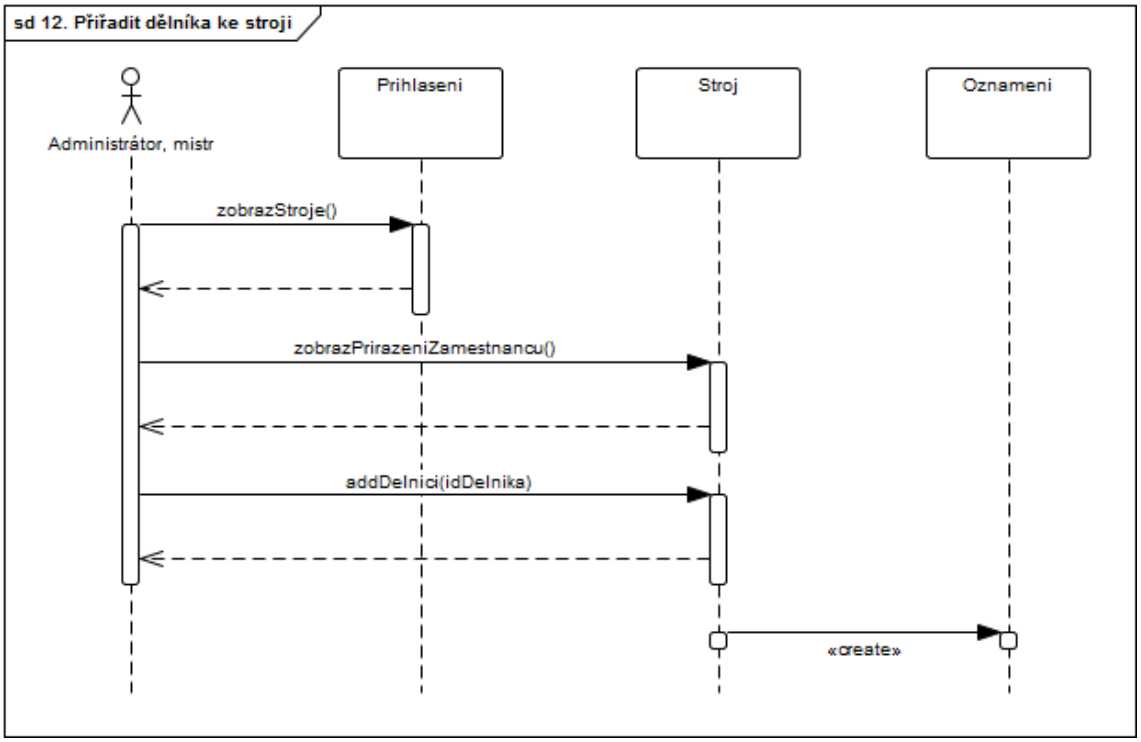
Obrázek 49: Sekvenční diagram: rozdělit výrobní dávku na více částí



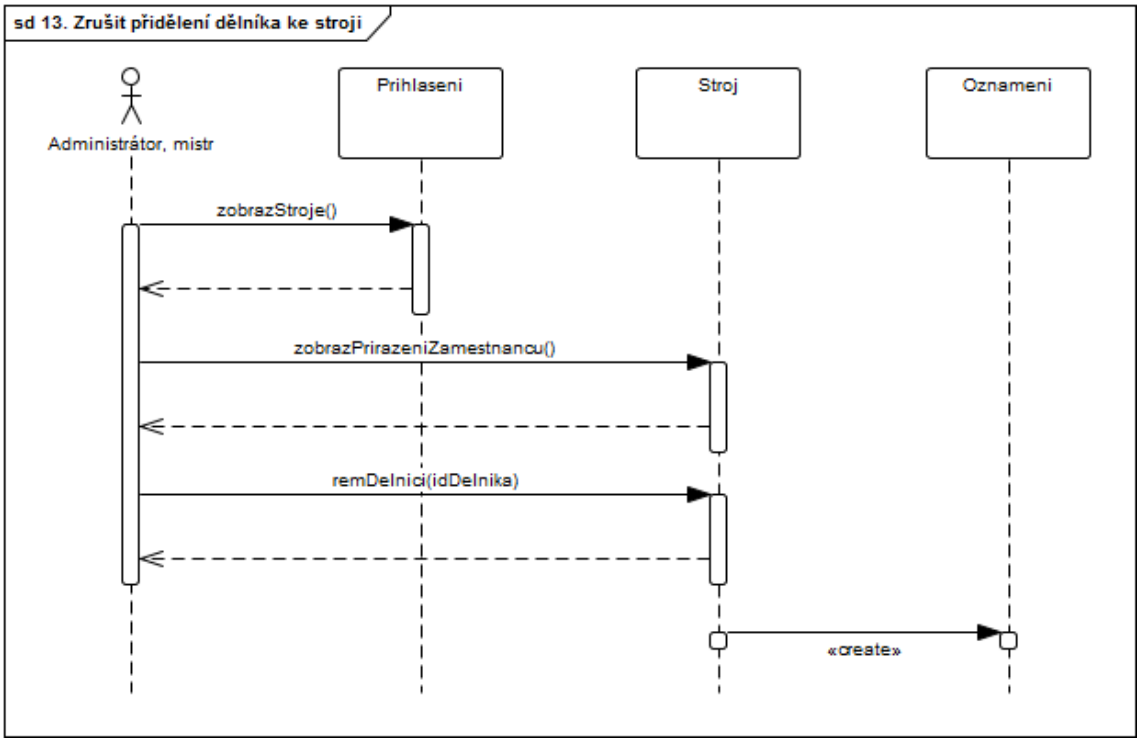
Obrázek 50: Sekvenční diagram: sloučit části výrobní dávky



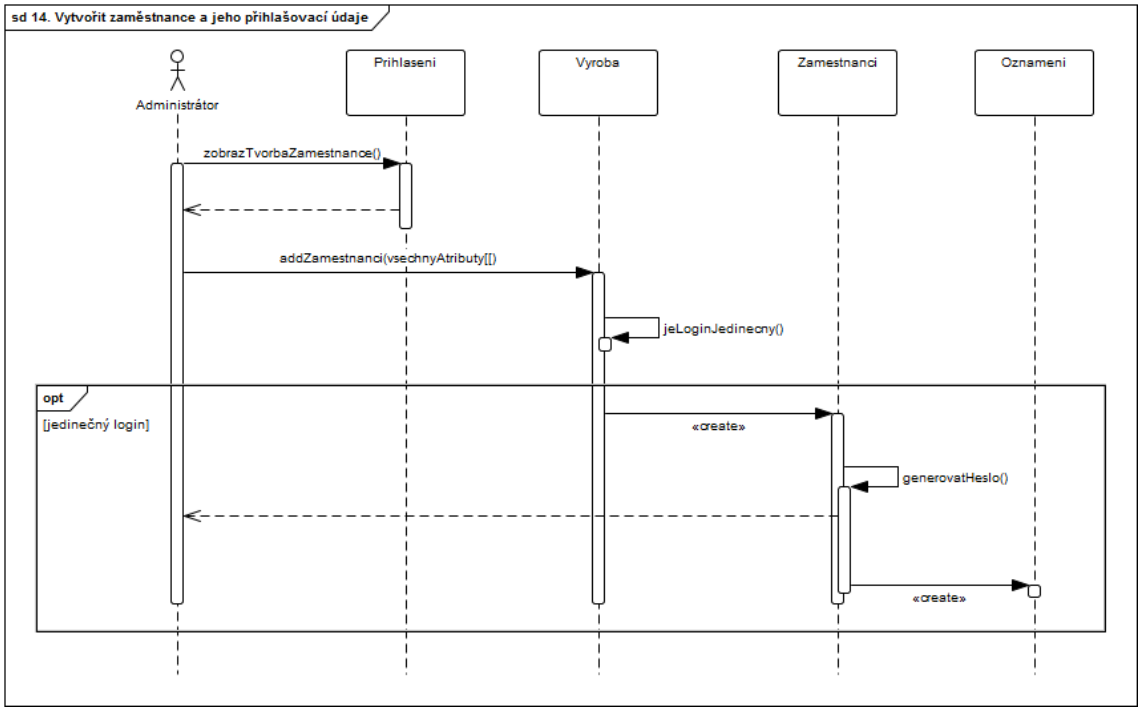
Obrázek 51: Sekvenční diagram: změnit pořadí výrobních dávek a jejich částí



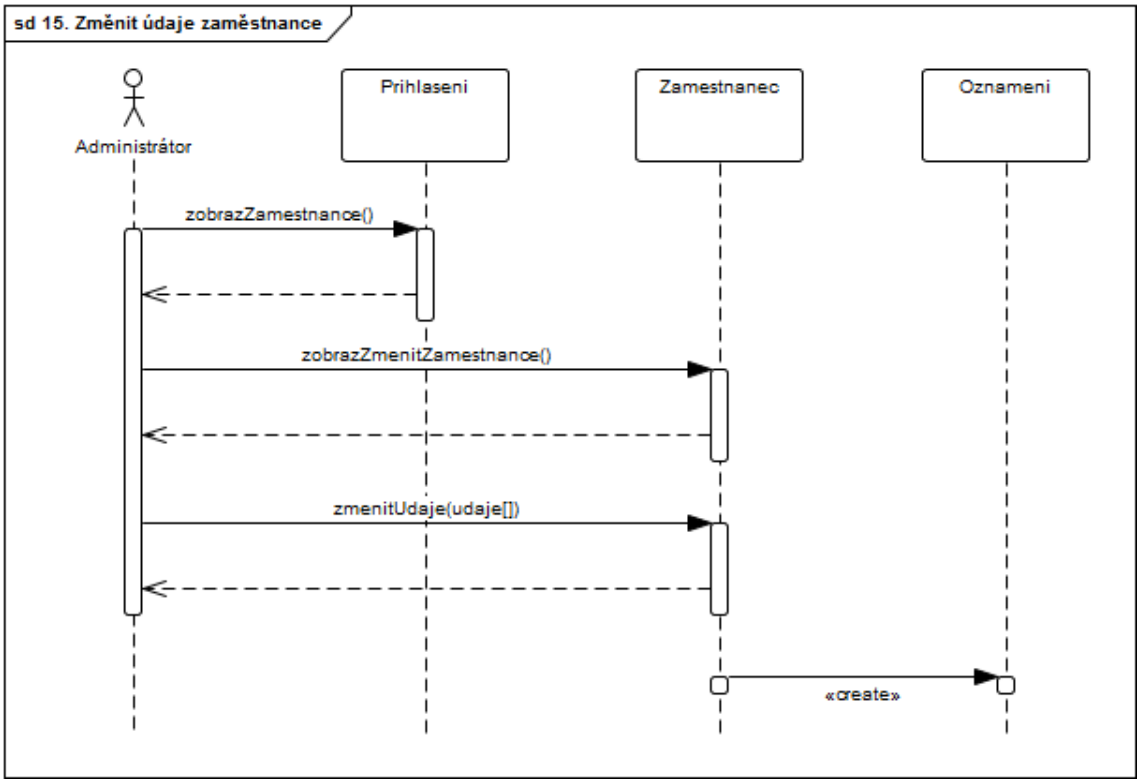
Obrázek 52: Sekvenční diagram: přiřadit dělníka ke stroji



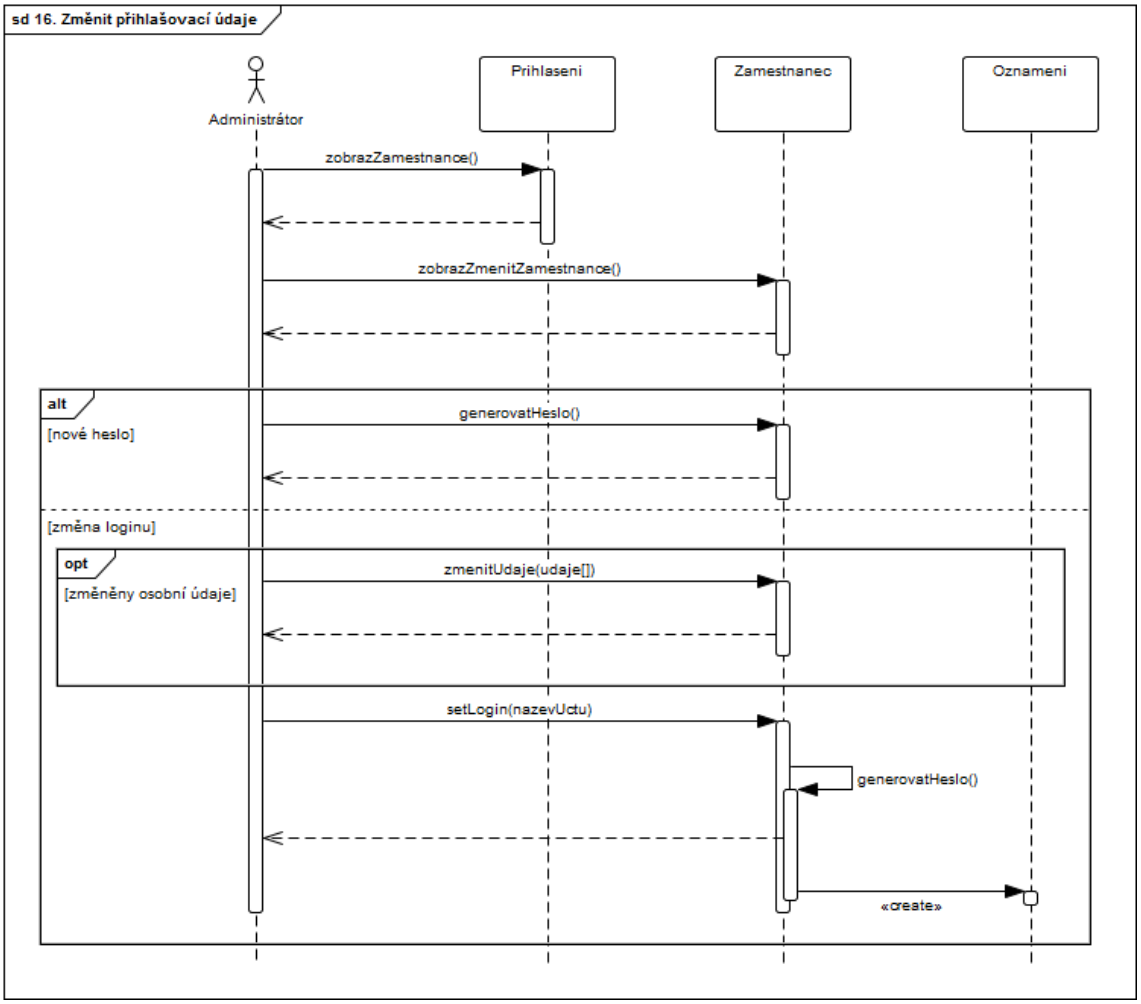
Obrázek 53: Sekvenční diagram: zrušit přidělení dělníka ke stroji



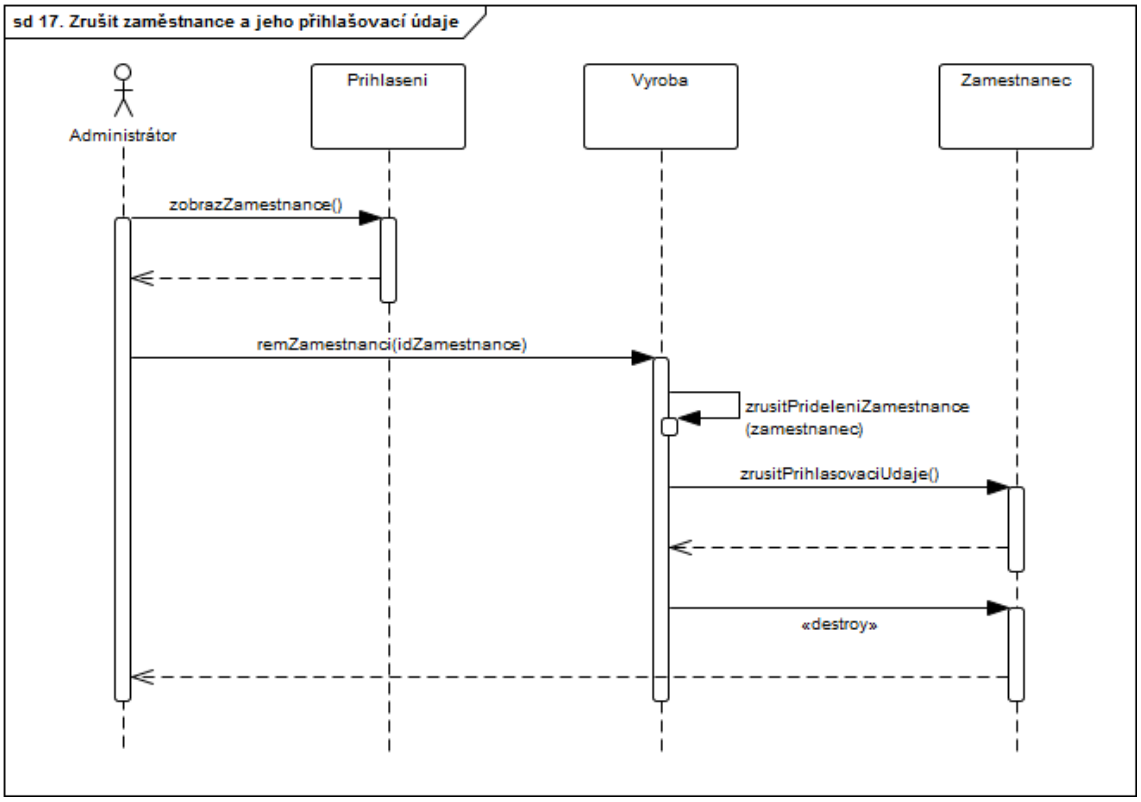
Obrázek 54: Sekvenční diagram: vytvořit zaměstnance a jeho přihlašovací údaje



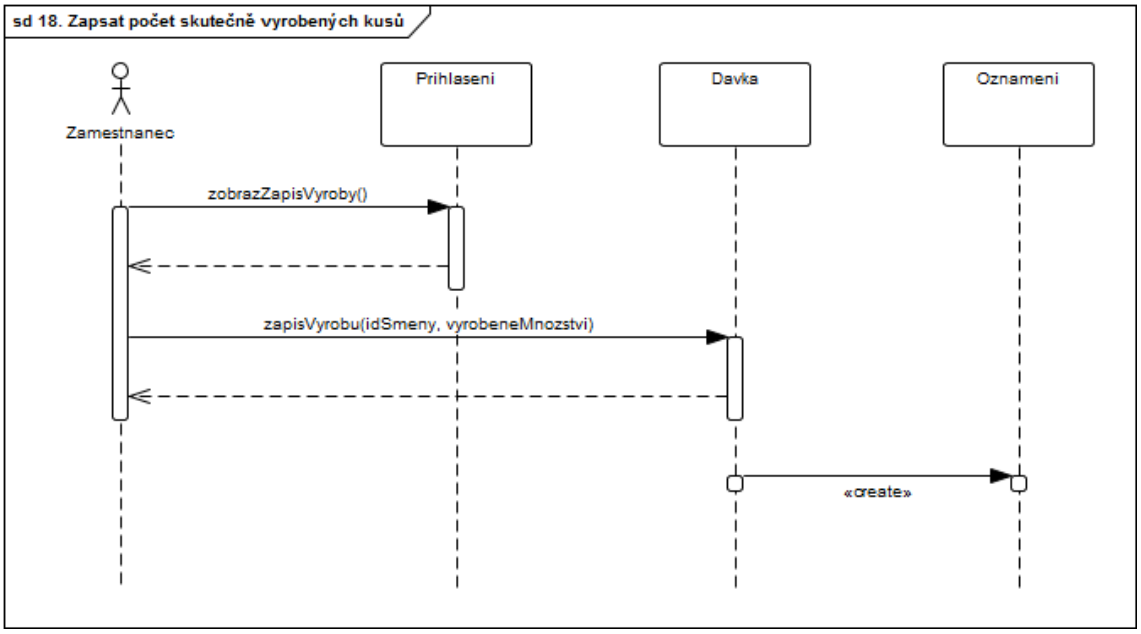
Obrázek 55: Sekvenční diagram: změnit údaje zaměstnance



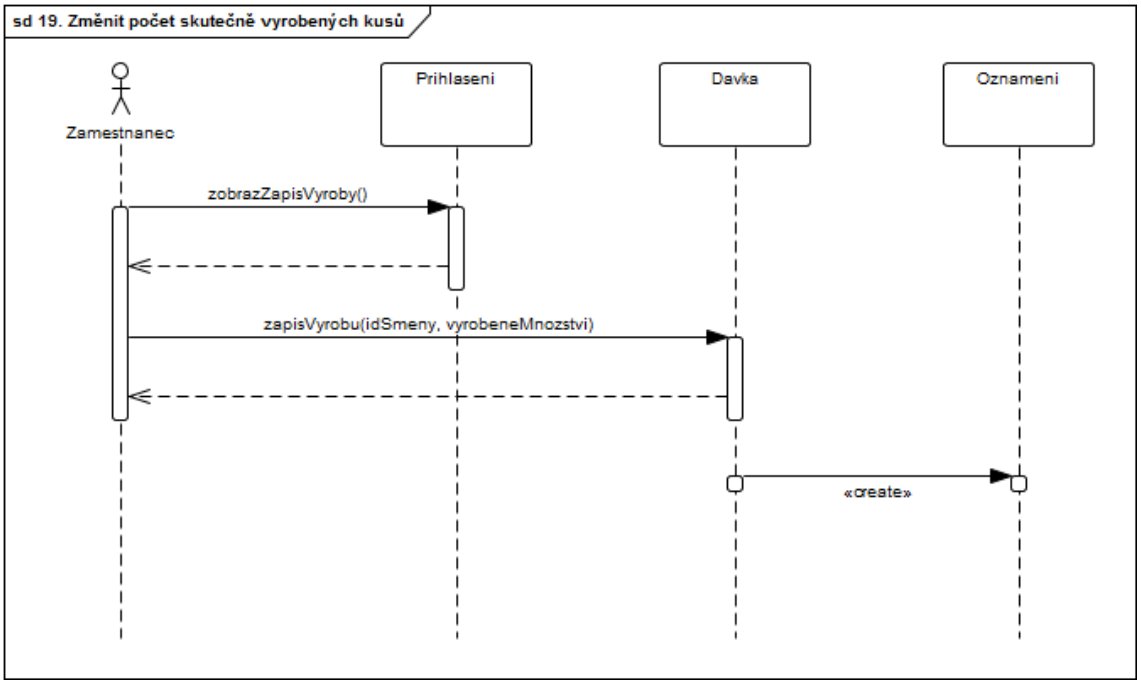
Obrázek 56: Sekvenční diagram: změnit přihlašovací údaje



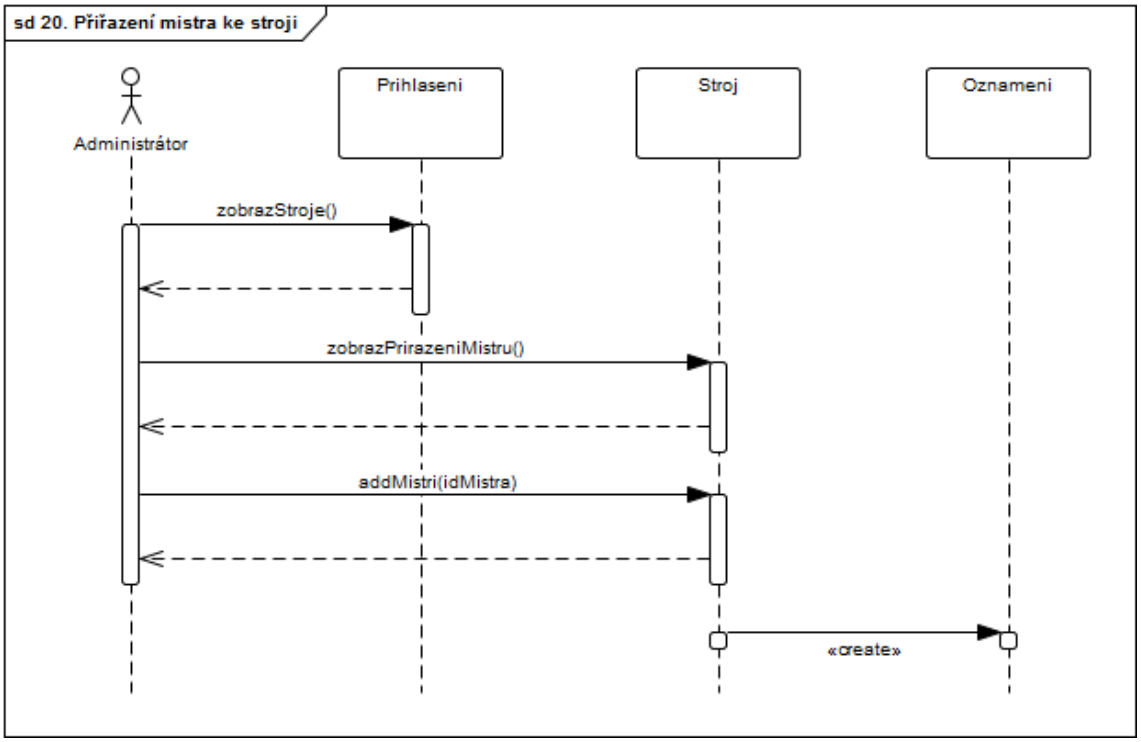
Obrázek 57: Sekvenční diagram: zrušit zaměstnance a jeho přihlašovací údaje



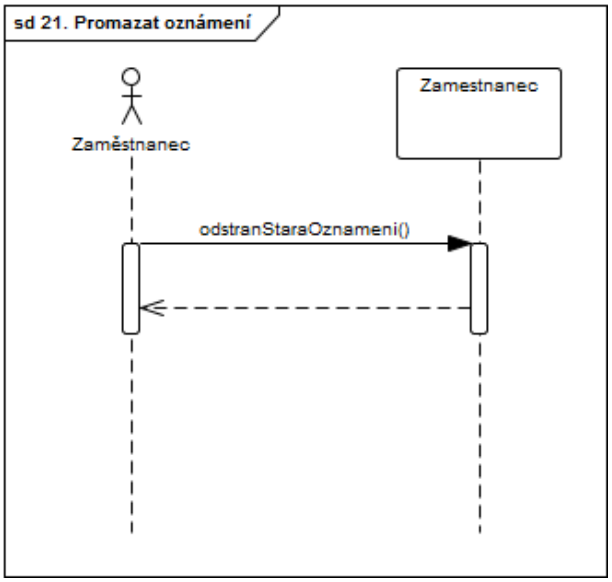
Obrázek 58: Sekvenční diagram: zapsat počet skutečně vyrobených kusů



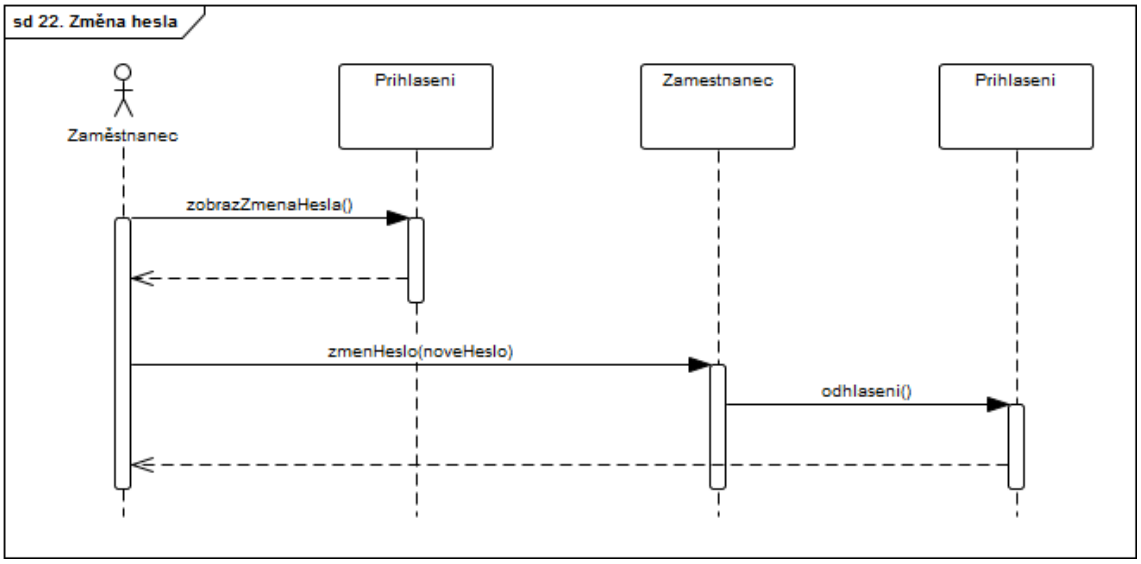
Obrázek 59: Sekvenční diagram: změnit počet skutečně vyrobených kusů



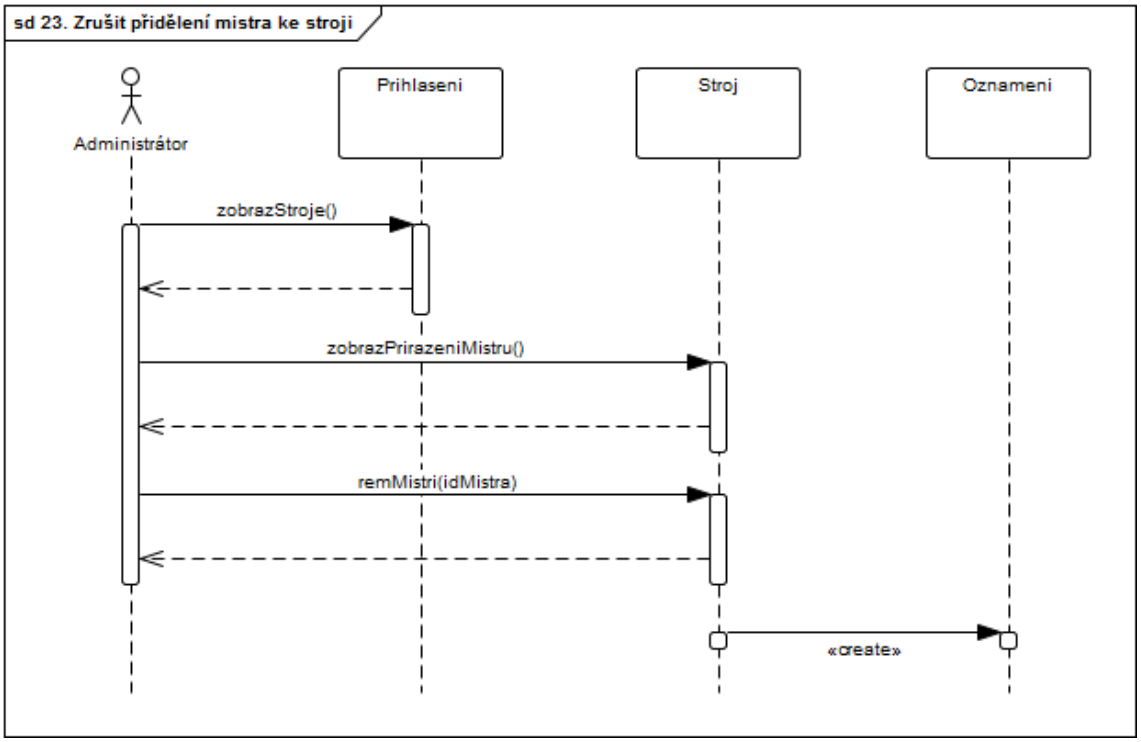
Obrázek 60: Sekvenční diagram: přiřazení mistra ke stroji



Obrázek 61: Sekvenční diagram: promazat oznámení



Obrázek 62: Sekvenční diagram: změna hesla



Obrázek 63: Sekvenční diagram: zrušit přidělení mistra ke stroji

C OCL v úplném formátu

1. délka hesla a loginu musí být delší jak 5 znaků

```
context Zamestnanec
inv velikostLoginu: self.login.size() > 4

context Zamestnanec
inv velikostHesla: self.heslo.size() > 4
```

2. číslo zaměstnance, login, SAP označení stroje a materiálu, číslo stroje a označení dávky musí být jedinečné

```
context Zamestnanec
inv jedinecneCisloZamestnance:
Zamestnanec.allInstances()->isUnique(zam | zam.cisloZamestnance)

context Zamestnanec
inv jedinecnyLogin:
Zamestnanec.allInstances()->isUnique(zam | zam.login)

context Stroj
inv jedinecneSAPOznaceniStroje:
Stroj.allInstances()->isUnique(str | str.SAPOznaceni)

context Stroj
inv jedinecneCisloStroje:
Stroj.allInstances()->isUnique(str | str.cisloStroje)

context Davka
inv jedinecneOznaceniDavky:
Davka.allInstances()->isUnique(dav | dav.oznaceni)

context Material
inv jedinecneSAPOznaceniMaterialu:
Material.allInstances()->isUnique(mat | mat.SAPOznaceni)
```

3. počet směn u dávky a počet kusů materiálu musí být větší jak 1

```
context Material
inv pocetKs: self.pocet >= 1

context Davka
inv mnozstviCelkem: self.celkoveMnozstvi >= 0
```

```
context Davka
inv pocetSmen: self.pocetSmen >= 1
```

4. v případě že jde o seřizovací dávku, musí být počet kusů roven 0

```
context Davka
inv serizeni:
  if self.serizeni = true then
    self.celkoveMnozstvi = 0
  else
    self.celkoveMnozstvi >= 1
  endif
```

5. vždy musí existovat jeden administrátor v aplikaci

```
context Vyroba::remZamestnanci(idx:Integer)
pre: idx <> 0

context Zamestnanec::setRole(rol:String)
post:
  if self.cisloZamestnance=0 then
    self.role = 'administrator'
  else
    self.role = rol
  endif
```

6. k neaktivnímu stroji nelze přidávat dávky

```
context Stroj::addDavka(dav:Davka)
pre: self.aktivni = true
```

7. funkce zobrazující kolik zbývá vyrobit

```
context Davka::zbyvaVyrobit():Integer
post: result = self.celkoveMnozstvi - self.skutecnostSmen->sum()
```

8. celkový počet vyrobených kusů nemůže být větší jak plánovaný

```
context Davka
inv: self.celkoveMnozstvi >=  self.skutecnostSmen->sum()
```

9. každý stroj musí mít alespoň jednoho mistra

```
context Stroj
inv: self.mistr->size() >= 1
```

10. hodnoty jsou vždy zadané (např. SAP označení a počet kusů materiálu)

```
context Material
inv vzdyZadanoSAPOznaceni: not self.SAPOznaceni.oclIsUndefined()

context Material
inv vzdyZadanPocet: not self.pocet.oclIsUndefined()
```

11. get/set metody (např. getPocet(), setPocet())

```
context Material::getPocet():Integer
post: result = self.pocet

context Oznameni::getHlaska():String
post: result = self.hlaska

context Material::setPocet(poc:Integer)
post: self.pocet = poc

context Oznameni::setHlaska(hlas:String)
post: self.hlaska = hlas
```

D Zdrojový kód aplikace na CD s návrhovým diagramem tříd