Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Webová aplikace pro vedení plánu obsazení kolejí v železniční stanici

Bc. Jan Žampach

Diplomová práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Žampach**
Osobní číslo:	**I11427**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Webová aplikace pro vedení plánu obsazení kolejí v železniční stanici**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Pro větší osobní železniční stanice jsou vedeny plány obsazení kolejí. Cílem diplomové práce je vytvořit webovou aplikaci pro vedení takového plánu. Klientská část aplikace by měla sloužit vlakovým dispečerům jako pomůcka k operativnímu vyhodnocování nastalých situací. Předpokladem je využití jazyka Java EE a dalších vhodných webových technologií.
V teoretické části práce bude proveden krátký přehled problematiky tvorby plánu obsazení kolejí. Před samotnou implementační částí DP bude provedena analytická část DP, která bude obsahovat všechny součásti UP (unified process).

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. Tacy Adam, Hanson Robert, Essington Jason, Tokke Anna. GWT in Action. Manning Publications, 2012. ISBN 978-1935182849.
2. Arlow J., Neustadt I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition). 2005. ISBN 978-0321321275.
3. Mojžíš V., Molková T. Technologie a řízení dopravy I. Univerzita Pardubice, 2002. ISBN 80-7194-424-6.

Vedoucí diplomové práce: **Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

PROHLÁŠENÍ AUTORA

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. 5. 2013 Bc. Jan Žampach

PODĚKOVÁNÍ

Tímto bych rád poděkoval především vedoucímu diplomové práce Ing. Michaelu Bažantovi, Ph.D. za cenné rady a připomínky při návrhu aplikace a řešení problémů a za jeho čas i ochotu při konzultacích.

Tato diplomová práce vznikla v rámci řešení projektu „Podpora stáží a odborných aktivit při inovaci oblasti terciárního vzdělávání na DFJP a FEI Univerzity Pardubice, reg. č.: CZ.1.07/2.4.00/17.0107“ v týmu „TRANSIM – simulace dopravních a obslužných systémů“.

ANOTACE

Práce se zabývá problematikou tvorby a návrhu plánu obsazení kolejí ve větších osobních železničních stanicích. Shrnuje současné techniky tvorby plánu obsazení kolejí, jeho využití staničními dispečery a dalšími řídícími pracovníky. Praktickým cílem práce je navrhnout a implementovat webovou aplikaci, která umožní jednak alespoň z části automatizovat proces tvorby plánu obsazení na základě vstupních dat a dále sloužit jako interaktivní pomůcka k operativnímu řízení provozu s možností dynamické změny.

KLÍČOVÁ SLOVA

plán obsazení kolejí, řízení vlakového provozu, webová aplikace

TITLE

Web application for online occupancy plan in a railway station.

ANNOTATION

The work deals with the problem of creation of track occupation diagram in larger passenger railway stations. It summarizes the current techniques of plans creation, its usefulness with dispatchers and other management personnel. The practical aim of this work is to design and implement a web application that would partially permit automate the process of plan creation and provide an interactive tool for operational traffic control with possibility of making dynamic changes.

KEYWORDS

track occupation diagram, railway transportation, web application

OBSAH

SEZNAM ZKRATEK

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
ČD	České dráhy
DOM	Document Object Model
DTO	Data Transfer Object
GVD	Grafikon vlakové dopravy
GWT	Google Web Toolkit
HQL	Hibernate Query Language
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JRE	Java Runtime Environment
JSNI	JavaScript Native Interface
MVC	Model View Controller
ORM	Object-relational mapping
POJO	Plain Old Java Object
POK	Plán obsazení kolejí
RCP	Regionální centrum provozu
RIA	Rich Internet Application
RPC	Remote Procedure Call
SQL	Structured Query Language
SVG	Scalable Vector Graphics
SŽDC	Správa železniční a dopravní cesty
UP	Unified Process
W-POK	Název aplikace
XSL	Extensible Stylesheet Language
ŽST	Železniční stanice

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Diplomová práce si klade za cíl z teoretického hlediska provést přehled problematiky současného stavu tvorby a využívání plánu obsazení kolejí ve větších osobních ŽST a navrhnout vhodný softwarový nástroj, který by umožnil částečnou automatizaci procesu tvorby plánu obsazení kolejí a sloužil by jako pomůcka staničním zaměstnancům.

Plán obsazení kolejí slouží především jako pomůcka staničním dispečerům k operativnímu řízení provozu. Doplňuje primární zdroje dat o dění ve stanici, ze kterých však není hned zřejmé obsazení kolejí. Jeho přínos je jednak při řešení běžných situací, jako jsou zpoždění vlaků (např. v případě, kdy má přijet opožděný vlak a na stejnou kolej již aktuálně čeká vlak jiný) nebo při naprosto rutinní sestavě vlakové cesty jednotlivých vlaků. Další přínos je pak zejména v případě řešení nestandardních situací, jako je např. výluka koleje, kdy musí být všechny vlaky z inkriminované koleje přemístěny jinam. Při rozhodování, na jakou kolej budou umístěny, pak může pomoci právě plán obsazení.

Data pro plán jsou čerpána z aktuálního GVD, přičemž jsou převedena do specifické grafické podoby, která přehledně koncentruje data vztažená k ŽST, pro kterou je daný plán obsazení určen. Soubor potřebných dat pro vytvoření plánu tvoří zejména seznam dopravních kolejí a seznam vlaků, které se v ŽST vyskytují v průběhu jednoho dne. Důležitými atributy vlaku jsou druh, číslo, výchozí pobytová kolej a časové údaje o pobytu. Z tohoto souboru dat logicky vyplývá grafická struktura plánu, kterou tvoří horizontálně zakreslený seznam kolejí v časovém rozsahu 0.00 – 24.00 hod. Vlaky jsou na těchto kolejích (osách) zakresleny v odpovídajících časech specifickým grafickým způsobem, kde je patrná délka pobytu, čas a směr příjezdu a odjezdu a kalendář, kdy daný vlak jede.

Grafická podoba plánu obsazení je specifická a podléhá zavedeným konvencím jednotlivých ŽST. To také může být jedním z důvodů, proč v současné době neexistuje žádný softwarový nástroj, který by automatizoval náročnou manuální sestavu. Dříve se plány rýsovaly ručně pomocí barevných tuší. Dnes jsou využívány podpůrné vektorové grafické nástroje, které dovolují úpravy již vytvořených plánů a následnou lepší manipulaci s hotovými plány (ať již je to vektorový dokument nebo velkoformátový tisk). Samotná sestava je však v podstatě opět manuální práce pověřeného zaměstnance a mnohdy se neliší od způsobu práce s perem a tuší. (1)

V rámci diplomové práce bude navrhnuta a implementována webová aplikace pro sestavení a vedení plánu. Architektura aplikace bude typu klient-server, tedy uživatelská část aplikace bude fungovat ve webovém prohlížeči. Výhodou takového řešení bude okamžitá dostupnost pro vlakové dispečery. Nebude tak nutné instalovat speciální software na všechny potřebné počítače. Aplikace umožní zadávání aktuálních dat např. o zpožděních a poskytne tak nástroj pro interaktivní a operativní vyhodnocení nastalých situací.

1 PLÁN OBSAZENÍ KOLEJÍ A JEHO VYUŽITÍ

Plán obsazení kolejí, celým názvem *Plán obsazení dopravních kolejí* (dále jen POK), patří do tzv. *Technologických pomůcek ke GVD* (grafikon vlakové dopravy). Řídí se dle předpisu (2) a je vypracováván v ŽST uvedených v seznamu, který vydává RCP (regionální centrum provozu). V grafickém vyjádření se v něm zakresluje obsazení kolejí všemi pravidelnými vlaky včetně přestavných jízd, přičemž se uvádí i doba pravidelného obsazení kolejovými vozidly. POK schvaluje vrchní přednosta ŽST a následně je vydán jako služební pomůcka pro potřebu provozních zaměstnanců. (2)

1.1 Pomůcka pro dispečery ve velkých ŽST

POK zastává při řízení provozu v ŽST roli podpůrné pomůcky při řešení běžných i nestandardních situací. Bohužel v současném pojetí POK se jedná o statickou pomůcku (vytisknutý plán, plán ve formě obrázku nebo plán vytvořený v aplikaci, ve které byl vytvářen, ale nelze do něj triviálně zasahovat), která se nemění v případě vzniku běžných ani nestandardních situací. Vytvářená aplikace jde naopak nad rámec tohoto statického pojetí a cílem je vytvoření pomůcky pro dispečery pro případ změny POK z libovolných příčin (výluka koleje, zpoždění vlaků apod.).

1.2 Současný stav sestavy plánu obsazení kolejí

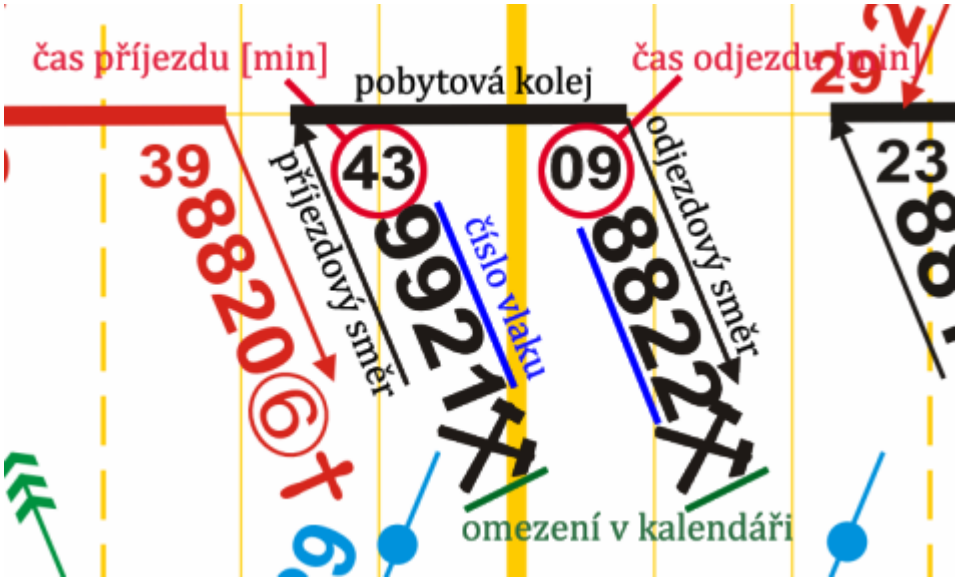
Jak již bylo zmíněno, v současné době nejsou pro sestavu POK používány jednotné software nástroje a POK nejsou sestavovány centrálně. Každá větší vlaková stanice má své konvence. To je způsobeno jednak tím, že neexistuje žádný specializovaný software ani podpůrná pomůcka pro tvorbu plánů a jednak neexistencí interního předpisu, který by problém tvorby POK nějakým způsobem upravoval. To sice dává možnosti dispečerům upravovat si POK specificky dle svých potřeb, ale tím pádem neexistuje mezi POK žádná jednotnost. Data pro POK vychází z aktuálního GVD a jsou zachycena v *Seznamech vlaků pro staniční zaměstnance* nebo v interních informačních systémech. U ČD je to např. IS Kango. (1)

1.3 Grafická podoba plánu obsazení kolejí

Grafická podoba podléhá specifickým pravidlům a konvencím a není upravena v (2). Z toho důvodu se grafické podoby POK v jednotlivých ŽST liší a zachycují i rozdílné úrovně detailů.

Vlastností plánu je, že je tvořen pro generický den. To znamená, že v něm jsou zahrnuty všechny vlaky bez rozdílu, zda jedou každý den nebo jen vybrané dny roku. Tím se může stát plán v kritických místech nepřehledný, a proto jsou zavedena pomocná grafická pravidla. Např. pro odlišení vlaků jedoucí méně než tři dny v týdnu se využívá jiné barevné odlišení.

Plán je tvořen na ose *y* seznamem kolejí dané ŽST a osa *x* představuje čas od 0.00 do 24.00 hod. Počáteční a koncový bod osy *x* tedy splývá. Tím je vytvořena jakási síť, kde kombinací koleje a času vznikne jednoznačná poloha vlaku v plánu. Vlak je zachycen na dané časové ose (dle koleje) úsečkou, jejíž délku určuje délka pobytu vlaku ve stanici. Pokud vlak pouze projíždí, bude znázorněn pouze čarou křížící projížděnou kolej. Pokud vlak ve stanici jízdu končí (resp. svou jízdu začíná), bude vlak znázorněn šipkou končící (resp. začínající) u dané koleje. Specifický „tvar“ vlaku v plánu, kdy má vlak určen pobyt na koleji, znázorňuje obrázek 1.



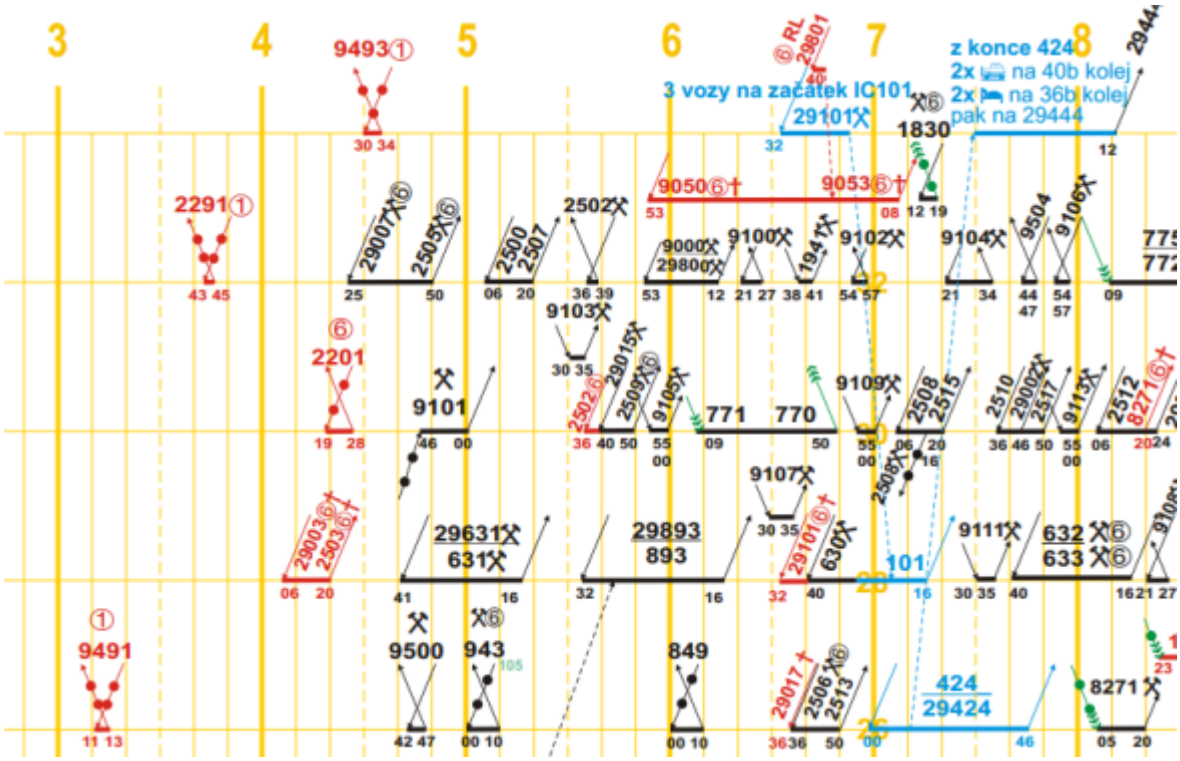
Obrázek 1 – Příklad zakreslení vlaku v plánu. Zdroj: (3)

1.4 Odlišnosti v plánech obsazení kolejí v různých ŽST

Z analýzy vypracované v (1) vyplývá, že konvence tvorby POK v různých ŽST se velmi liší jak způsobem fyzické tvorby plánu, tak i způsobem zakreslení jednotlivých značek. Např. v ŽST Praha, hl. n. a Pardubice, hl. n. jsou pro tvorbu plánu využívány vektorové programy Corel Draw™, v ŽST Plzeň, hl. n. je používán rozšířený a všude dostupný MS Excel. Tedy už z principu se tyto vytvořené plány musí velmi odlišovat, protože každý ze zmíněných softwarových nástrojů nabízí naprosto rozdílné možnosti a funkcionality.

1.4.1 ŽST Praha, hl. n.

Nejvíce komplexní vzor pro tvorbu POK představuje varianta ze ŽST Praha, hl. n, jejíž část zobrazuje obrázek 2. Splňuje základní požadavky přehlednosti a použitelnosti. Vychází z barevných konvencí zakreslování různých druhů vlaků, podkladové mřížky s více úrovněmi přesnosti (zpřesňující časovou osu), průběžného číslování kolejí uvnitř plánu apod. Tvorba probíhá ručně v programu Corel Draw™ a vychází se při ní především z dokumentu *Seznam vlaků pro staniční zaměstnance*. Tvorba je velice náročná na čas a přesnost, ale výsledek je vysoce přehledný plán ve tvaru vektorového dokumentu. (1)

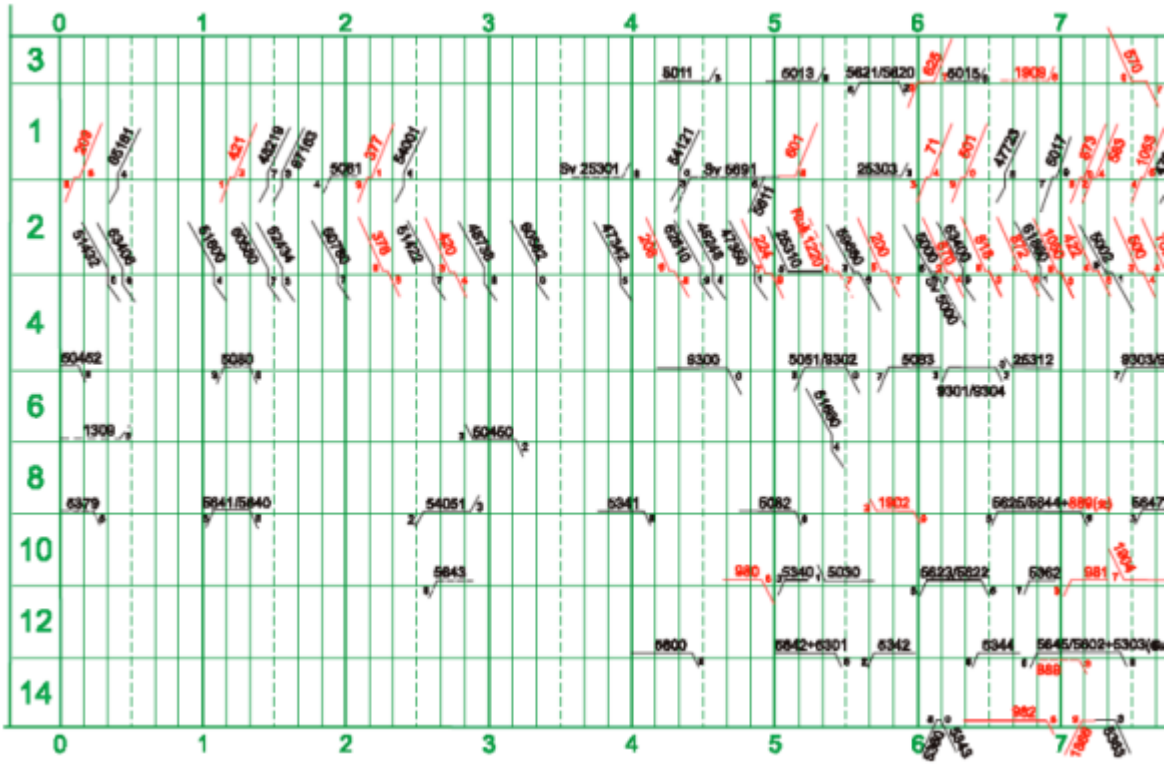


Obrázek 2 – Část POK ze ŽST Praha, hl. n. Zdroj: (3)

1.4.2 ŽST Pardubice, hl. n.

V ŽST Pardubice, hl. n. je POK součástí komplexnějšího plánu – *Plán provozních procesů stanice*. Z hlediska použitých konvencí při tvorbě POK je velmi podobný tomu ze ŽST Praha, hl. n. a je taktéž vytvořen ve vektorovém programu Corel Draw™. Odlišností je rozdělení kolejí do skupin dle účelu. Skupinu tvoří vjezdové a odjezdové koleje nákladní části stanice a koleje osobní části stanice. Tímto je zvýrazněna logická struktura stanice. Barevná konvence je podobná již výše zmíněnému POK ze ŽST Praha, hl. n. Odlišuje se ale např. způsob zakončení čar odjezdových a příjezdových směrů, kde v tomto případě nejsou zakresleny šipky. Vlaky s nulovým časovým pobytem jsou zakresleny nikoliv

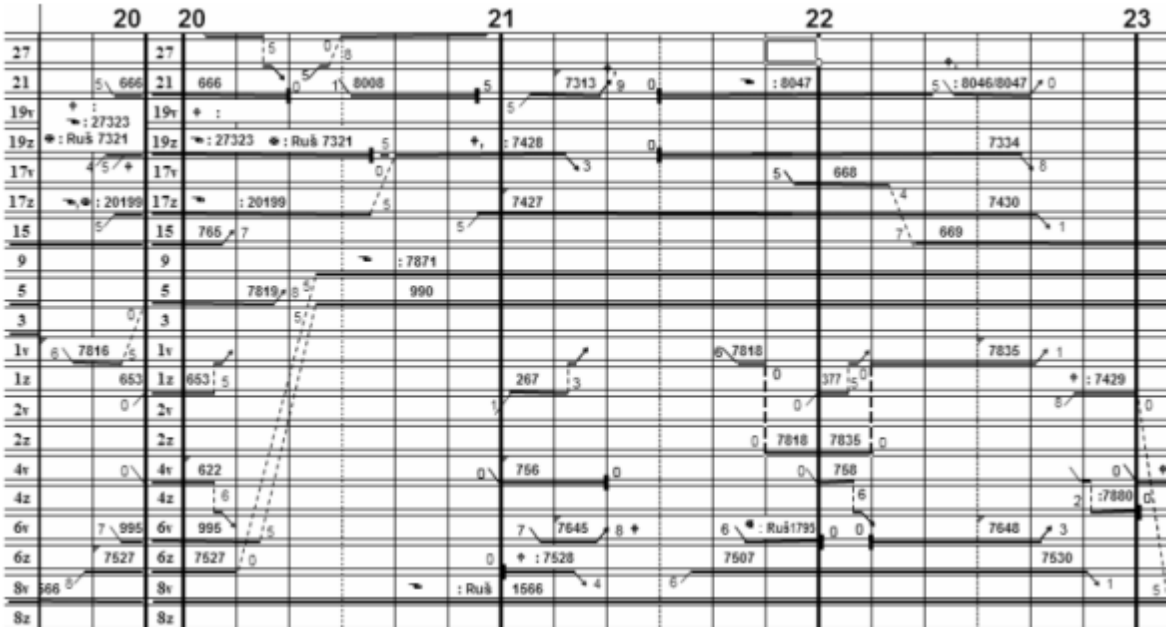
vodorovnou, ale svislou čárou. Nejsou zaznamenány ani piktogramy kalendářního omezení vlaků, popisky nástupišť ani průběžné číslování kolejí. Část plánu je znázorněná na obrázku, viz obrázek 3. (1)



Obrázek 3 – Část POK ze ŽST Pardubice, hl. n. Zdroj: (1)

1.4.3 ŽST Plzeň, hl. n.

POK vytvářený v ŽST Plzeň, hl. n. pomocí tabulkového procesoru má specifickou podobu. Barevné schéma je tvořeno pouze černou a bílou barvou a při jeho sestavě je využíváno vlastností tabulkového procesoru, kdy jednotlivé tvary jsou zvýrazňovány pomocí ohraničení buněk. Přesto je i takto vytvořený POK použitelný v reálném provozu. Část plánu znázorňuje obrázek 4. (1)



Obrázek 4 – Část POK ze ŽST Plzeň, hl. n. Zdroj: (1)

1.5 Výhody a nevýhody současného řešení

Současná situace tvorby a používání POK je zažitá již řadu let. I přes to, že tvorba nového POK je manuálně a časově náročná práce, je tento proces nezbytný. Výhodou současného řešení je možnost zahrnout při tvorbě nového POK specifické grafické prvky a zažité konvence, jelikož tyto nejsou nijak upraveny v (1). POK je tak plně přizpůsoben a „ušit na míru“ dané ŽST. Nevýhodou současného řešení je především náročnost a zdlouhavost procesu tvorby, kdy je nutné (ať již v jakémkoli grafickém nebo tabulkovém editoru) umisťovat veškeré elementy na plán ručně. Při tom vstupuje do role i lidský faktor a možnost zanesení chyby. Také z toho důvodu je každý plán revidován vrchním přednostou ŽST, který za něj zodpovídá. Po revizi je tento vydán jako služební pomůcka pro potřeby provozních zaměstnanců.

Z výše uvedeného textu plyne, že nelze triviálně vyvinout komplexní nástroj pro tvorbu a vedení POK, který by uspokojil stávající potřeby staničních dispečerů. Lze vyvinout jakýsi kompromis mezi přehledností a použitelností. Aplikace může na rozdíl od statických plánů těžit ze své funkcionality, více viz kapitola 2.

1.6 Nástroje pro tvorbu plánu obsazení kolejí

Jak již bylo zmíněno výše, v současné době se v komerčním prostředí nepoužívá žádná specializovaná aplikace, která by umožnila nebo usnadnila sestavu a vedení POK, naopak jsou využívány běžně dostupné aplikace, které nejsou k podobným účelům vůbec primárně určené.

V minulosti byla k účelu sestavy POK již implementována jedna aplikace, rovněž v rámci diplomové práce. V následujícím odstavci bude krátce představena.

Aplikace PLOK (1) primárně slouží jako pomůcka při náročném procesu sestavy plánu obsazení. Jedná se o desktopovou aplikaci, jejíž hlavním cílem je poskytnout software základnu pro sestavitele plánu, díky které lze vytvořit pokročilou kostru plánu a sestaviteli umožnit před generováním plánu pro tisk nastavit mnoho parametrů, které jsou specifické pro určenou ŽST. Jsou to např. barvy, úhly natočení šipek, tvary šipek, tvary popisků apod. Dále je možné sestavený plán editovat (umisťovat vlaky, měnit přečíslování vlaků atd.) a připravit jej tak pro následný export nebo tisk. Aplikace PLOK vychází ze stejných zdrojových dat, jako aplikace vyvíjená v rámci této diplomové práce.

2 MOTIVACE A NÁVRH K ŘEŠENÍ PROBLÉMU

Kapitola obsahuje návrh možného řešení problémů při sestavě POK, probraných v minulé kapitole. Dále budou zmíněny technologie, které budou použity při implementaci aplikace.

Aplikace vytvářená v rámci této diplomové práce nemá ambici poskytnout sestaviteli kostru plánu, který je následně tisknut nebo dále používán jako statická pomůcka, ale vytvořit možnou koncepci aplikace, jejíž hlavní součástí bude dynamický plán, který poskytuje možnosti interaktivních dynamických zásahů do plánu a umožní tak dispečerům využít lépe jeho potenciál, než je tomu ve statickém pojetí POK.

Nedostatky při používání POK, plynoucí z jeho vlastností (statický plán, ruční tvorba), se snaží řešit tato diplomová práce. Koncepce řešení je následující. Navrhnout aplikaci, která by usnadnila sestavu POK na základě zdrojových dat. Pomocí aplikace bude tento sestavený POK veden a spravován. Umožní dispečerům (dle různých oprávnění) tuto aplikaci využívat k dynamické modifikaci plánu z hlediska zadávání zpoždění vlaků, změn doby pobytů na kolejích, změn pobytových kolejí. Tyto změny budou proveditelné jak na úrovni grafické – přímo v plánu, tak na úrovni seznamu všech vlaků (s možností filtrování dle různých parametrů). Aplikace by měla sloužit jako interaktivní podpůrná pomůcka pro rozhodování při řešení nastalých nestandardních situací. Aplikace si neklade za cíl naplno nahradit stávající zavedenou pomůcku, protože by bylo velmi obtížné obsáhnout v POK všechny zavedené konvence – i tak by se lišily v závislosti na vybrané ŽST.

2.1 Architektura řešení

Při praktickém využití POK v centru řízení provozu bývá k dispozici více kopií plánů, zpravidla každý dispečer má k dispozici svojí kopii. Navrhovaná aplikace bude tuto zvyklost podporovat díky architektuře klient-server, kdy na serveru společnosti bude spuštěna jedna instance aplikace a uživatelé (dispečeři) si budou moci kdykoli zobrazit aktuální POK, popř. jej editovat pod svým uživatelským účtem a se svými oprávněními.

Tato architektura řešení má určitě nespornou výhodu v tom, že aplikaci si může zobrazit jakákoli oprávněná osoba s připojením do firemní sítě. Odpadá tak nutnost instalovat kopie aplikace (popř. jejich aktualizace) na jednotlivé počítače ve společnosti, pokud by se jednalo o desktopovou aplikaci. Jediným požadavkem na klientské počítače tak zůstává mít aktualizovaný prohlížeč a funkční připojení k síti.

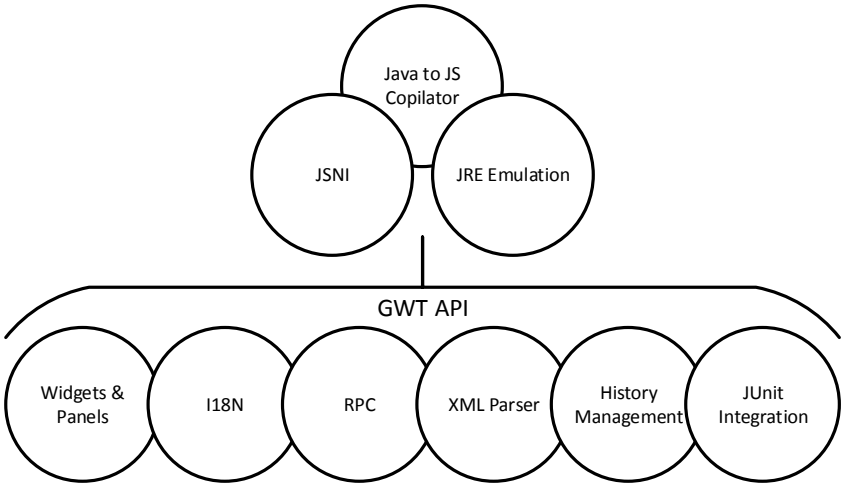
Nevýhodou architektury klient-server může být ve chvílích přetížení sítě samotná komunikace po síti, kdy server bude vytížen a při komunikaci s ním se mohou vyskytnout prodlevy. Další nevýhodou zmíněné architektury je poměrně náročnější implementace, jak z hlediska použitých technologií, tak z hlediska návrhu a samotného ladění a nasazení aplikace. Mohou nastat i problémy s nekompatibilitou na straně serveru, např. při použití jiné verze, popř. úplně jiného webového serveru.

2.2 Zvolené technologie

V následujících kapitolách bude proveden přehled nejdůležitějších technologií, které budou použity při realizaci aplikace.

2.2.1 GWT

GWT (Google Web Toolkit) je sada vývojářských nástrojů, paleta komponent, která umožňuje vytvářet RIA aplikace kompletně v jazyce Java. Rozdíl oproti jiným webovým frameworkům spočívá právě v možnosti psát kód na straně prohlížeče v jazyce Java, namísto v jazyce JavaScript. JavaScript je slabě typovaný jazyk, má málo datových typů, slabou modularizaci. Proto JavaScript není pro implementaci větších projektů moc vhodný. Na druhou stranu JavaScript kód funguje skoro ve všech prohlížečích (pokud není zakázáno spouštění). GWT spojuje výhody jazyka Java (a všechny jeho vlastnosti) s výhodami JavaScriptu. GWT poskytuje kompilátor, který je schopný vyprodukovat JavaScript kód, možnost psaní nativního JavaScript kódu (JSNI) a emulované knihovny JRE. Nad těmito funkcionalitami GWT je poskytováno zdokumentované API, viz obrázek 5. (4)



Obrázek 5 – Sada nástrojů GWT. Zdroj: (4 str. 6)

GWT kompilátor

Úkolem kompilátoru je převést Java kód na JavaScript podobně, jako když se kompiluje Java kód do byte kódu. Kompilaci má na starost Java program `com.google.gwt.dev.GWTCompiler`, jehož parametrem při kompilaci je umístění modulu aplikace. Modul je sada Java tříd a souvisejících souborů, které jsou definovány XML konfiguračním souborem. Vstupním bodem do modulu je tzv. `EntryPoint` třída, která je vykonána po startu aplikace.

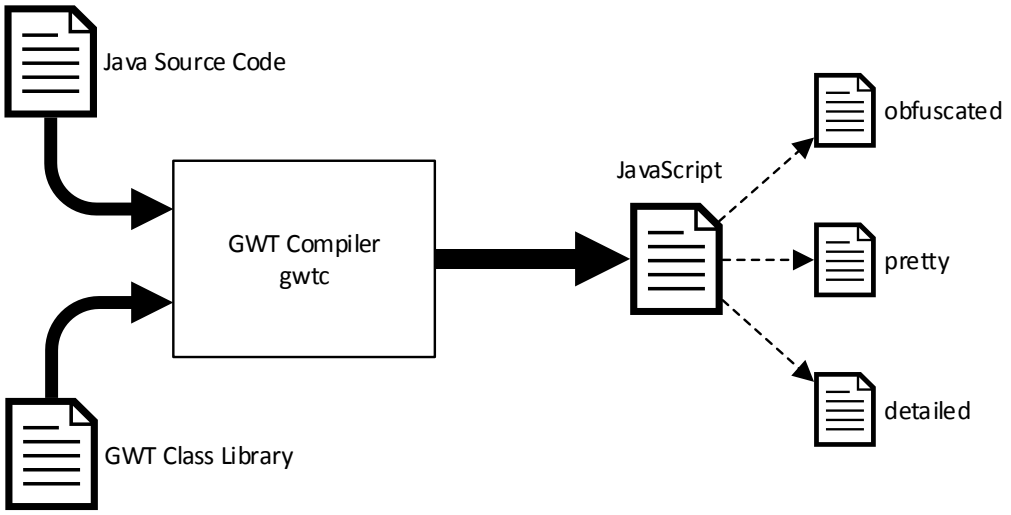
Kompilace začíná výše zmíněnou třídou, následována dalšími závislostmi (knihovny, další třídy). Existuje zde rozdíl od kompilátoru `javac` – do kompilace se nezahrnou všechny třídy obsažené v modulu, ale jen ty, které se budou používat.

Kompilátor poskytuje tři módy kompilace, které určují výsledný vzhled (a do určité míry i vlastnosti a výkonnost) výsledného JavaScript kódu. Výchozím módem je `obfuscated`, jehož výsledkem je maximálně komprimovaný kód, který je pro visuální inspekci prakticky nečitelný. Důvodem je minimalizace objemu kódu a jeho lepší výkon při interpretaci prohlížečem. Výhodou může být i zamezení odkrytí implementace třetí straně.

Dalším módem kompilace je styl `pretty`, který generuje čitelný JavaScript kód. Z kódu však stále nelze přečíst veškeré implementační detaily (např. jaké třídě náleží daná metoda apod.).

Tento požadavek pokrývá `detailed` mód kompilace, který doplňuje mód `pretty` o názvy tříd u metod jako součást jejich názvu. Díky tomuto módu lze poměrně snadno procházet zkompilovaný kód a porovnávat s Java kódem a případně ladit a hledat potíže, způsobené JavaScript kódem v prohlížeči.

Mód `pretty` a `detailed` jsou tedy určeny především pro ladění aplikace. Pro produkční nasazení je určen výhradně mód `obfuscated`. Obrázek 6 znázorňuje proces kompilace GWT kompilátorem a poukazuje na možné módy kompilace GWT kompilátorem.



Obrázek 6 – Proces GWT kompilace do jazyka JavaScript a její možné úrovně. Zdroj: (4)

Další důležitou vlastností GWT kompilátoru je to, že jeho zdrojem mohou být pouze Java zdrojové soubory, nikoliv kompilované byte kód soubory. To může být překážka při použití knihoven bez dostupných zdrojových kódů.

Omezením kompilátoru je i nutné dodržení kompatibility verze GWT kompilátoru a verze Java. Např. Java verze 7 je podporována od GWT verze 2.5.0 a vyšší. Při použití nekompatibilních verzí dojde k chybě při kompilaci a ta bude ukončena. Toto omezení se však týká pouze kódu, který bude kompilován na stranu klienta, na straně serveru toto omezení neplatí.

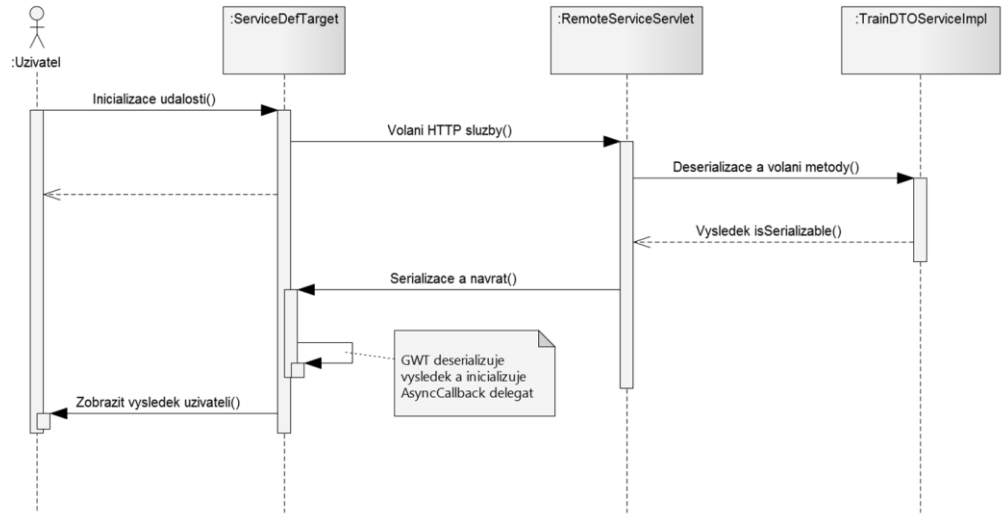
Jak již bylo zmíněno, výstupem kompilátoru je JavaScript kód. Pro dosažení maximální kompatibility mezi prohlížeči je výsledkem kompilace několik samostatných souborů. Každý tento soubor je optimalizovaný pro daný engine prohlížeče (např. Gecko, WebKit), jeho verzi a lokalizaci. Toto nastavení lze specifikovat v konfiguračním XML souboru nastavení kompilátoru. Tzv. bootstrap skript, který je nejprve prohlížečem načten, se postará o načtení správného souboru pro konkrétní prohlížeč. Výhodou této koncepce je, že prohlížeč si načte jen ta data, která jsou opravdu nutná pro správné fungování aplikace v konkrétním prohlížeči. (4)

RPC komunikace v GWT

Přenos informací ze strany klienta na server a naopak je základním předpokladem většiny netriviálních webových aplikací. Současné prohlížeče poskytují speciální rozhraní nazvané `XMLHttpRequest`, které umožňuje komunikovat mezi klientem a serverem prostřednictvím

protokolu HTTP bez nutnosti znovunačtení stránky. Toto rozhraní tvoří základ pro tzv. AJAX technologii.

GWT poskytuje dva nástroje komunikace založené na rozhraní `XMLHttpRequest`, jednak třídu `RequestBuilder`, která v podstatě zapouzdřuje toto rozhraní a dále mechanizmus GWT RPC (Remote Procedure Call). Ten je z praktického hlediska více použitelný, protože dovoluje posílat a přijímat reálné Java objekty mezi klientem a serverem.



Obrázek 7 – Sekvenční diagram RPC přenosu dat. Zdroj: (5)

Dynamiku RPC přenosu dat zachycuje sekvenční diagram, viz obrázek 7. Celý proces je popsán v následujících odstavcích.

Před použitím RPC komunikace je nutné definovat rozhraní služby (`TrainDTOService`), které bude implementované na straně serveru. Rozhraní definuje metody, prostřednictvím kterých bude možné se serverem komunikovat. Toto rozhraní dědí od rozhraní `RemoteService`.

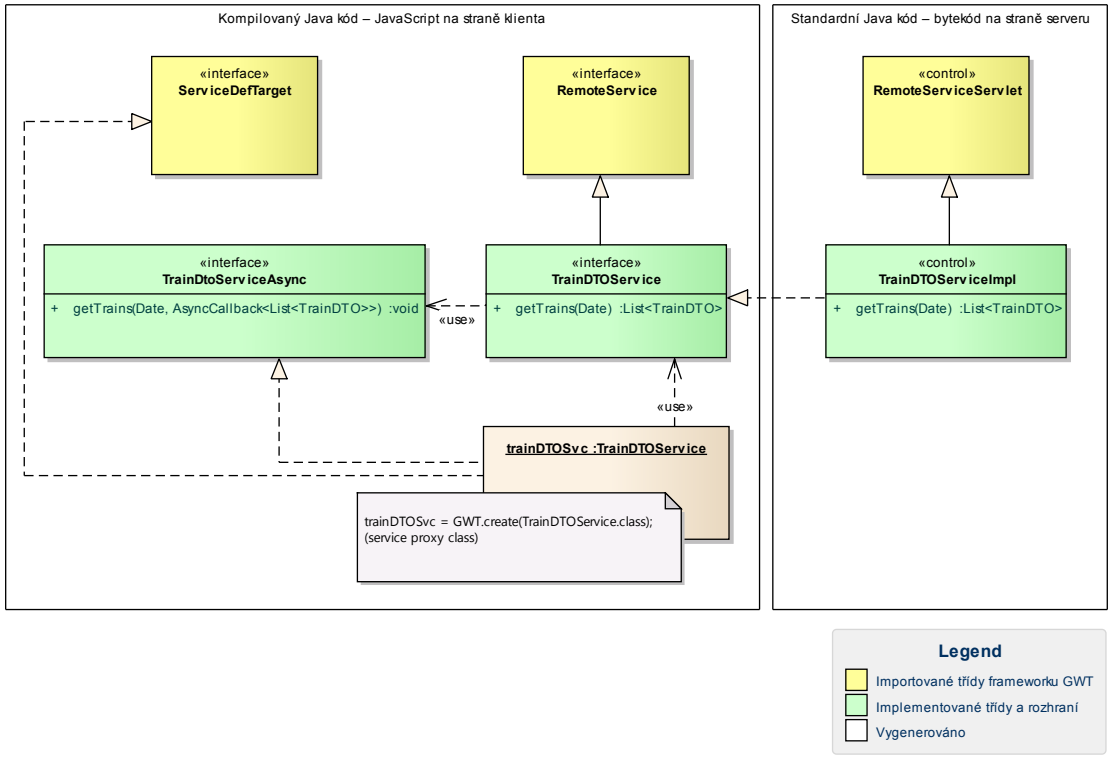
Následně je nutné implementovat na straně serveru definované rozhraní služby. Tato třída je potomkem rozhraní `RemoteServiceServlet` – servlet přijímá data ve formě textu, který deserializuje na Java objekty. Zmíněné objekty jsou již zpracovány implementovanými

metodami na straně serveru. Po zpracování jsou tyto objekty opět serializovány prostřednictvím `RemoteServiceServlet` a ve formě textu díky rozhraní `XMLHttpRequest` zaslány zpět klientovi.

Pro přijetí dat na straně klienta je nutné zavolat implementovanou metodu, které se vloží jako parametr generický delegát, tzv. `AsyncCallback` handler. Ten zpracuje výsledek ze serveru a prostřednictvím anonymních metod `onFailure()` (zavolá se v případě chyby na straně serveru) a `onSuccess()` (zavolá se v případě úspěšného dokončení) lze přistoupit k přijatému, transparentním způsobem deserializovanému, objektu.

GWT umožňuje i standardní práci s výjimkami a jejich předávání ze strany serveru na stranu klienta. Po volání implementované metody z rozhraní, která může generovat výjimku, je nutné tuto standardním způsobem ošetřit. Mechanismem GWT RPC lze přenášet i vlastní implementované třídy výjimek, je pouze nutné, aby implementovaly rozhraní `IsSerializable`, popř. `Serializable`.

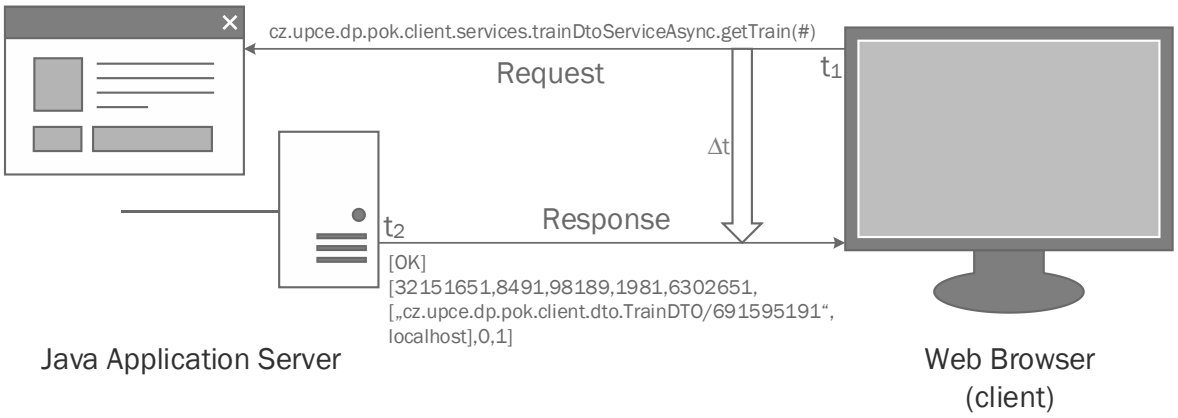
UML diagram tříd znázorňující model RPC komunikace, viz obrázek 8.



Obrázek 8 – UML diagram tříd sloužících pro GWT RPC komunikaci. Zdroj: (6)

Důležitým aspektem RPC komunikace je, že vzdálené (klient-server) volání metod probíhá asynchronním způsobem. To znamená, že po zavolání metody vzdálené služby

pokračuje na straně klienta provádění kódu bez čekání na návratovou hodnotu. Mezi požadavkem na server a jeho odpovědí tedy uplyne časový úsek, viz obrázek 9.



Obrázek 9 – Znázornění časového průběhu GWT RPC komunikace. Zdroj: (4 str. 349)

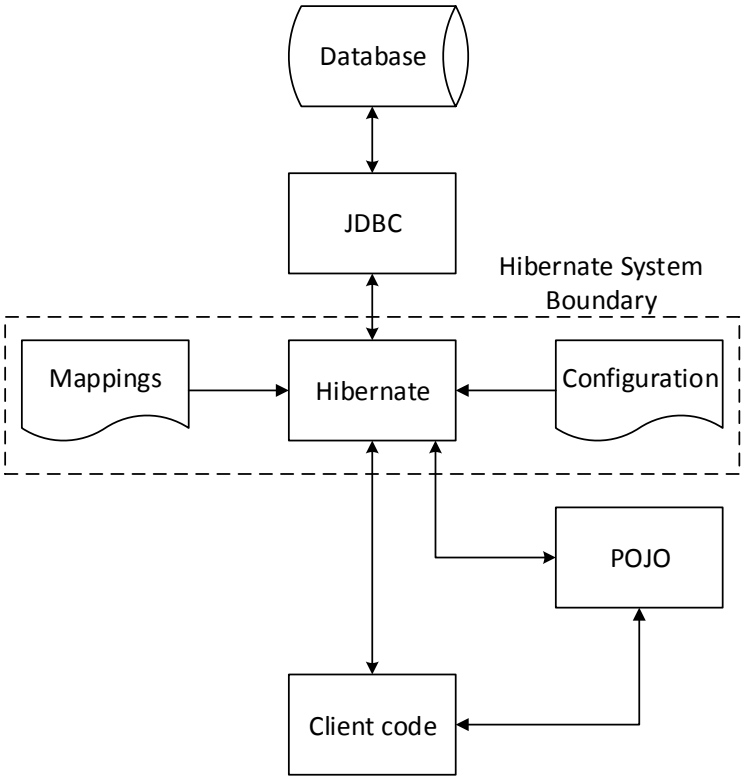
Použití tohoto typu komunikace přirozeně plyne z fungování rozhraní `XMLHttpRequest`, které je při komunikaci využíváno.

Asynchronní komunikace přináší do prostředí síťové komunikace zásadní výhodu: při volání metody na serveru se nečeká na výsledek a právě prováděný kód na straně klienta může pokračovat dál bez prodlevy. Tím nevznikají nepříjemné prodlevy a celá webová aplikace se blíží chováním aplikaci desktopové. (4), (6)

2.2.2 ORM Hibernate

Hibernate je open-source Framework určený pro objektově relační mapování (ORM) v jazyce Java. Hlavním úkolem je především synchronizace mezi entitními objekty, které se používají v aplikaci a jejich relační reprezentací na úrovni databáze. Tím se zajišťuje perzistence dat. Do jisté míry se tak řeší problém mezi rozdílným přístupem k relační reprezentaci a objektové reprezentaci dat díky nahrazení přímých SQL dotazů vyšší vrstvou pro manipulaci s objekty. Framework umožňuje mapovat Java objekty na relační databázové tabulky a zároveň Java datové typy na SQL datové typy.

Při dotazování do databáze jsou generovány SQL dotazy, čímž je zvýšena abstrakce od konkrétní použité databáze. Aplikace využívající ORM Hibernate jsou přenositelné z hlediska použité databázové platformy a pro programátora je práce s perzistentní vrstvou transparentní. Roli frameworku Hibernate v Java aplikaci znázorňuje obrázek 10. (7)



Obrázek 10 – Role Hibernate v Java aplikaci. Zdroj: (7 str. 2)

Mapování Java tříd na databázové entity je uskutečněno pomocí XML konfiguračních souborů nebo pomocí tzv. anotací. Většina moderních IDE podporuje generování XML konfiguračních souborů na základě struktury databáze, ke které se IDE připojuje díky rozhraní JDBC. V XML konfiguračním souboru každé entity je zachycena její relace k ostatním entitám a jsou definovány vazby např. 1:N (one to many) nebo M:N (many to many), dále jsou definovány mapovací datové typy (Java objekt vs. typ v databázi), jejich omezení (NOT NULL, délka řetězců, velikost číselných typů), názvy tabulek a sloupců, resp. objektů a atributů a další nastavení, viz následující fragment XML.

```
<hibernate-mapping>
  <class name="cz.upce.dp.pok.server.model.Traintype" schema="POK"
table="TRAINTYPE">
    <id name="id" type="int">
      <column name="ID" precision="8" scale="0"/>
      <generator class="increment"/>
    </id>
    <property name="traintypenumber" type="string">
      <column length="10" name="TRAINTYPENUMBER" not-null="true"/>
    </property>
    <set inverse="true" name="trains">
      <key>
        <column name="TRAINTYPE" precision="8" scale="0"/>
      </key>
      <one-to-many class="cz.upce.dp.pok.server.model.Train"/>
    </set>
  </class>
</hibernate-mapping>
```

Entity z databáze jsou mapovány do tzv. Java POJO (plain old java object) objektů. Tyto objekty bývají obvykle taktéž vygenerovány spolu s konfiguračními soubory. Programátor má možnost upravit např. implicitně zvolené datové typy a uzpůsobit mapování svým potřebám (např. použít výčtový typ apod.). Pro reprezentaci relací jsou nejčastěji použity generické `HashSet` kolekce. Na POJO třídu je v zásadě kladeno pouze jedno omezení – musí mít bezparametrický konstruktor (stačí privátní). Jinak může POJO třída obsahovat libovolnou funkcionalitu – obvykle se od tohoto však upouští a zachovává se datový charakter třídy dle návrhového vzoru MVC a doplňují se pouze překryté metody `equals()` a `hashCode()`, popř. `toString()`.

Pro realizaci dotazů do datových objektů poskytuje Hibernate jazyk HQL, který je v mnoha ohledech podobný klasické SQL syntaxi. K dispozici jsou i nástroje pro práci s transakcemi.

V standardním nastavení jsou data načítána dle techniky Lazy loading – objekt je z databáze fyzicky načten a následně inicializován až v okamžiku jeho prvního použití, např. z dotazované velké kolekce dat je ve výsledku potřeba jen část, zbytek nebude z databáze fyzicky načten. Tím se zvyšuje výkonost perzistentní vrstvy. (7), (8)

2.2.3 SVG grafika

Jedním z nefunkčních požadavků na aplikaci je, aby grafická část POK byla vykreslena vektorovou grafikou. Ta poskytuje oproti grafice rastrové výhody, jež lze v navrhované aplikaci plně využít. Klíčové vlastnosti SVG (Scalable Vector Graphics) jsou:

- používáno pro definici vektorově založené grafiky na webu,
- je definováno pomocí XML formátu,
- možnost libovolného zvětšení a zmenšení bez ztráty kvality vykreslení,
- s každým objektem (vlaky, koleje) lze pracovat samostatně a vázat na něj určité dynamické akce (zpoždění, změna koleje apod.),
- text v SVG grafice lze vyhledat,
- vektorová grafika je podporována moderními prohlížeči a lze provázat s jinými standardy (DOM či XSL),
- elementy lze stylovat pomocí CSS,
- paměťová náročnost výsledné reprezentace je mnohem menší,
- standard SVG je zaštítěn konsorciem W3C.

SVG je v současných prohlížečích plně podporováno (kromě Internet Explorer verze 8 a nižší) a SVG kód je možné vložit do stránky několika způsoby. Nejvhodnějším se jeví vložení SVG elementu přímo do HTML kódu stránky. (9)

Zmíněný způsob byl použit i v aplikaci, viz ukázka v příloze D. Na ní je zobrazeno vykreslení jednoho „tvaru“ vlaku. Na vykreslení jsou použity styly (barva, šířka čáry, výplň), grafické SVG objekty (přímka, text), a SVG transformace (translace, rotace). Logicky navazující SVG fragmenty jsou shlukovány do skupin, s nimiž se lépe pracuje, např. se na ně dobře aplikují transformace. Pomocí skupiny zmíněných operací je zhotoven celý POK.

K vytvoření takového SVG plánu bylo nutné sestavovat ho dynamicky v závislosti na aktuálních požadavcích pro vykreslení. Proto bylo nutné najít způsob, jakým lze generovat SVG elementy přímo do HTML stránky dynamicky pomocí programovacího jazyka. K tomu účelu byla zvolena knihovna `lib-gwt-svg`, díky níž lze pomocí konstrukcí jazyka Java vytvářet libovolné SVG elementy. Tato knihovna je určena pro Framework GWT, který samotný poskytuje pouze grafické komponenty pro rastrové kreslení na HTML Canvas. Knihovna je úzce propojena s GWT a výsledný kód je kompilován jako součást klientské části aplikace. Způsob práce s knihovnou je znázorněn na následující ukázce zdrojového kódu, pomocí kterého se vytváří transformace skupiny, v které bude umístěna šipka[1].

```
OMSVGGElement arrowGroupObject = doc.createSVGGElement();
OMSVGTransform translate = svg.createSVGTransform();
translate.setTranslate(x, y);
OMSVGTransform rotate = svg.createSVGTransform();

Directions direction = arrival ? Directions.getDirection(train.getNextStation())
: Directions.getDirection(train.getPrevStation());

rotate.setRotate(direction.getAngle(), 0, 0);

arrowGroupObject.getTransform().getBaseVal().appendItem(translate);
arrowGroupObject.getTransform().getBaseVal().appendItem(rotate);

OMSVGGElement arrowGroup = doc.createSVGGElement();
OMSVGLineElement line = doc.createSVGLineElement(0,0,Consts.ARROW_LINE_LENGTH,0);
arrowGroup.appendChild(line);
```

[1] Šipka je součástí „tvaru“ vlaku v POK a má za úkol znázornit odjezdový nebo příjezdový směr vlaku.

3 REALIZACE APLIKACE

V následujících kapitolách bude představen souhrn jednotlivých kroků metodiky UP (Unified Process), podle které byla vyvíjena výsledná aplikace. V diplomové práci budou uvedeny důležité pasáže z hlediska implementace nebo návrhu modelu, kompletní model (projekt v programu Enterprise Architect) bude uložen jako příloha na optickém médiu. Informace týkající se metodiky UP byly čerpány z (10).

3.1 Požadavky na aplikaci

V následujících kapitolách budou shrnuty funkční a nefunkční požadavky na aplikaci a důležité realizované případy užití.

3.1.1 Funkční požadavky

Funkční požadavky na aplikaci byly rozděleny do balíčků dle logického rozčlenění aplikace na moduly. A to na hlavní modul nazvaný *POK* – bude obsahovat hlavní část aplikace, kterou je vykreslený plán obsazení kolejí. Dalším logickým celkem aplikace je seznam vlaků – umožňuje uživateli přehledně vlaky řadit dle různých parametrů, filtrovat a pro uživatele vymezené role i editovat, přidávat vlaky nové a přidávat výluky a zpoždění. Dále lze přidávat výluky kolejím. Tento modul je nazván jednoduše *Vlaky*. Modul zabezpečující přihlašování uživatelů je nazván *Přihlášení* a modul pro správu uživatelů je nazván *Administrace*. Následuje seznam funkčních požadavků rozdělený právě dle výše zmíněných modulů.

Funkční požadavky pro modul *POK*:

- Systém bude zobrazovat aktuální stav POK.
- Systém bude v POK umožňovat rychlé zadání zpoždění.
- Systém bude v POK umožňovat rychlé zadání změny koleje.
- Systém bude v POK umožňovat rychlé zadání změny doby pobytu.
- Systém bude po editaci uživatelem upozorňovat na konflikty v POK.
- Systém bude umožňovat filtrování v POK dle data.
- Systém bude umožňovat zobrazení detailních informací o vybraném vlaku formou popisku po najetí kurzorem nad vlak.

Funkční požadavky pro modul *Vlaky*:

- Systém bude zobrazovat seznam všech vlaků.
- Systém bude umožňovat řadit, popř. filtrovat vlaky:
    - dle jejich čísla,
    - dle jejich kategorie,
    - dle času plánovaného příjezdu,
    - dle jejich kalendáře.
- Systém bude umožňovat přiřazení výluky vlaku (centrální úroveň).
- Systém bude umožňovat přiřazení výluky vlaku (lokální úroveň).
- Systém bude umožňovat přidání nového vlaku (centrální úroveň).
- Systém bude umožňovat přidání nového vlaku (lokální úroveň).
- Systém bude umožňovat základní editaci atributů vlaku (centrální úroveň).
- Systém bude umožňovat základní editaci atributů vlaku (lokální úroveň).
- Systém bude umožňovat přidat zpoždění vlaku (centrální úroveň).
- Systém bude umožňovat přidat zpoždění vlaku (lokální úroveň).
- Systém bude umožňovat správu editovaných změn.
- Systém bude umožňovat import nových dat do databáze ze souboru.

Funkční požadavky pro modul *Přihlášení*:

- Systém umožní uživateli přihlásit se do systému.
- Systém bude umožňovat uživateli změnit si heslo.

Funkční požadavky pro modul *Administrace*:

- Systém bude umožňovat jednoduchou správu uživatelů.

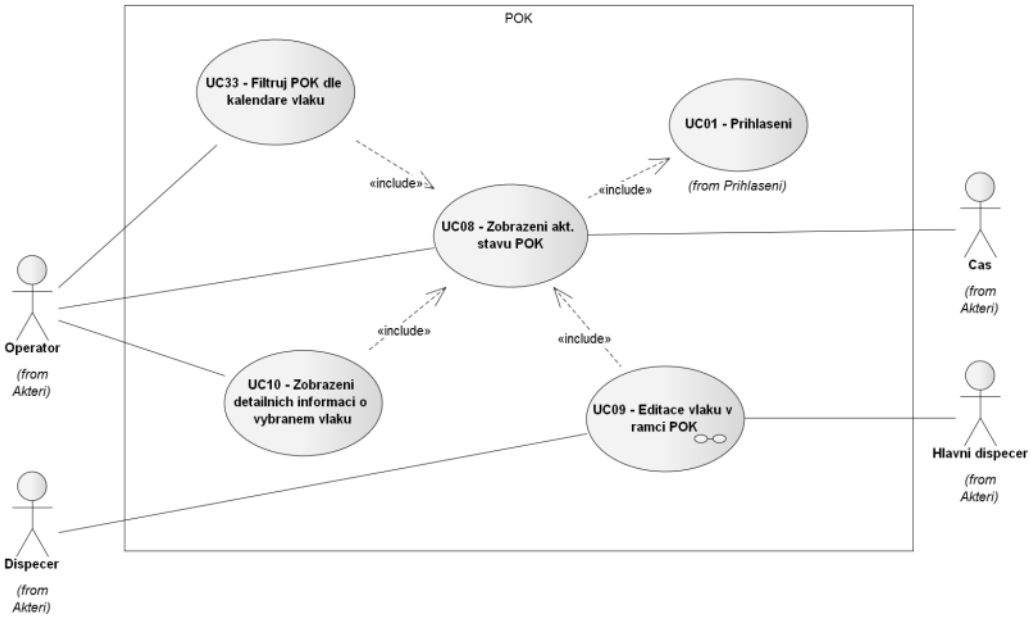
3.1.2 Nefunkční požadavky

Nefunkční požadavky zahrnují oblast požadavků na výkon, spolehlivost, škálovatelnost, bezpečnost a použitelnost systému.

- Systém bude pracovat na architektuře klient-server.
- Systém bude vyvinut v technologii Java EE.
- Systém bude přístupný přes webový prohlížeč.
- Systém bude POK vykreslovat vektorovou grafikou.
- Systém bude autentizovat a autorizovat uživatele.
- Systém bude pracovat s uživatelskými rolemi ve třech úrovních.
- Systém bude pracovat s databází Oracle.
- Systém bude pracovat s přesnosti času 0,5 min.

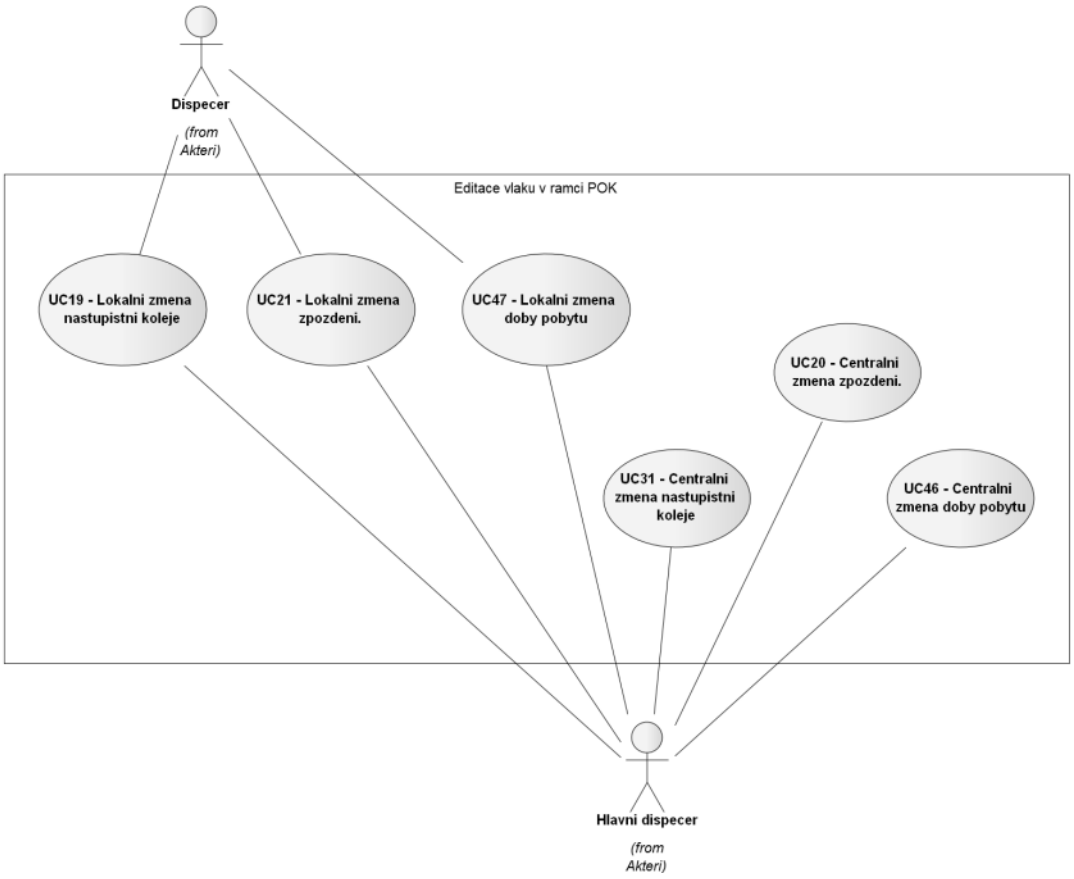
3.2 Případy užití

Realizace případů užití byla provedena na základě seznamu požadavků a jednotlivé skupiny případů užití byly rozčleněny do stejných modulů. Obrázek 11 zachycuje realizaci případu užití pro modul *POK*.



Obrázek 11 – Realizace případu užití pro modul POK

Všechny případy užití jsou podmíněny přihlášením do systému, které je realizováno případem užití UC01. Základním případem užití je pak případ UC08 – Zobrazení akt. stavu POK, v kterém vystupují aktéři Operátor (uživatel se základním stupněm oprávnění) a aktér Čas, díky kterému je zobrazován plán pro aktuální den (seznam vlaků je implicitně vyfiltrován pouze na vlaky, které jedou v daný den). Aktér Operátor má dále možnost filtrovat plán dle libovolného jiného dne, popř. zobrazit všechny vlaky (UC33) a dále má možnost zobrazit detailní informace o vybraném vlaku formou najetí kurzoru myši nad vybraný vlak (UC10). V modulu *POK* dále vystupují aktéři Dispečer a Hlavní dispečer, kteří mají možnost do plánu zasahovat formou editací (viz obrázek 12) s rozdílnou úrovní oprávnění. UC09 Editace vlaku v rámci POK tedy obsahuje další případy užití pro funkční požadavky rychlé editace vlaku v rámci POK – přidání zpoždění, změna koleje a změna délky pobytu. Obojí v lokálním a globálním měřítku změn, kde je vidět rozdíl v oprávnění mezi rolemi Hlavní dispečer a Dispečer, viz obrázek 12.



Obrázek 12 – Editace vlaku v rámci POK

Detail UC09 – Editace vlaku v rámci POK znázorňuje rozdíl mezi oprávněním uživatelské role Dispečer a Hlavní dispečer, kdy Hlavní dispečer má možnost provádět oba typy změn v databázi – v lokálním i globálním měřítku změn. UC09 postihuje možnosti změn v rámci modulu POK – jsou to rychlé uživatelské editace přímo ve vykresleném plánu obsazení, kdy uživatel má možnost kliknout na vybraný vlak a provádět s vlakem zmíněné operace.

Ostatní diagramy případů užití viz příloha A. Ke všem vytvořeným případům užití byly sepsány i scénáře. Pro ilustraci jsou uvedeny scénáře pro centrální změnu zpoždění a přidání vlaku na lokální úrovni, viz tabulka 1 a tabulka 2.

Tabulka 1 – Scénář pro případ užití UC20 – Centrální změna zpoždění

Název případu užití	UC20 – Centrální změna zpoždění
Stručný popis	Systém bude umožňovat v hlavním modulu POK rychlé provedení změny zpoždění na centrální úrovni, změny se projeví pro všechny uživatele. Tato operace bude dostupná pouze pro uživatele vymezené role.
Primární aktéři	Hlavní dispečer
Vstupní podmínky	Uživatel je přihlášen do systému a má dostatečné oprávnění.
Hlavní scénář	1. UC začíná, když Hlavní dispečer vybere v POK vlak, kterému chce přiřadit zpoždění, klikne na něj levým tlačítkem a vybere „Zadat zpoždění“. 2. Systém zobrazí uživateli okno, kde lze zadat zpoždění v minutách. 3. Hlavní dispečer zadá zpoždění v řádu minut, zvolí, že změna se provede centrálně a klikne na „Provést změny“. 4. Systém zkontroluje, zda je možné zadané zpoždění bez problému provést, provede změnu a překreslí POK.
Vedlejší scénář (detekce konfliktu)	4. Systém zjistí, že kolej je v daném čase již obsazena a zobrazí uživateli hlášku s upozorněním. 5. Systém zvýrazní konflikt na POK.

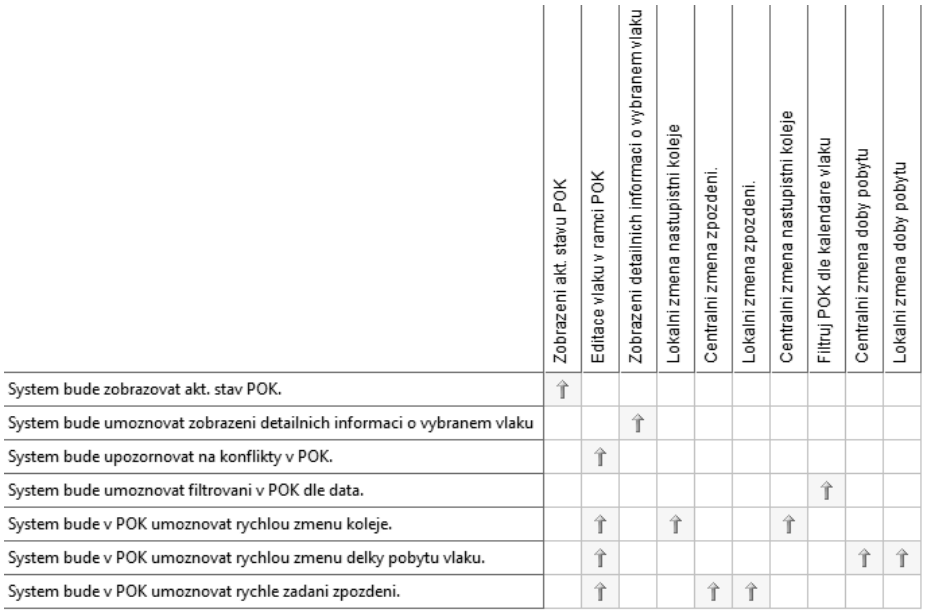
Tabulka 2 – Scénář pro případ užití UC39 – Přidání vlaku na lokální úrovni

Název případu užití	UC39 – Přidání vlaku na lokální úrovni
Stručný popis	Systém bude umožňovat v rámci Editoru seznamu vlaků přidání nového vlaku do databáze na lokální úrovni – změny se projeví pouze pro daného uživatele. Dostupné pouze pro uživatele vymezené role.
Primární aktéři	Hlavní dispečer, Dispečer
Vstupní podmínky	Uživatel je přihlášen do systému a má dostatečné oprávnění.
Hlavní scénář	1. UC začíná, když Dispečer nebo Hlavní dispečer vybere z nabídky Editoru vlaku „Lokální meřítko změn“ a následně volbu „Přidat vlak“. 2. Systém vloží do seznamu vlaků nový vlak s prázdnými atributy, kde je nutné vyplnit číslo vlaku, jeho kolej a volitelně příjezdový a odjezdový směr a čas příjezdu a odjezdu. 3. Hlavní dispečer nebo Dispečer vyplní povinné údaje a volitelné editovatelné atributy vlaku a stiskne tlačítko „Uložit vlak“. 4. Systém zkontroluje unikátní číslo vlaku a uloží vlak do databáze.
Vedlejší scénář (neunikátní číslo vlaku)	4. Systém zjistí duplicitní číslo vlaku, oznámí to uživateli a vlak do databáze neuloží. 5. Hlavní dispečer nebo Dispečer změní číslo vlaku na unikátní a znovu stiskne tlačítko „Uložit vlak“.

3.3 Sledování požadavků

V metodice UP je velmi důležité sledovat, zda jsou všechny funkční požadavky pokryty případy užití. Protože mezi funkčními požadavky a případy užití existuje mnoho relací (jeden případ užití může pokrývat mnoho více požadavků a naopak jeden požadavek může být vyjádřen ve více různých případech užití), je vhodné využít k zachycení pokrytí požadavků matici sledovatelnosti.

Obrázek 13 znázorňuje matici sledovatelnosti pro modul *POK*, kde na vodorovné ose je seznam požadavků a na svislé ose jsou případy užití. Pokrytí jednotlivých požadavků vyjadřuje šipka na příslušných souřadnicích. Matice sledovatelnosti pro modul *Vlaky* viz projekt v programu Enterprise Architect v příloze.



	Zobrazeni akt. stavu POK	Editace vlaku v ramci POK	Zobrazeni detailnich informaci o vybranem vlaku	Lokalni zmena nastupistni koleje	Centralni zmena zpozdeni.	Lokalni zmena zpozdeni.	Centralni zmena nastupistni koleje	Filtruj POK dle kalendare vlaku	Centralni zmena doby pobytu	Lokalni zmena doby pobytu
System bude zobrazovat akt. stav POK.	↑									
System bude umoznovat zobrazeni detailnich informaci o vybranem vlaku			↑							
System bude upozornovat na konflikty v POK.		↑								
System bude umoznovat filtrovani v POK dle data.								↑		
System bude v POK umoznovat rychlou zmenu koleje.		↑		↑			↑			
System bude v POK umoznovat rychlou zmenu delky pobytu vlaku.		↑							↑	↑
System bude v POK umoznovat rychle zadani zpozdeni.		↑			↑	↑				

Obrázek 13 – Matice sledovatelnosti požadavků pro modul POK

3.4 Analytický model

Před návrhem analytického modelu tříd byly hledány analytické třídy pomocí metody podstatných jmen a sloves. Jako podklad sloužilo zadání diplomové práce, zápisky z konzultací s vedoucím diplomové práce a hlavně seznam požadavků. Výsledný analytický model se nachází v příloze B.

3.5 Datový model

Datový model odráží požadavky pro uchování struktury dat pro POK a běh aplikace. Důležitým aspektem při návrhu byl požadavek na editaci vlaků, přičemž vždy by mělo být možné tyto změny vrátit zpět do původního stavu. Proto byla navržena struktura umožňující uchovat všechny změny v datech pro POK a zároveň zobrazovat poslední stav po úpravách. Datový model zároveň dbá na třetí normální formu a importovaná data s původně duplicitními hodnotami uchovává efektivně v patřičných tabulkách. Diagram datového modelu se nachází v příloze C.

3.5.1 Struktura datového modelu

V následujících odstavcích budou popsány důležité tabulky.

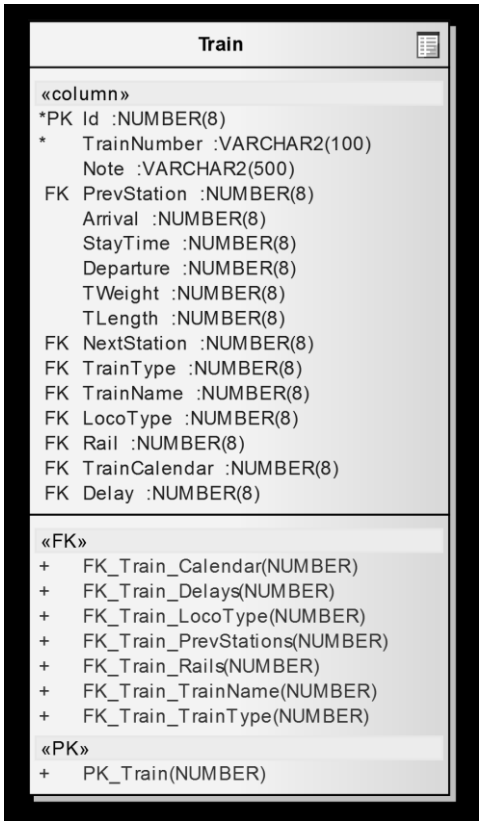
Tabulka Train

Tabulka `Train` uchovává všechny nezbytné atributy vlaku, které se k němu přímo vážou nebo mají velkou variabilitu (číslo vlaku, poznámka, příjezd, odjezd, doba pobytu,

hmotnost a délka). Ostatní atributy jsou soustředěny v jiných tabulkách, navázaných pomocí cizích klíčů.

I když by mělo být číslo vlaku unikátní, byl pro jistotu zvolen primárním klíčem jednoznačně generovaný číselný sloupec. Dalším důvodem je i možná neunikátnost čísla vlaku v případě jeho editace a uchovávání obou instancí tohoto vlaku. Číslo vlaku by se mělo teoreticky skládat pouze z numerických znaků, pro potřeby editace byl ale zvolen raději alfanumerický datový typ `VARCHAR2` (např. lze označit editovaný vlak postfixem „b“ za číslem vlaku).

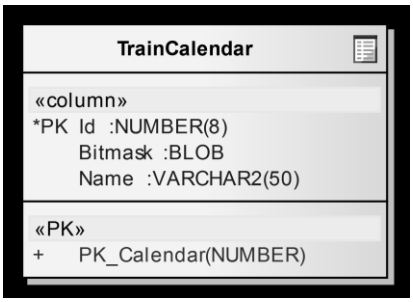
Pro uchování časových dat vlaku (příjezd, pobyt na koleji, odjezd) se nejevil jako vhodný vestavěný datový typ `DATE`, popř. `TIMESTAMP`, protože ani jeden nevystihuje charakter tohoto časového razítka. Byl zvolen datový typ `NUMBER`, do kterého bylo uloženo časové razítko (příjezd, odjezd) ve formě počtu sekund od půlnoci, resp. počet sekund pro vyjádření doby pobytu na koleji. Tento časový údaj je následně jednoduše přepočítán po načtení z databáze na odpovídající datový typ jazyka Java. V případě, že čas není uveden, je v databázi uchována hodnota `NULL`. Přesný popis použitých datových typů je na UML diagramu, viz obrázek 14.



Obrázek 14 – Tabulka Train

Tabulka TrainCalendar

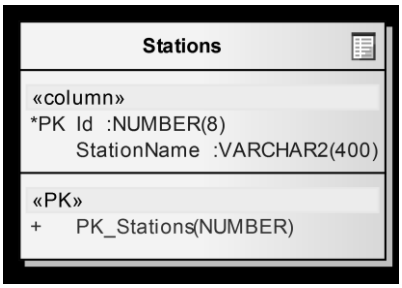
Tabulka `TrainCalendar` uchovává jednotlivé kalendáře vlaků. Ty jsou v databázi uloženy pomocí datového typu `BLOB` a kalendář je tak reprezentován jako pole bitů, kdy každý den v roce, kdy vlak jede, je bit nastaven na logickou „1“. Nevýhodou tohoto řešení je nemožnost provádět restrikci přímo na úrovni databáze, ale je nutné načíst všechny vlaky s jejich kalendáři a filtraci provést až s použitím bitové logiky. Výhodou je naopak kompaktnost řešení na úrovni perzistentní vrstvy. Při využití bitových operací je manipulace s kalendářem snadná. Obrázek 15 zobrazuje UML diagram tabulky.



Obrázek 15 – Tabulka TrainCalendar

Tabulka Stations

Tabulka `Stations` uchovává fyzicky dostupné okolní ŽST okolo dané ŽST, pro kterou je POK tvořen. Tyto stanice udávají výsledný „tvar“ vlaku v POK, přičemž nastavení tohoto tvaru není uchováno v databázi, ale ve formě výčtového typu v samostatné třídě. Tabulka je napojena na tabulku `Train` pomocí dvou cizích klíčů (předchozí a následná stanice). UML diagram tabulky `Stations` viz obrázek 16.



Obrázek 16 – Tabulka Stations

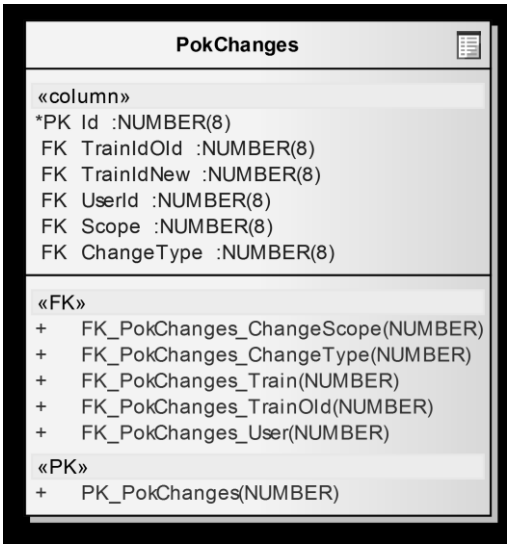
Analogicky jako tabulka `Stations` jsou tvořeny i ostatní tabulky tvořící zbylé atributy vlaku:

- `Rails` – uchovává seznam dostupných kolejí v dané ŽST,
- `LocoType` – uchovává typy hnacích vozidel, vyskytujících se v POK,

- `TrainType` – uchovává typy vlaků (Os, Ex, Pn, R, …),
- `TrainName` – uchovává názvy vlaků,
- `Delay` – uchovává údaje o zpoždění vlaků.

Tabulka PokChanges

Tabulka `PokChanges` slouží v databázi pro uchování provedených změn v rámci POK. Obsahuje pouze primární klíč a zbylé atributy jsou tvořeny cizími klíči. Je to především vazba na tabulku `Train`, kde je uchována vazba na původní needitovaný vlak a nový editovaný vlak. Dále jsou uchovány vazby na tabulky evidující typ změny, měřítko změny a uživatele, jenž danou změnu provedl. Díky těmto atributům lze aktuální stav POK načítat pro konkrétního uživatele na míru, včetně jemu náležejících změn (v lokálním nebo globálním měřítku). UML diagram tabulky `PokChanges` viz obrázek 17.



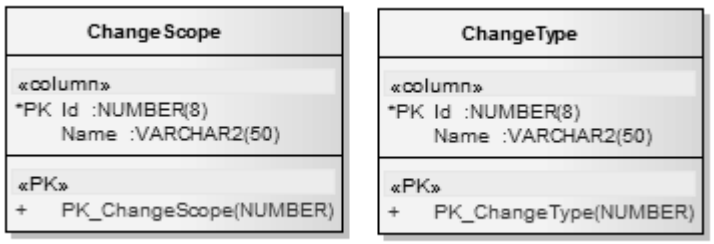
Obrázek 17 – Tabulka PokChanges

Tabulky ChangeType a ChangeScope

Tabulky `ChangeType` a `ChangeScope` uchovávají informace pro identifikaci provedených editací vlaků. Tabulka `ChangeType` uchovává výčet uživatelských změn, které lze u vlaku provést. Těmito změnami jsou:

- vložení nového vlaku, reprezentováno stavem *nový*,
- přiřazení výluky k vlaku, reprezentováno stavem *výluka*,
- editace atributů vlaku, reprezentováno stavem *editace* a
- přiřazení zpoždění k vlaku, reprezentováno stavem *zpoždění*.

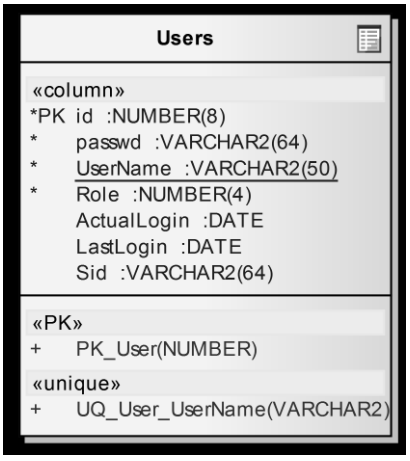
Tabulka `ChangeScope` uchovává možná měřítka provedených změn. Jsou jimi stav *lokální* a *globální*. O uživatelských oprávněních a těchto měřítkách pojednává blíže kapitola 3.6.2. UML diagramy tabulek viz obrázek 18.



Obrázek 18 – Tabulky ChangeScope a ChangeType

Tabulka Users

Tabulka `Users` uchovává základní informace o uživatelích. Z hlediska cílového zaměření aplikace stačí uchovávat pouze unikátní uživatelské jméno, heslo a číselné vyjádření role. Informativní charakter mají položky `ActualLogin` a `LastLogin` uchovávající informaci o poslední návštěvě uživatele. Dále je uchována informace o aktuálním tzv. session ID, které reprezentuje unikátní alfanumerický řetězec, sloužící pro ověření přihlášení uživatele v prohlížeči, které je platné po dobu 24 hodin. Session ID je generováno na základě aktuálního časového razítka a přihlašovacího jména uživatele a následně je vypočítán MD5 hash. UML diagram tabulky viz obrázek 19.



Obrázek 19 – Tabulka Users

3.6 Implementace aplikace

V následujících kapitolách budou představeny důležité implementační pasáže.

3.6.1 Import dat

Vytvořená aplikace pracuje s reálnými daty, která byla poskytnuta od pracovníka SŽDC. Data pochází z interního informačního systému ČD Kango a vztahují se ke stanici Praha, hl. n. Díky poskytnutým zdrojovým datům a originálu POK (3) a (11) byla část aplikace přímo uzpůsobena pro tuto ŽST. Nejinak tomu bylo i u procesu importu zdrojových dat do databáze. V případě potřeby použití této aplikace pro jinou ŽST by tedy bylo nutné uzpůsobit část aplikace pro konkrétní importovaná data a upravit grafickou část aplikace (např. příjezdové a odjezdové směry).

Data určená k importu do aplikace, jsou exportována z IS Kango. Jsou ve formátu dBase, který je určený pro uchování strukturovaných dat pomocí souboru s příponou DBF. Díky problémům s kódováním češtiny při importu přímo z tohoto souboru byl implementován import textových dat oddělených středníkem, tedy ze souboru s příponou CSV. Data z dBase formátu lze na CSV jednoduše převést pomocí tabulkového procesoru.

Struktura importovaných dat

Ve zdrojovém souboru se nachází 18 sloupců s daty, přičemž použity byly pouze následující důležité a nezbytné pro sestavení POK:

- *Druh* – označuje typ vlaku, tedy např. Os, R, Pn, Ex apod.[2]
- *Číslo* – unikátní číslo označující daný vlak.
- *Kolej* – pobytová kolej v ŽST; tento údaj je nezbytný pro vykreslení vlaku v POK.
- *Příjezd* – příjezd vlaku do ŽST; tento údaj není pro vlak povinný v případě, že vlak v ŽST jízdu začíná.
- *Pobyt* – délka pobytu vlaku na koleji; určuje „délku tvaru“ vlaku v POK.
- *Odjezd* – odjezd vlaku ze ŽST; tento údaj není pro vlak povinný v případě, že vlak v ŽST jízdu končí.
- *Poznámka* – popisu se věnuje kapitola 3.6.1.
- *Název* – název vlaku, není povinným údajem.

[2] Na této úrovni byla provedena selekce vlaků, které nejsou pro sestavení POK potřebné (Mn, Lv).

- *PředDopr* – předchozí ŽST, uvedena pouze v případě, že je uveden i čas příjezdu. Určuje „tvar“ vlaku – směr šipky.
- *NáslDopr* – následující ŽST, uvedena pouze v případě, že je uveden i čas odjezdu. Určuje „tvar“ vlaku – směr šipky.
- *Délka* – délka vlaku; nepovinný údaj s informativním charakterem.
- *Hmotnost* – hmotnost vlaku; nepovinný údaj s informativním charakterem.
- *Řada* – řada hnacího vozidla; nepovinný údaj s informativním charakterem.

Detekce kalendáře vlaku

V importovaných datech se vyskytuje i sloupec s názvem Poznámka, který uchovává ve formě textového řetězce dodatečné informace o vlaku, jako jsou:

- *PP* – vlak jedoucí podle potřeby,
- *Obs* – jízda po obsazené koleji ve stanici,
- *PosPr* – posilový vlak.

Většinu dat ve sloupci *Poznámka* však tvoří informace o tom, kdy ve smyslu kalendářního období má vlak jet. Pokud poznámka neobsahuje žádné informace, vlak jede pravidelně každý den po celé období GVD.

Struktura poznámky upřesňující kalendář vlaku je dost individuální a strojové zpracování bylo dost pracné. Tyto informace totiž nemají strojově zpracovatelný formát, protože jsou vkládány pracovníkem ČD/SŽDC. Pro ilustraci obsahu poznámky je uveden příklad upřesňující kalendář vlaku 19051:

```
jede v !6 a !+ do 17.III. a od 2.XI., od 23.III. do 26.X. jede v !6 a 1.,
8.V., 5.VII., nejede 6.VII./zavádějte v !6 a !+
```

Poznámka může obsahovat piktogramy v textové formě, které se běžně používají v jízdních řádech. Před tímto piktogramem je vždy uveden znak vykřičníku následovaný zástupným znakem, viz tabulka 3.

Tabulka 3 – Piktogramy obsažené v importovaných datech

Obvyklý piktogram	Zástupná forma	Význam
†	!+	Jede v neděli a státem uznané svátky.
⚒	!x	Jede v pracovní den.
⑥	!6	Jede v sobotu.[3]

Pro rozpoznávání poznámek pro kalendář vlaku byl implementován heuristický algoritmus, díky kterému je možné rozpoznat většinu poznámek a převést je tak na bitovou reprezentaci. Jednotlivé věty poznámek jsou parsovány pomocí oddělovačů (znaky mezera, lomítko a čárka) a tím vzniknou tokeny pro jednotlivou poznámku. Tokeny jsou postupně procházeny. Důležitou částí algoritmu je určování, o jaký druh tokenu (klíčového slova) jde. Jsou rozlišována klíčová slova `jede` (popř. `zavádějte`) a `nejede`, která určí následný proces nastavování kalendáře. Za tímto klíčovým slovem následují upřesňující informace, kdy má/nemá vlak jet. Tyto výrazy mohou být psány výše uvedenými piktogramy anebo mohou být použity zpřesňující intervaly. Pokud vlak v daný den kalendáře jede, má nastavenou logickou „1“ – v počátku je vlaku přiřazen prázdný kalendář a jsou nastavovány jednotlivé dny (bity s hodnotou logické „1“). V opačném případě, kdy je poznámkou stanoveno, že vlak nejede, je vlaku nastaven plný kalendář (jede každý den) a následně jsou daným dnům nastavovány bity na hodnotu logické „0“. V poznámce se však může vyskytnout i kombinace typu „vlak jede v pracovní dny, nejede 1. a 8. 5“. Poté je třeba nastavit všechny pracovní dny do prázdného kalendáře a následně použít operaci XOR pro zrušení dnů v podmnožině, kdy vlak má jet.

Pro oba typy klíčových slov byl implementován mechanizmus pro rozpoznání s využitím především regulárních výrazů, jako pomůcky při validaci a rozhodování, o jaký výraz se jedná. V časových intervalech se používají klíčová slova `od` a `do` a formát data je psán s využitím řecké abecedy na místě čísla měsíce, popř. může být uváděn interval s využitím piktogramů (typu „jede od neděle do čtvrtka“). Toho je využito při rozpoznání, že daný soubor výrazů určuje interval. Piktogramy jsou jednoduše určeny díky použitému prefixu, který je tvořen vykřičníkem (např. !x nebo !+).

Oba typy tokenů (určující, zda vlak jede nebo ne a zpřesňující tokeny) určí výsledný kalendář vlaku.

[3] Analogicky platí pro ostatní dny.

Pro ilustraci je uvedena metoda pro nastavení daného dne v roce do bitové reprezentace kalendáře. Parametrem metody je pořadové číslo dne v roce, kdy 1. leden je reprezentován číslem 1. Následně se vypočítá byte, v kterém se nachází testovaný den a pomocí offsetu a bitového AND se zjistí, zda je inkriminovaný den nastaven na hodnotu logické „1“. Kalendář je reprezentován jako pole 48 bytů (366 dní děleno 8 bity na byte plus jeden byte pro rok a jeden byte jako speciální příznak).

```
public boolean getDay(int dayOfYear) {
        dayOfYear--;
        return (calendar[1 + dayOfYear / 8] & (1 << (dayOfYear % 8))) != 0;
}
```

Stejným způsobem bitového přístupu ke kalendáři vlaku je realizováno nastavování jednotlivých dní, viz následující fragment zdrojového kódu. Příznak `state` značí, zda bude nastavována logická „0“ nebo „1“.

```
public void setDay(int dayOfYear, boolean state) {
        dayOfYear--;
        int offset = 1 + dayOfYear / 8;
        int bite = 1 << (dayOfYear % 8);

        if (state) {
            calendar[offset] = (byte) (calendar[offset] | bit);
        } else {
            calendar[offset] = (byte) (calendar[offset] & ~bit);
        }
}
```

Pro jednoduchou manipulaci s českým kalendářem byla implementována třída s pomocnými metodami, díky kterým je např. možné určit pro aktuální rok všechny pracovní dny, svátky (včetně plovoucího data Velikonoc) nebo dané dny v týdnu, převést je na bitovou reprezentaci a následně uložit jako kalendář pro daný vlak.

Některé složitější a individuálně specifické případy reprezentace kalendáře nebyly rozpoznávány a byl nastaven kalendář bez tohoto dodatečného upřesnění. Přesný kalendář ve formě poznámky je možné zobrazit v detailu vlaku jak v modulu *POK*, tak i v modulu *Vlaky*.

Díky většině rozpoznaných kalendářů vlaků lze v aplikaci vlaky filtrovat dle data, což je zejména z hlediska přehlednosti velkým přínosem. Implicitně se pak v modulu *POK* zobrazují vlaky pro konkrétní den (s možností zobrazit libovolný den nebo zobrazit všechny vlaky).

3.6.2 Uživatelské role

Jedním z nefunkčních požadavků na aplikaci je práce s uživatelskými rolemi ve třech úrovních. Všichni uživatelé s jakoukoli rolí jsou přihlášenými uživateli systému, tedy mají přiřazeno uživatelské jméno a heslo. Oprávnění jednotlivých rolí nejlépe vystihuje tabulka 4, kde L a G značí úroveň přístupu[4].

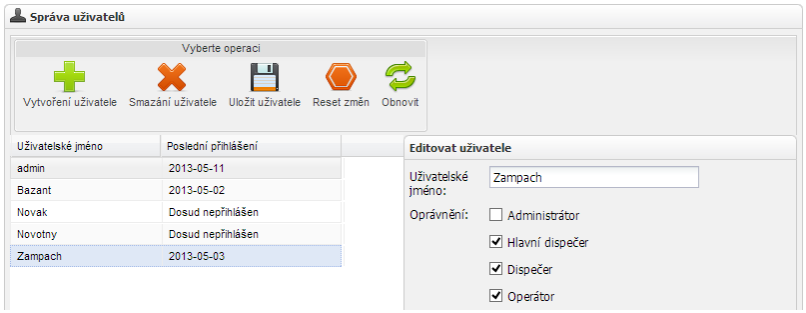
Tabulka 4 – Oprávnění uživatelských rolí

Číselný kód	**Název role**	**POK**		**Seznam vlaků**	**Editace vlaků**		**Editované vlaky**	**Editor kolejí**	**Změna hesla**	**Správa uživatelů**	**Import dat**
Přístup		L	G	–	L	G	–	G	–	–	–
1	Administrátor	✗		✗	✗		✗	✗	✓	✓	✗
2	Hl. dispečer	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
4	Dispečer	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗
8	Operátor	✓		✓	✗		✗	✗	✓	✗	✗

Dodatečnou roli tvoří Administrátor, který má přístup pouze ke správě uživatelů. Uživatelské role jsou koncipovány tak, že jeden uživatel může disponovat libovolnými rolemi. Např. uživatel Hlavní dispečer může mít přidělenu i roli Administrátora. Uživatelské role jsou reprezentovány v databázi pomocí čísla (sloupec Číselný kód v tabulce 4), které vyjadřuje roli. S touto reprezentací se dá dobře pracovat pomocí bitových operací a zjišťovat tak, které role má uživatel přidělené.

K popsanému řešení bylo implementováno korespondující řešení grafického rozhraní[5], kde lze uživateli přiřadit libovolnou kombinaci uživatelské role, viz obrázek 20.

[4] L – lokální úroveň přístupu, G – globální úroveň přístupu.
[5] Komponenta pro správu uživatelů je dostupná pouze uživateli, který má roli Administrátor.

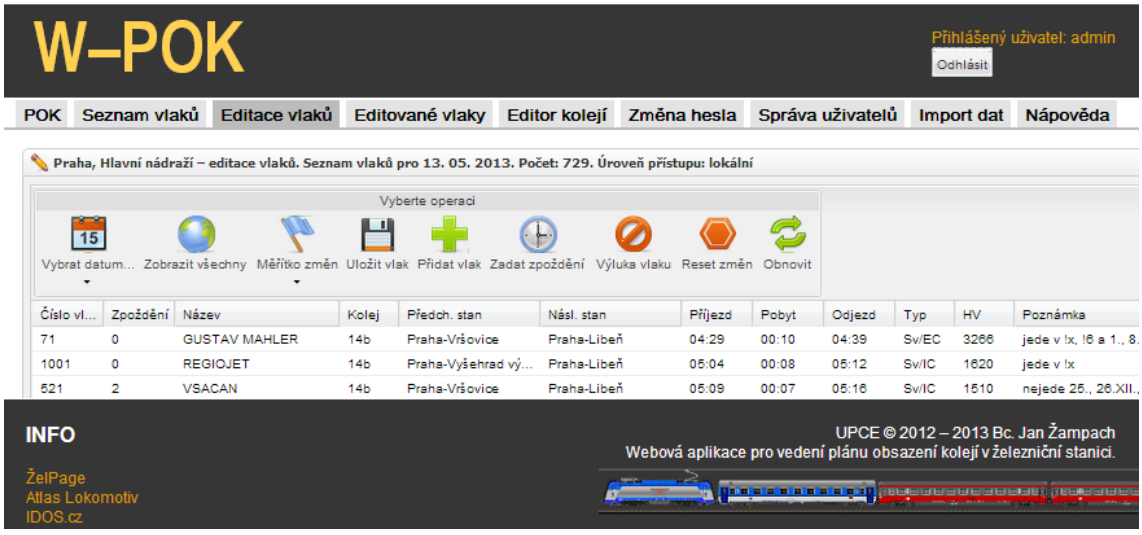


Obrázek 20 – Rozhraní pro správu uživatelů

3.6.3 Návrh grafického rozhraní

Návrh uživatelského rozhraní vychází z požadavků na aplikaci. Je do jisté míry omezen skutečností, že uživatelskou část aplikace tvoří webová aplikace. Pro návrh tedy byly využity kaskádové styly CSS a rozvržení pomocí HTML elementů. Design aplikace byl volen s důrazem na jednoduchost, přehlednost a použitelnost.

Celkový náhled na aplikaci v okně prohlížeče zobrazuje obrázek 21. Šablona je koncipována tak, aby se zobrazovala na 100 % šířky plochy monitoru uživatele. V horní části šablony se nachází název aplikace (W-POK) a jednoduchý uživatelský panel, informující o přihlášeném uživateli a tlačítko pro odhlášení z aplikace. Následuje seznam dostupných záložek komponenty (tzv. *tabbed panel*), mezi kterými se lze přepínat a zobrazovat tak jednotlivé uživatelské komponenty. Uživateli zobrazené záložky jsou závislé na jeho roli – uživatelské rozhraní je sestavováno dynamicky až po přihlášení uživatele do aplikace. Jednotlivé komponenty budou popsány v dalších podkapitolách. Šablonu uzavírá jednoduchá patička s informacemi o autorovi.



Obrázek 21 – Celkový náhled na aplikaci v okně prohlížeče

Modul Vlaky

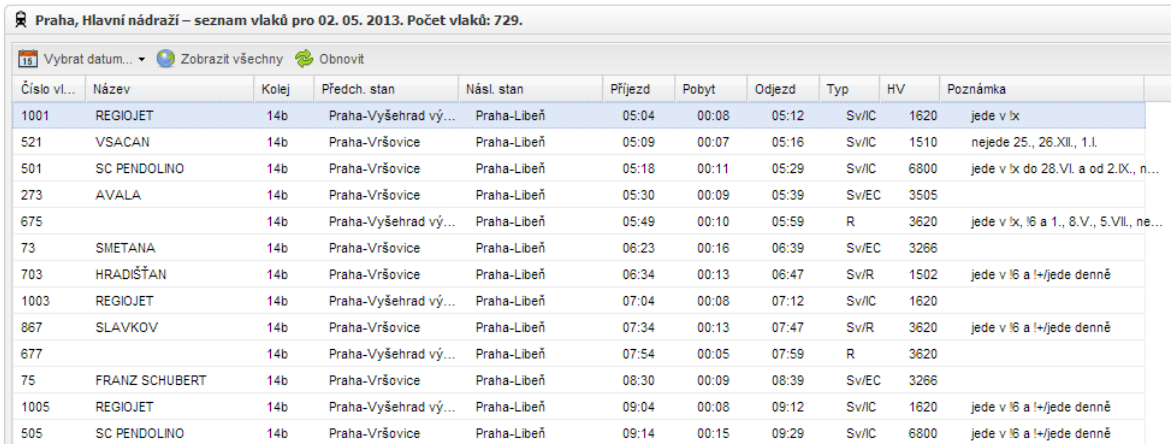
Komponenty interaktivních tabulek v jednotlivých záložkách jsou vytvořeny pomocí knihovny Ext GWT (též se používá zkrácené označení GXT). Tato knihovna je rozšířením GUI GWT a nabízí celou paletu uživatelsky přívětivých komponent a widgetů, díky kterým lze vytvářet RIA aplikace. GXT využívá GWT kompilátor a díky tomu se dá snadno integrovat do GWT aplikace – do projektu stačí přiložit potřebné knihovny a kaskádové styly pro správné zobrazování komponent. Po zkompilování vzniká optimalizovaný HTML a JavaScript kód (viz kapitola 2.2.1), jenž je podporován napříč všemi moderními prohlížeči. Díky GXT získává GWT mnohem širší možnosti využití, samotné GWT bez této knihovny nenabízí ani zdaleka tak širokou paletu GUI komponent.

Knihovna GXT je produktem společnosti *Sencha* a použitá verze 2.2.5 je vydána pod open-source licencí GNU GPL verze 3. (12)

Pro využití v aplikaci byly vyloženě vhodné komponenty s názvem *Grid* – tabulky pro zobrazování dat. Nabízí široké možnosti nastavení třídění, filtrování, nastavení formátu zobrazení a především možnost editace. Dále byly z knihovny GXT využity formulářové komponenty a tzv. *Toolbar* prvky, díky kterým byly vytvořeny uživatelsky přívětivé nástrojové lišty.

Modul *Vlaky* je v aplikaci tvořen seznamem importovaných vlaků, kde jsou přehledně zobrazeny všechny jejich parametry. Tato data v podstatě tvoří datovou základnu pro sestavení grafického plánu obsazení. Seznam vlaků lze řadit dle různých parametrů, kdy nejpoužívanější asi bude řazení dle času příjezdu nebo dle dané pobytové koleje a vyhledávat (zadáním čísla vlaku nebo názvu).

Základní komponenta *Vlaky – Seznam vlaků* je dostupná uživatelům se základní rolí oprávnění (Operátor) a umožňuje pouze prohlížení (s možností filtrace a řazení), viz obrázek 22. V záhlaví komponenty je uveden název komponenty, aktuální datum, pro které je zobrazen seznam a počet načtených vlaků.



Praha, Hlavní nádraží – seznam vlaků pro 02. 05. 2013. Počet vlaků: 729.

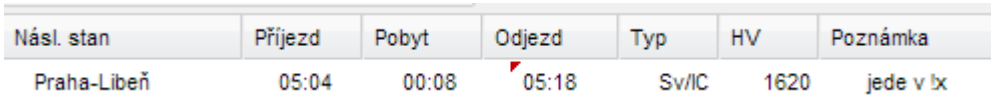
Vybrat datum... Zobrazit všechny Obnovit

Číslo vl...	Název	Kolej	Předch. stan	Násl. stan	Příjezd	Pobyt	Odjezd	Typ	HV	Poznámka
1001	REGIOJET	14b	Praha-Vyšehrad vý...	Praha-Libeň	05:04	00:08	05:12	Sv/IC	1620	jede v !x
521	VSACAN	14b	Praha-Vršovice	Praha-Libeň	05:09	00:07	05:16	Sv/IC	1510	nejede 25., 26.XII., 1.I.
501	SC PENDOLINO	14b	Praha-Vršovice	Praha-Libeň	05:18	00:11	05:29	Sv/IC	6800	jede v !x do 28.VI. a od 2.IX., n...
273	AVALA	14b	Praha-Vršovice	Praha-Libeň	05:30	00:09	05:39	Sv/EC	3505	
675		14b	Praha-Vyšehrad vý...	Praha-Libeň	05:49	00:10	05:59	R	3620	jede v !x, !6 a 1., 8.V., 5.VII., ne...
73	SMETANA	14b	Praha-Vršovice	Praha-Libeň	06:23	00:16	06:39	Sv/EC	3266	
703	HRADIŠŤAN	14b	Praha-Vršovice	Praha-Libeň	06:34	00:13	06:47	Sv/R	1502	jede v !6 a !+/jede denně
1003	REGIOJET	14b	Praha-Vyšehrad vý...	Praha-Libeň	07:04	00:08	07:12	Sv/IC	1620	
867	SLAVKOV	14b	Praha-Vršovice	Praha-Libeň	07:34	00:13	07:47	Sv/R	3620	jede v !6 a !+/jede denně
677		14b	Praha-Vyšehrad vý...	Praha-Libeň	07:54	00:05	07:59	R	3620	
75	FRANZ SCHUBERT	14b	Praha-Vršovice	Praha-Libeň	08:30	00:09	08:39	Sv/EC	3266	
1005	REGIOJET	14b	Praha-Vyšehrad vý...	Praha-Libeň	09:04	00:08	09:12	Sv/IC	1620	jede v !6 a !+/jede denně
505	SC PENDOLINO	14b	Praha-Vršovice	Praha-Libeň	09:14	00:15	09:29	Sv/IC	6800	jede v !6 a !+/jede denně

Obrázek 22 – Základní komponenta pro zobrazení seznamu vlaků

Pokročilejší komponenta *Vlaky – Editor vlaků* už umožňuje uživatelům s vyšším stupněm oprávnění vlaky editovat v tom rozsahu, jak bylo stanoveno v požadavcích. Komponenta má v nástrojové liště seznam intuitivních tlačítek, jejichž funkce plyne z jejich názvu. Pro úplnost lze jen shrnout seznam a možnosti úprav, jež lze provádět v této komponentě.

Jednotlivé buňky tabulky jsou editovatelné[6] a přímo po kliknutí do buňky lze zapsat novou hodnotu[7], popř. hodnotu vybrat z roletové nabídky (např. změna pobytové koleje). Po provedení libovolné editace je editované políčko označeno červeným bodem, který znázorňuje, že provedené změny zatím nebyly uloženy do databáze a nejprve je nutné kliknout na tlačítko Uložit vlak. Zároveň není možné před uložením změn editovat jiný vlak. Provedená, neuložená změna, viz obrázek 24.



Násl. stan	Příjezd	Pobyt	Odjezd	Typ	HV	Poznámka
Praha-Libeň	05:04	00:08	05:18	Sv/IC	1620	jede v !x

Obrázek 23 – Zvýraznění neuložených změn v modulu Vlaky – Editace vlaků

Při vložení nového vlaku je přidán nový řádek do tabulky a vyplnění atributů vlaku probíhá naprosto obdobným způsobem, jako při editaci atributů.

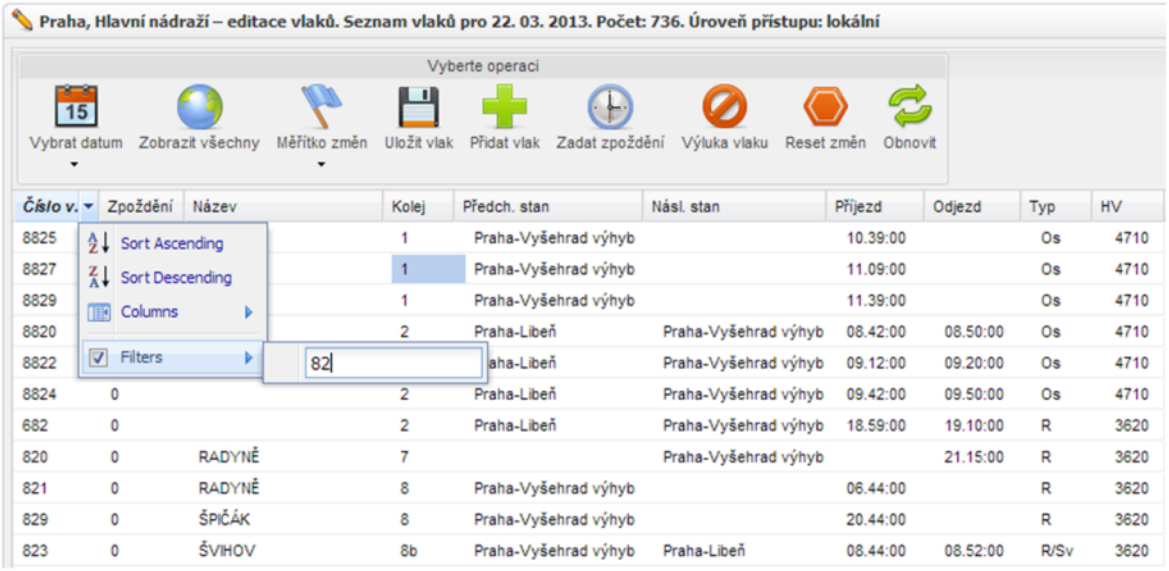
Zadávání zpoždění probíhá přes jednoduchý formulář, kde je uživatel vyzván k zadání zpoždění v řádu minut. Před zadáním zpoždění je třeba vybrat konkrétní vlak kliknutím na jakékoli jeho pole.

[6] Pouze základní atributy vlaku, nelze např. změnit název vlaku.
[7] Při zadávání hodnot do pole je hodnota kontrolována, zda je zapsána ve správném formátu. Nelze tedy uložit např. nevalidní časový údaj 24:62.

Přiřazení výluky vlaku lze jednoduše vykonat po výběru vlaku a následném stisku tlačítka Výluka vlaku. Vlak bude ze seznamu jedoucích vlaků odebrán a bude zařazen do seznamu editovaných vlaků, kde lze jeho výluku následně opět zrušit.

Všechny popsané změny lze provádět ve dvou úrovních[8] (měřítkách) – lokální a globální. Měřítko je možné změnit v nástrojové liště. Implicitně je nastaveno na lokální úroveň a o aktuálně nastaveném měřítku změn je uživatel informován v záhlaví modulu.

Náhled editovatelné komponenty *Vlaky – Editace vlaků*, kde je znázorněno filtrování dle čísla vlaku, viz obrázek 24.



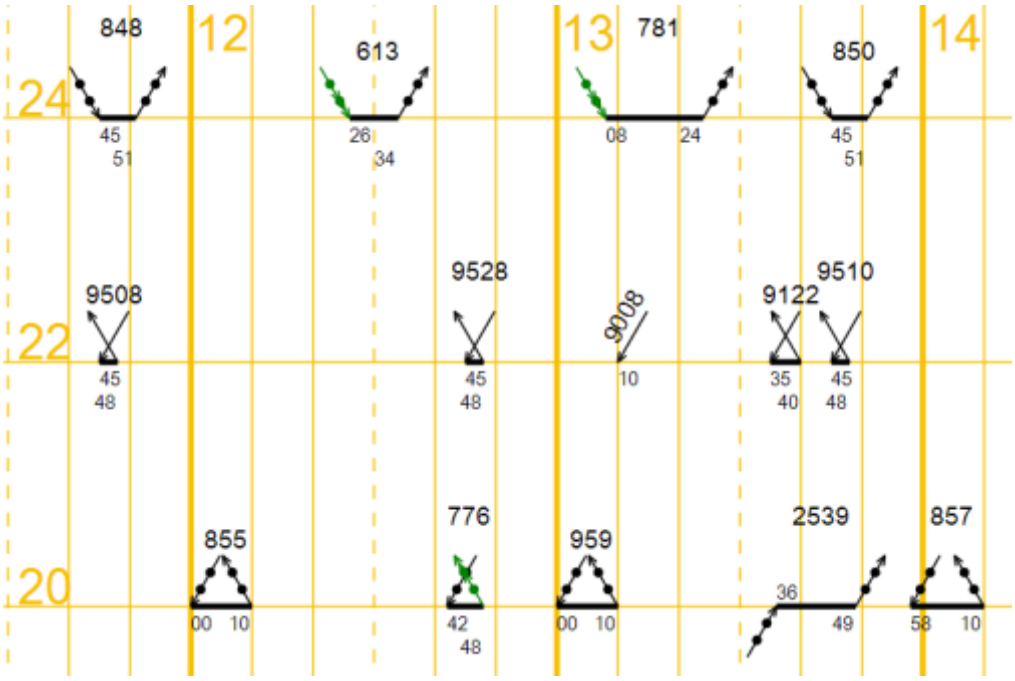
Obrázek 24 – Interaktivní editovatelná komponenta s nástrojovou lištou

Modul POK

Nejdůležitější záložka uživatelského rozhraní je nazvaná POK a ukrývá samotný vykreslený plán obsazení kolejí. Jeho část je umístěna v příloze E. Modul sestává z uživatelské lišty (filtr dle data, zobrazení všech vlaků a tlačítko pro obnovení) a vektorového obrázku. Zobrazení lze díky vektorovému formátu libovolně přibližovat (nativní funkcí prohlížeče). Samotný plán má rozměry cca 4500 × 3500 px a v prohlížeči se lze v plánu snadno orientovat pomocí rolování v horizontální i vertikální poloze (nativně pomocí kurzorových šipek nebo svislého a vodorovného posuvníku v prohlížeči). Rozměr plánu byl zvolen empiricky a při 100% přiblížení jsou vykreslené informace dobře rozpoznatelné.

[8] Platí pouze pro uživatele s rolí Hlavní dispečer, pro roli Dispečer je k dispozici pouze lokální měřítko.

Při takto velkém rozlišení se plán na běžném monitoru nikdy nevykreslí celý, a proto nemohou být popisky kolejí a hodin umístěny na plánu staticky v okrajích. Z toho důvodu důležitým prvkem, který zabezpečuje dobrou orientaci v plánu, jsou plovoucí popisky, které zůstávají i při odrolování okna (horizontálně i vertikálně) ve své poloze (okraje okna) a uživatel tak má stále přehled o souřadnicích, viz obrázek 25.



Obrázek 25 – Plovoucí popisky kolejí (vertikálně) a hodin (horizontálně)

Grafická podoba vychází z reálné předlohy, kterou byl POK pro Prahu, hl. n, viz (3) a (11). Byly převzaty shodné barvy pro hlavní mřížku, stejný styl dělení hlavní i vedlejší mřížky. Logicky odlišené koleje s postfixem *b* byly vykresleny čárkovanou čárou.

Jako interaktivní prvek grafického plánu byl implementován popisek vlaku s detailními informacemi, který se uživateli zobrazí při najetí kurzorem nad daný vlak. Zobrazí se všechny důležité (a dostupné) informace o vlaku, viz obrázek 26.

Obrázek 26 – Popisek vlaku s detailními informacemi

Možnosti dynamických změn plánu obsazení kolejí

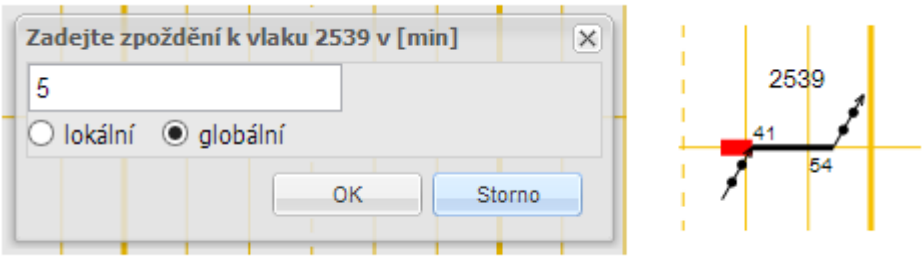
Na základě požadavků je v aplikaci implementována možnost dynamicky zadávat změny prostřednictvím editační části modulu *Vlaky*. Aby byla ale práce s aplikaci příjemnější a svižnější, byla i přímo do hlavního modulu *POK* zanesena základní možnost editace. Po kliknutí levým tlačítkem myši na vybraný vlak se uživateli[9] zobrazí kontextová nabídka, viz obrázek 27.

Obrázek 27 – Kontextová nabídka pro rychlou editaci vlaku

Vlaku lze zadat jednoduše a pohotově zpoždění skrz jednoduchý formulář. Pokud vlak již zpoždění má přiřazené, pak se v poli pro zadávání nové hodnoty zobrazí hodnota aktuální, kterou lze modifikovat. Po zadání zpoždění v řádu minut může ještě uživatel s rolí Hlavní dispečer vybrat, zda chce provést změny na lokální nebo globální úrovni. U uživatele s rolí

[9] Uživatel musí mít roli Dispečer nebo Hlavní dispečer, jinak nemá možnost zobrazit tuto kontextovou nabídku.

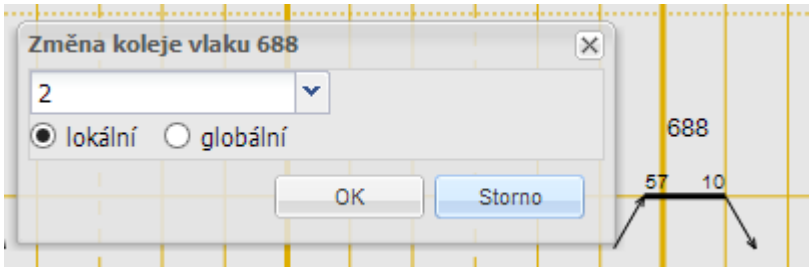
Dispečer je tato možnost implicitně nastavena na lokální úroveň a nelze ji změnit. Formulář pro zadání zpoždění a výsledek akce, viz obrázek 28.



Obrázek 28 – Formulář pro zadání zpoždění vlaku v modulu POK

Grafické řešení přidání zpoždění vlaku je řešeno formou přidání výrazné červené linky před „tvar“ vlaku, kdy tato linka začíná v pozici plánovaného příjezdu a končí v době předpokládaného příjezdu, následovaná nemodifikovaným „tvarem“ vlaku[10]. V praxi se může stát, že opožděný vlak bude mít modifikovanou (zkrácenou) dobu pobytu ve stanici. Tuto dobu však nelze exaktně určit (u vlaků jsou plánovány technologické operace a nelze tedy např. jednoduše získat čas odjezdu pouhým odečtením doby pobytu a doby zpoždění vlaku). Proto má uživatel možnost změnit vlaku dobu pobytu na koleji[11]. Proces zadání doby pobytu je obdobný zadání zpoždění vlaku.

Další položkou rychlé kontextové nabídky, která nemůže chybět, je *Změna pobytové koleje vlaku*. Po výběru této nabídky má uživatel možnost vybrat si ze seznamu (pouze aktuálně dostupných) kolejí tu, na kterou bude vlak v rámci POK přemístěn. Dále nechybí ani výběr měřítka provedené změny. Snímek kontextové nabídky, viz obrázek 29.



Obrázek 29 – Formulář pro zadání změny pobytové koleje vlaku

Všechny modifikace v POK, provedené pomocí výše uvedené kontextové nabídky v modulu *POK*, jsou taktéž zaznamenávány do provedených změn a tyto změny lze vracet

[10] Jsou pouze aktualizovány hodnoty času příjezdu a odjezdu.
[11] Pouze v případě, že vlak má definovanou dobu pobytu.

zpět. I na tuto funkcionalitu bylo pamatováno v kontextové nabídce. U modifikovaného vlaku, lze zvolit položku *Reset poslední změny* – nehledě na typ provedené změny je tato odrolována zpět. Takto lze resetovat postupně všechny provedené změny u daného vlaku (postupuje se formou historie změn).

Řešení konfliktů vlaků

Po jakékoli provedené změně v plánu proběhne kontrola, zda nedošlo ke konfliktu vlaků. Konflikt je v aplikaci definován jednoduchým způsobem – nastane, pokud se na totožné koleji kryjí doby pobytu více vlaků. Další logika pro řešení konfliktů nebyla v aplikaci implementována, protože by překračovala její rámec. Je tak plně v režii dispečera. Následující fragment kódu zachycuje způsob, kterým jsou procházeny vlaky při kontrole konfliktu:

```
public static TrainDTO checkSafeOperation(MultiDTO multi, TrainDTO queryTrain) {
        // projde vsechny vlaky na stejne koleji a zkontroluje,
        // zda doslo ke konfliktu
        for (TrainDTO train : multi.getMap().get(queryTrain.getRails())) {
            if (!train.getTrainnumber().equals(queryTrain.getTrainnumber())) {
                if (isInConflict(train, queryTrain)) {
                    return train;
                }
            }
        }

        return null;
}
```

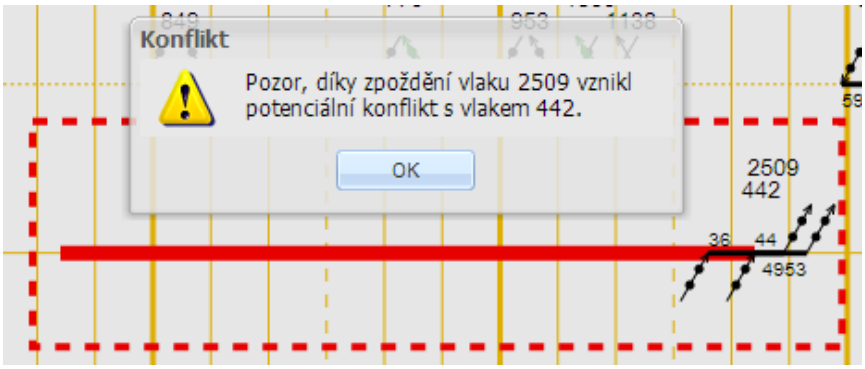
Vlaky a koleje jsou na straně klienta v aplikaci uchovány v objektu `MultiDTO`, který disponuje datovou strukturou hash mapa, kde klíč je tvořen identifikátorem koleje a hodnotu pak tvoří seznam vlaků na koleji. Při vyhledávání konfliktu jsou tak vždy procházeny pouze vlaky na testované koleji.

Metoda `boolean isInConflict(TrainDTO t1, TrainDTO t2)` pak již vrací výsledek, zda jsou dva testované vlaky mezi sebou v konfliktním stavu. Tento stav nastává právě tehdy, když je splněna podmínka:

```
(Start_T1 <= End_T2) AND (End_T1 >= Start_T2),
```

kde `Start_` znamená počátek pobytu vlaku na koleji a `End_` konec pobytu vlaku na koleji.

Na detekovaný konflikt je uživatel upozorněn primárně formou dialogového okna a sekundárně v modulu *POK* zvýrazněním konfliktu pomocí čárkované červené čáry kolem konfliktního vlaku, viz obrázek 30.

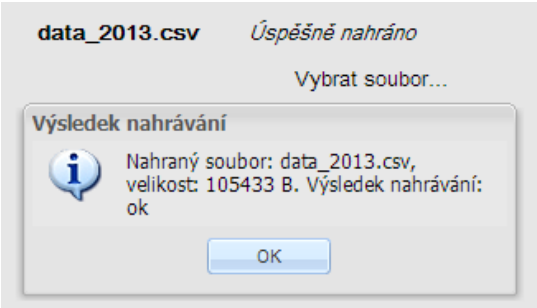


Obrázek 30 – Detekovaný konflikt v modulu POK po zadání zpoždění vlaku

Pro úplnost budou v následujících podkapitolách ve stručnosti zmíněny zbylé záložky grafického rozhraní.

Grafické rozhraní pro import dat

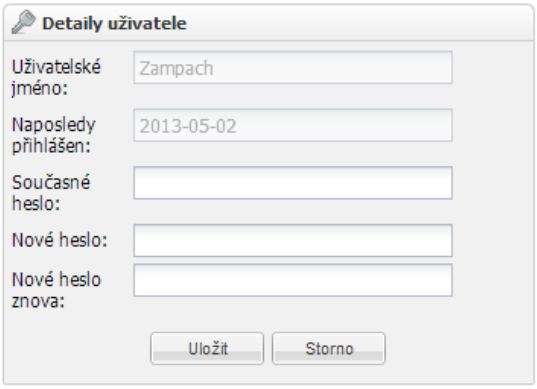
Uživatel s rolí Hlavní dispečer má po přihlášení do aplikace možnost importovat zdrojová data. Formátu zdrojových dat, způsobu importu a zpracování se věnují kapitoly 3.6.1 a 3.8.3. Záložka *Import dat* ukrývá jednoduché grafické rozhraní, které poskytuje uživateli možnost po kliknutí na odkaz vybrat soubor z pevného disku počítače a nahrát jej na server. Po úspěšném importu dat je uživateli zobrazena informativní hláška (obrázek 33) a záznam o nahraném souboru je uveden v seznamu nahraných souborů. V případě potřeby je možné import dat zopakovat. Pokud nahrávání nebo import dat selže, nebo je vybrán špatný typ souboru, uživatel je rovněž informován patřičnou chybovou hláškou.



Obrázek 31 – Informace o nahrávání souboru s daty pro POK

Grafické rozhraní uživatelského účtu

Jak již bylo uvedeno dříve, aplikace poskytuje pouze jednoduchou správu uživatelských účtů. S tím koresponduje i jednoduché grafické rozhraní pro editaci profilu uživatele a omezuje se pouze na možnost změny hesla, viz obrázek 32. Zbylé změny (editace rolí, mazání a přidávání uživatelů) jsou ponechány v kompetenci uživateli s rolí Administrátor.



Obrázek 32 – Grafické rozhraní pro editaci hesla uživatele

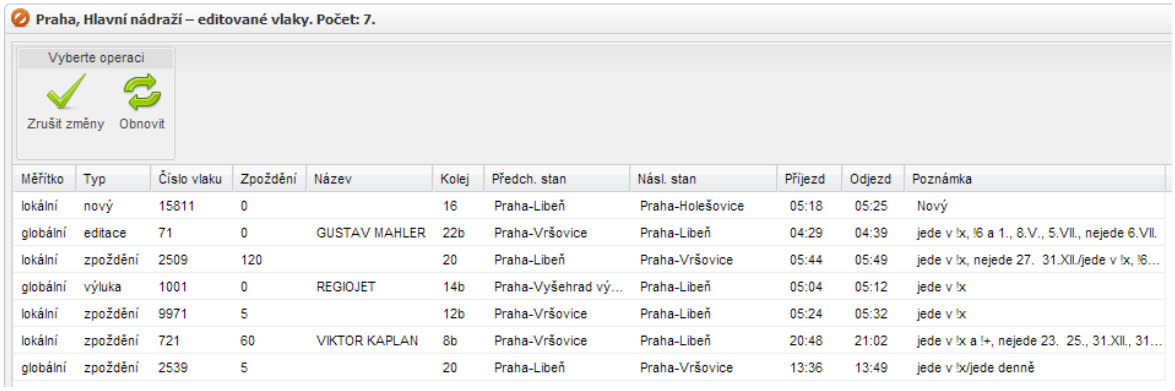
3.6.4 Evidence editovaných změn

Jak již bylo zmíněno v kapitole 3.5, aplikace je navržena tak, aby umožnila evidovat všechny provedené změny v POK a v případě potřeby umožňuje vrátit veškeré změny zpět do původního stavu. Provedená změna je vždy vázána na uživatele, který změnu realizoval. To znamená, že pouze majitel změny může tuto smazat.

Provedené změny jsou evidovány v modulu *Vlaky – Editované vlaky*. Zde uživatel najde veškeré svoje provedené změny v POK. Jsou rozlišovány čtyři typy změn:

- vložení nového vlaku,
- editace libovolného atributu vlaku,
- zadání zpoždění k vlaku a
- přiřazení výluky k vlaku.

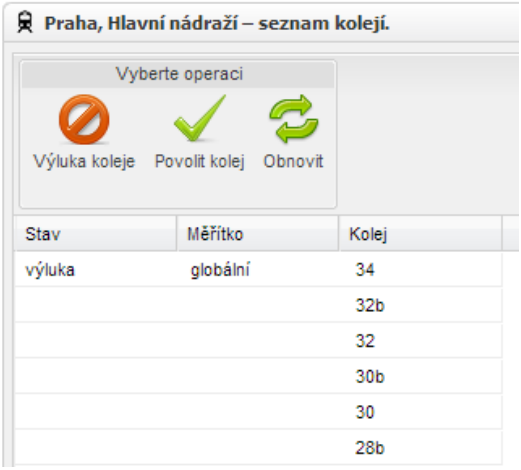
Modul pro evidenci změn je realizován tak, že zobrazuje poslední provedené změny u daného vlaku. To znamená, že pokud má vlak evidováno více změn, je zobrazena pouze poslední provedená změna. Po odrolování poslední změny je opět zobrazena ta předchozí (zřetězený seznam provedených změn) atd. až do okamžiku, kdy vlak nemá evidovánu žádnou změnu. Náhled grafického rozhraní modulu pro správu editovaných vlaků znázorňuje obrázek 33.



Měřítko	Typ	Číslo vlaku	Zpoždění	Název	Kolej	Předch. stan	Násl. stan	Příjezd	Odjezd	Poznámka
lokální	nový	15811	0		16	Praha-Libeň	Praha-Holešovice	05:18	05:25	Nový
globální	editace	71	0	GUSTAV MAHLER	22b	Praha-Vršovice	Praha-Libeň	04:29	04:39	jede v !x, !6 a 1., 8.V., 5.VII., nejede 6.VII.
lokální	zpoždění	2509	120		20	Praha-Libeň	Praha-Vršovice	05:44	05:49	jede v !x, nejede 27. 31.XII./jede v !x, !6...
globální	výluka	1001	0	REGIOJET	14b	Praha-Vyšehrad vý...	Praha-Libeň	05:04	05:12	jede v !x
lokální	zpoždění	9971	5		12b	Praha-Vršovice	Praha-Libeň	05:24	05:32	jede v !x
lokální	zpoždění	721	60	VIKTOR KAPLAN	8b	Praha-Vršovice	Praha-Libeň	20:48	21:02	jede v !x a !+, nejede 23. 25., 31.XII., 31...
globální	zpoždění	2539	5		20	Praha-Libeň	Praha-Vršovice	13:36	13:49	jede v !x/jede denně

Obrázek 33 – Grafické rozhraní pro správu editovaných vlaků

Aplikace umožňuje provádět i výluky jednotlivých kolejí. V praxi je tato situace relativně málo častá a i z toho důvodu byla pro evidenci kolejí implementována jednoduchá, ale samostatná a jednoúčelová komponenta. Umožňuje pouze vést evidenci všech kolejí, kde uživatel vidí, která kolej je ve výluce[12] a přiřazovat kolejím výluku, popř. výluku rušit, viz obrázek 34. Pro přehlednost je seznam kolejí seřazen do stejného pořadí, jako je v POK. Výluku lze přiřadit pouze koleji, na které není v dané době evidován žádný vlak. Pokud tomu tak není, aplikace zobrazí uživateli chybovou hlášku, kde zobrazí aktuální počet evidovaných vlaků na dané koleji. Uživatel pak musí manuálně přemístit všechny vlaky z uvažované koleje jinam. Tento proces nebyl automatizován, protože podléhá specifickým pravidlům a je nutné, aby ho provedla osoba znalá, tedy dispečer.



Stav	Měřítko	Kolej
výluka	globální	34
		32b
		32
		30b
		30
		28b

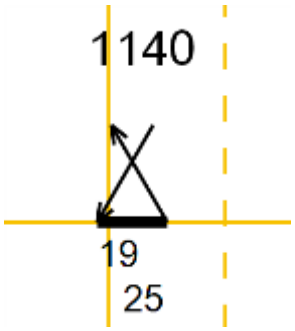
Obrázek 34 – Jednoduché grafické rozhraní pro evidenci kolejí

[12] Výluky kolejí jsou prováděny výhradně v globálním měřítku změn.

3.6.5 Automatické řešení grafických konfliktů v plánu obsazení kolejí

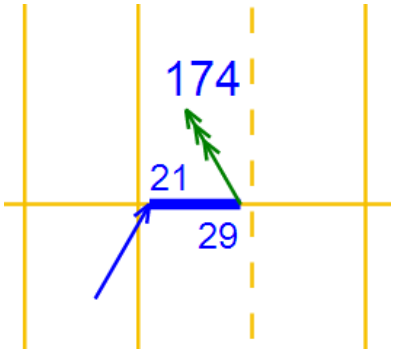
Při ruční tvorbě POK se zhotovitel řídí vlastním estetickým cítěním a jakýkoli grafický prvek intuitivně umístí tak, aby byl výsledný plán co nejvíce přehledný a plnil tak svoji roli. V automatickém generování plánu je situace složitější a nežádoucí grafické kolize objektů se hůře rozpoznávají. Za účelem co nejpřehlednějšího POK byly implementovány jednotlivé techniky, které sice nejsou v reálném POK zaneseny, ale v generovaném plánu zvyšují míru přehlednosti a jsou tak ve výsledku přínosné.

Jedním z problémů je překrývání minut času příjezdu a odjezdu v případě krátkého pobytu vlaku na koleji. V tom případě je popisek minut odjezdu posunut ve směru osy *y* dolů tak, aby se nepřekrýval s popiskem minut příjezdu, viz obrázek 35.



Obrázek 35 – Řešení grafické kolize minutové popisky vlaku

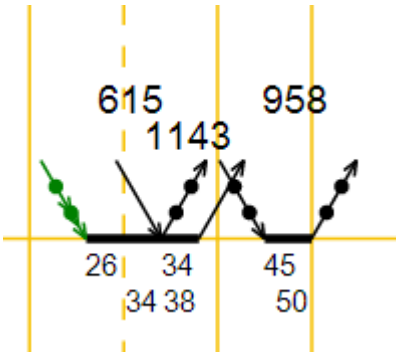
Dalším problémem byla kolize šipek (příjezdových a odjezdových směrů) s minutovými popisky v případě, kdy tyto jedna nebo obě šipky vedly odspodu (myšleno graficky) a překrývaly tak popisky minut. Řešením bylo umístit popisky minut na opačnou stranu osy *y* (pobytové koleje), než přichází šipka, viz obrázek 36.



Obrázek 36 – Řešení grafické kolize minut a směrů příjezdu nebo odjezdu

V některých časových intervalech se na frekventovaných kolejích vyskytuje velmi rušný provoz. Z toho důvodu se i v plánu vyskytují místa s vysokou koncentrací informací

a vznikají tak grafické kolize čísel vlaků, šipek a popisků minut. Jedním z možných řešení je implementovat proměnlivou výšku umístění popisku čísla vlaku tak, aby se do určité míry eliminoval překryv sousedních popisků. Obrázek 37 znázorňuje implementované řešení.



Obrázek 37 – Řešení grafické kolize překrývajících se čísel vlaků

Ani toto řešení ale není dokonalé a v plánu se vyskytují kolidující texty. Pro tento případ se hodí výše zmíněný popisek vlaku s detailními informacemi, který pomůže kolizi zpřehlednit.

3.7 Design aplikace

Při implementaci aplikace byly využity návrhové vzory a programovací techniky, jejichž účelem je řešit problémy standardizovaným a efektivním způsobem.

Pro připojení k databázi byl použit návrhový vzor Singleton, řadící se do typu vzorů vytvářející objekty. Tento vzor zajišťuje používání pouze jedné instance objektu pro přístup do databáze, viz následující fragment zdrojového kódu.

```
public class HibernateUtil {

    private static Session session;

    public static Session getSession() {
        if (session == null) {
            session = sessionFactory.openSession();
        }

        return session;
    }
}
```

Pro dynamickou sestavu celkového oprávnění uživatele dle číselné role, která je uložena v databázi, byl využit návrhový vzor Decorator. V cyklu je bitově vyhodnocována číselná role uživatele (kombinace – součty hodnot 1, 2, 4 a 8) a dle ní je výsledkem seznam

objektů, které reprezentují jednotlivé role daného uživatele. Dle uživatelských rolí je následně sestavováno výsledné GUI aplikace a je rozhodováno, zda je uživatel oprávněn k výkonu dané operace. Popsaný postup asi nejlépe vystihuje přiložený fragment kódu.

```
public static IPrivileges getPrivs(UserDTO user) {
        List<IPrivileges> prava = new ArrayList<IPrivileges>();
        for (int i = 0; i < role.length; i++) {
            if ((user.getRole() & (1 << i)) != 0) {
                prava.add(role[i]);
            }
        }

        return new RoleDecorator(prava);
}
```

Dále byla při práci s frameworkem Hibernate implicitně použita technika Lazy initialization, která zajišťuje odloženou inicializaci objektu do doby jeho prvního použití. Tím je dosaženo větší efektivnosti při práci s perzistentní vrstvou.

3.8 MVC architektura aplikace

Aplikace je stavěna na architektuře MVC. Jedná se rozdělení aplikace na datový model (model), vrstvu uživatelského rozhraní (view) a aplikační logiku (controller). Díky tomuto rozvrstvení má úprava libovolné komponenty minimální vliv na ostatní.

3.8.1 Model

Z důvodu využití frameworku Hibernate byly datové třídy v aplikaci rozděleny do dvou úrovní, a to entitní POJO třídy a DTO třídy.

POJO entitní třídy

První úroveň je tvořena entitními (tzv. POJO) třídami, které mapují entity z perzistentní vrstvy. Ty mají identické atributy jako entity v databázi a pouze běžné metody `get()`, `set()` a překryté `equals()`. Tyto POJO třídy jsou využívány pouze na straně serveru při manipulaci s perzistentní vrstvou, tedy řídící logikou aplikace. Objekty, které vznikly jako výsledek dotazu do databáze, jsou inicializovány až v momentě potřeby (Lazy loading), avšak za předpokladu, že je stále otevřena jedna instance spojení s databází (tzv. session, sezení). V případě zavření aktuální session a následnému přístupu k výsledkům dotazu vzniká výjimka `LazyInitializationException`. Z toho plyne, že není vhodné s těmito identickými datovými objekty, na něž je vázána persistentní vrstva, pracovat i na úrovni klientské části aplikace.

DTO třídy

Z výše uvedeného důvodu byla implementována další úroveň entitních tříd, a to tzv. data transfer objekty (DTO). Výhodou tohoto přístupu je úplná nezávislost aplikační vrstvy na prezenční vrstvě. DTO objekty v aplikaci slouží pro reprezentaci dat napříč klientskou a serverovou částí aplikace. Objekty jsou inicializovány na straně serveru z dat z odpovídajících entitních objektů (vázaných na perzistentní vrstvu) a následně serializovány a transportovány mechanismem RPC do klientské části aplikace. DTO objekty se liší od entitních především absencí vazby na perzistentní vrstvu a absencí některých atributů (např. identifikátor, použitý jako primární klíč v databázi). Naopak mohou disponovat některými atributy a metodami navíc (např. pomocné metody pro konverzi času apod.). (13)

3.8.2 View

Vrstvu uživatelského rozhraní tvoří primárně HTML šablona, nastylovaná pomocí kaskádových stylů CSS.

Do šablony je zasazena hlavní komponenta – *tabbed panel*, poskytující uživateli intuitivní orientaci v jednotlivých modulech aplikace prostřednictvím záložek. Jednotlivé záložky ukrývají samostatné logické části aplikace.

Zmíněné komponenty neobsahují řídící logiku aplikace, pouze se starají o správné zobrazení rozhraní pro konkrétního uživatele. Veškerá funkcionalita je implementována buď na straně serveru, nebo v controller třídách na straně klienta.

3.8.3 Controller

Controller třídy obsahují veškerou řídící logiku celé aplikace a zpracovávají události vzniklé podnětem uživatele. Třídy lze rozdělit na ty na straně serveru a na straně klienta.

Serverové řídící třídy se starají primárně o komunikaci s databází. Dle balíčků jsou rozděleny na třídy pro práci s vlaky (primárně dotazy do databáze a následná transformace dat na DTO objekty), manipulace se změnami vlaků. Obdobně třídy pro manipulaci s atributy vlaku (zejména koleje a evidence jejich změn). Dále řídící třídy pro import dat, s tím spojené řídící třídy pro manipulaci a výpočet kalendářů vlaků, pomocné třídy pro konverzi dat, časů apod. Další skupinu tvoří řídící třídy pro práci s uživateli a správou autentifikace a autorizace a třídy pro RPC komunikaci s klientskou částí aplikace (implementace rozhraní `RemoteService`).

Na straně klienta se nachází ty řídící třídy, které manipulují pouze s objekty existujícími na straně klienta (DTO) a které primárně nevyžadují pro svou činnost dynamické dotazování do databáze. Fungují tak na straně klienta s daty, která jim byla poskytnuta při inicializaci klientského rozhraní nebo na základě podnětu uživatele. Mezi tyto patří třídy pro správu a sestavu uživatelského rozhraní po přihlášení konkrétního uživatele (práce s uživatelskými rolemi) a sestavu jednotlivých grafických komponent POK apod.

Samostatnou část řídících tříd tvoří servlet registrující požadavek na nahrání souboru na server. Má za úkol obsluhu požadavku, validaci souboru, jeho uložení na server a následně spustí import dat do databáze. Servlet o výsledku operace informuje klientskou část aplikace (view), která zobrazí uživateli výsledek, viz obrázek 31. Pro nahrání dat na server nebyl použit mechanismus RPC, který umožňuje serializovat data v textové podobě, ale klasický servlet pracující s objektem `Session`. Aby bylo možné data nahrávat na server bez obnovení stránky na straně klienta, tedy s pomocí technologie AJAX, byla využita knihovna `GWTUpload`, zajišťující snadnou implementaci nahrávání souboru na server v GWT aplikacích. (14)

3.9 Možnosti budoucího rozšíření aplikace

Implementovaná aplikace z hlediska zadání naplňuje všechny cíle. Byl implementován funkční návrh řešení stávajících problémů při používání plánů obsazení kolejí – byl z části automatizován proces tvorby a navrhnut nástroj pro vedení plánu. Pokud by však aplikace měla být nasazena jako plně funkční řešení, které by mělo nahradit stávající zavedenou „papírovou“ pomůcku, bylo by nutné aplikaci rozšířit o další funkcionalitu.

Zejména by bylo vhodné rozšířit vstupní množinu dat o informace, které by obsahovaly údaje o přečíslování vlaků – ve stávající množině dat jsou zahrnuty pouze holé údaje o daném spoji. V praxi ale vlak přijede do ŽST a následně dojde např. k manipulační jízdě a k vlaku jsou přidány další vozy, popř. je vyměněna lokomotiva a tím vznikne nový vlak (nové číslo vlaku), který pokračuje svou jízdu ze stejné nebo jiné koleje. Tyto informace jsou v praxi zanášeny do POK, avšak na základě několika různých vstupů dat (interní informační systémy, znalosti dispečerů apod.).

Dále by bylo vhodné implementovat možnost plně funkčního přístupu k modifikaci generovaného POK a také možnost zanášení individuálních změn, které není možné plně

podchytit v rámci vytvářené aplikace – jsou to např. individuální data pro konkrétní ŽST, která se do POK zanáší a podléhají zavedeným konvencím.

V implementované aplikaci byl vyvíjen POK dle vybraného vzoru skutečného plánu. V praxi však existují různé modifikace z hlediska použitých grafických primitiv a proto by bylo vhodné implementovat uživatelské rozhraní, kde by bylo možné definovat např. vlastní typy šipek, vlastní nastavení barev, možnost rozložení kolejí, definování vlastních pravidel pro zobrazování různých typů vlaků, nastavení pro příjezdové a odjezdové směry a ostatní nastavení, která podléhají zvyklostem zavedeným v konkrétní ŽST.

4 ZÁVĚR

Všechny cíle diplomové práce byly splněny. Nejprve byli čtenáři práce seznámeni s problematikou tvorby plánu obsazení kolejí. Pro ilustraci byly uvedeny odlišné přístupy k sestavě v různých ŽST. Byly zmíněny i softwarové nástroje, v kterých jsou plány sestavovány. Následně byly uvedeny výhody a nevýhody současného řešení.

V další kapitole byla navrhnuta koncepce možného řešení sestavy a vedení plánu pomocí softwarového nástroje, který je předmětem této diplomové práce. Následně byly představeny vhodné technologie, které poslouží při vývoji aplikace a zároveň vyhovují požadavkům kladeným na vyvíjenou aplikaci.

V následující kapitole byla provedena analýza požadavků na aplikaci a uveden postup řešení při implementaci, který se řídil metodikou Unified process. Zejména byly uvedeny funkční a nefunkční požadavky na aplikaci, byly vytvořeny případy užití, které tyto požadavky realizují a scénáře pro případy užití. Dále byl sestrojen analytický model tříd navrhované aplikace, který je umístěn v příloze diplomové práce. Při hledání analytických tříd byla použita metoda podstatných jmen a sloves. Následně byl sestavován datový model aplikace, který byl následně realizován v databázi Oracle. Byly popsány nejdůležitější tabulky datového modelu a odůvodněny použité datové typy, popř. koncepce ukládání historie uživatelských změn v databázi. Dále byly v kapitole popsány důležité implementační pasáže, tedy řešené problémy např. se sestavou kalendářů vlaků z importovaných dat, návrh uživatelského rozhraní a realizace vykreslení plánu pomocí vektorové grafiky v prostředí webového prohlížeče. Byla zmíněna i struktura aplikace v MVC modelu.

Vytvořená aplikace je plně funkční. Pro svou funkci vyžaduje aplikační server Java EE a databázi. Aplikace umožňuje sestavovat plán obsazení kolejí na základě zdrojových dat, není tedy nutné spoléhat se na ruční sestavu plánu v jakémkoli pomocném programu. Aplikace dále nabízí uživatelům možnost interaktivního zásahu do aktuálního plánu a tím může posloužit jako pomůcka při řešení standardních i nestandardních situací. Např. dovoluje dynamicky vlaky přesouvat v rámci kolejí, měnit délky pobytů na koleji a přidávat zpoždění. To vše při kontrole konfliktů s jinými vlaky. Dále je možné přistupovat k aplikaci s různými uživatelskými oprávněními a tím tak umožnit zpřístupnění aplikace co největšímu počtu zaměstnanců.

I přes vynaložené úsilí aplikace nedisponuje tak dokonalým výsledným grafickým plánem obsazení kolejí, jako je ten ručně sestavený. Jedním z důvodů byla i absence některých důležitých informací ve zdrojových datech a nemožnost jejich automatického vykreslení v plánu. Dalším důvodem byla obrovská variabilita zanesených individuálních informací a jejich závislost na konkrétní ŽST a zavedených konvencích. V neposlední řadě ručně sestavovaný plán je tvořen odborníkem v dané oblasti. I přes to aplikace přináší možný nadhled na problém při snaze zlepšit proces sestavy a vedení plánu obsazení kolejí. Do budoucna by bylo určitě vhodné získat detailnější zdrojová data a zpracovat je do sofistikovanějšího výsledného plánu.

5 POUŽITÁ LITERATURA

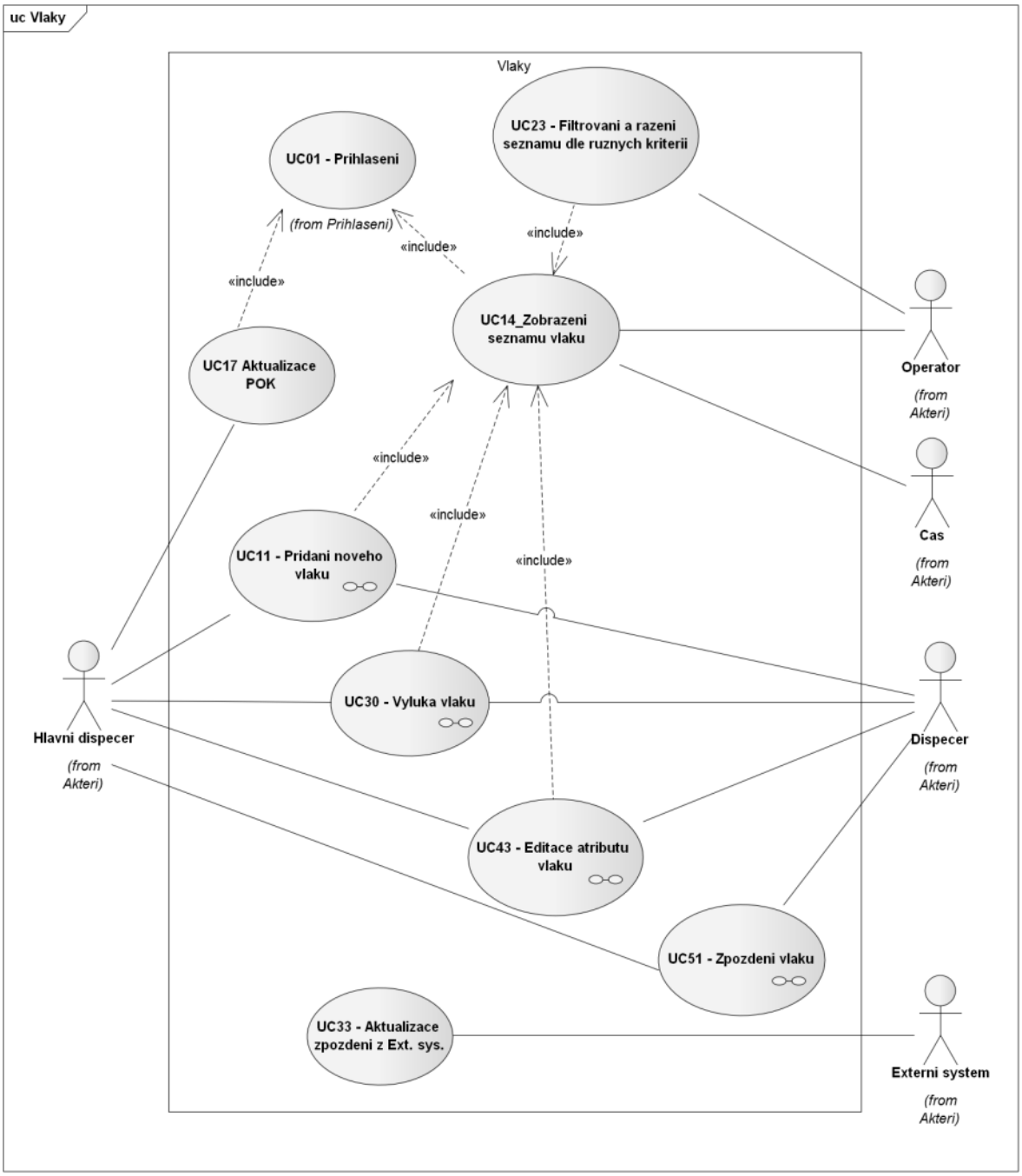
1. HEJZLAR, Jaroslav. *Sestava plánů obsazení kolejí v osobních železničních stanicích.* Pardubice: Univerzita Pardubice. Dopravní fakulta Jana Pernera. Katedra informatiky v dopravě, 2007. 69 s. Vedoucí diplomové práce Ing. Michael Bažant.
2. *Prováděcí směrnice ke směrnici ČD D5: Technologické postupy úkonů žst*. Praha: České dráhy, 1.12.2007.
3. ŠTEKR, Kamil, CIBULKA, František. *Obsazení dopravních kolejí v žst. Praha hl.n. pro GVD 2009/2010.* Praha: České dráhy, a. s., 2.12.2009.
4. HANSON, Robert, TACY, Adam. *GWT in action: easy Ajax with the Google Web toolkit*. Greenwich, CT: Manning, c2007, xxxii, 597 s. ISBN 19-339-8823-1.
5. RASOLO, Franck. Calling the server from the client. *We like GWT presentation* [online]. c2010, poslední revize 4.2.2010 [cit. 2013-05-06]. <https://code.google.com/p/welikegwt-presentation/wiki/GwtRpc>.
6. Google. Making Remote Procedure Calls. *Google Web Toolkit* [online]. Google Inc., 25.10.2012 [cit. 2013-03-20]. <https://developers.google.com/web-toolkit/doc/latest/tutorial/RPC>.
7. LINWOOD, Jeff, MINTER, Dave. *Beginning Hibernate*. 2nd ed. New York: Apress, c2010, 379 s. ISBN 978-143-0228-516.
8. JBoss Community. About Hibernate. *Hibernate* [online]. JBoss, c2009 [cit. 2013-04-06]. <http://www.hibernate.org/about>.
9. W3C. *Options for using SVG in Web pages* [online]. W3C, 16. 8. 2011 [cit. 2013-04-16]. <http://www.w3schools.com/svg/svg_inhtml.asp>.
10. ARLOW, Jim. *UML 2 and the unified process: practical object-oriented analysis and design*. Vyd. 1. Boston: Addison-Wesley, 2005, 592 s. ISBN 03-213-2127-8.
11. ŠTEKR, Kamil, CIBULKA, František. *Plán obsazení dopravních kolejí v žst. Praha hl. n. pro GVD 2012/2013.* Praha: České dráhy, 4. 4. 2013.
12. Sencha Inc. Application Framework for Google Web Toolkit. *Sencha GXT* [online]. Sencha Inc., c2013. [cit.2013-03-20]. <http://www.sencha.com/products/gxt>.

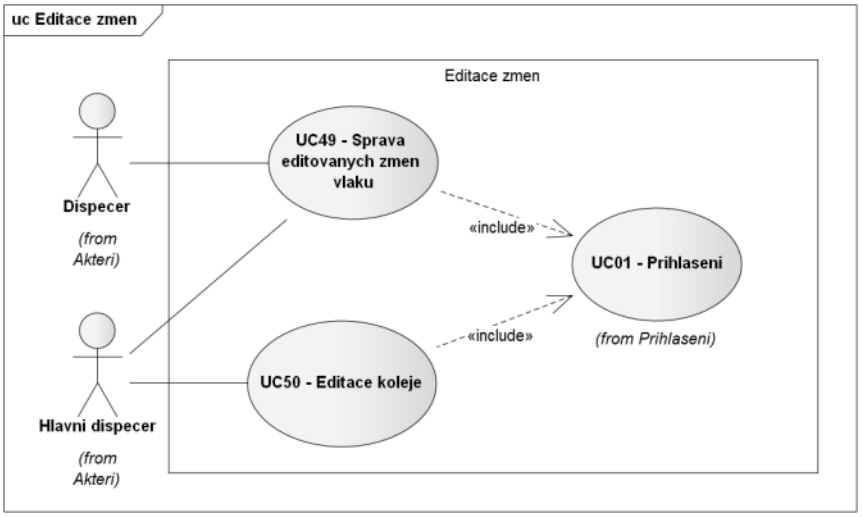
13. HRADIL, František. Automatické kopírování entitních objektů do DTO objektů. *BCVlog* [online]. 23.5.2010. [cit. 2013-04-21]. <http://blog.bcvsolutions.eu/automaticke-kopirovani-entitnich-objektu-do-dto-objektu>.

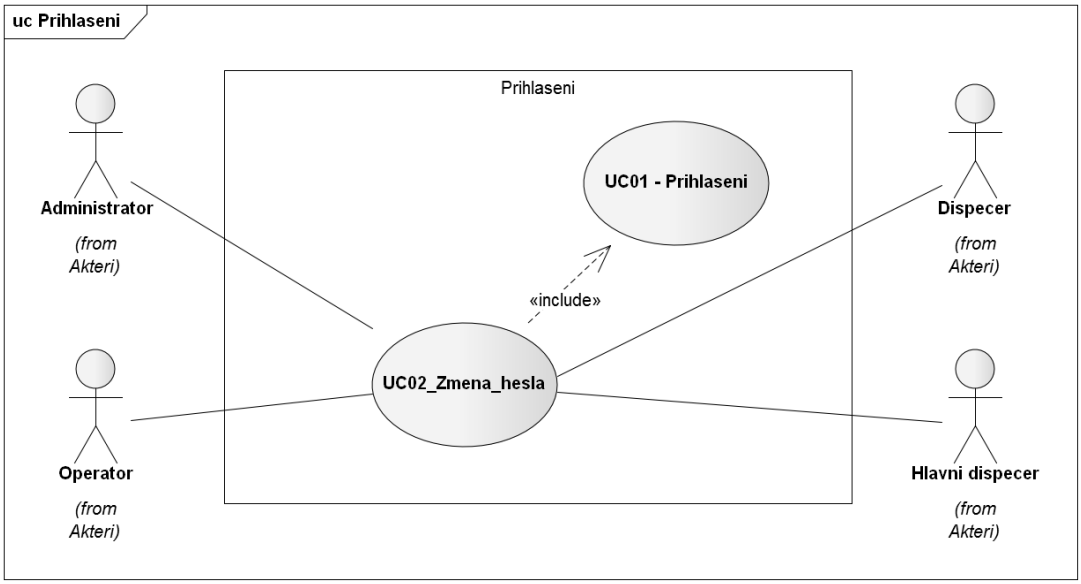
14. MOÑINO**,** Manolo Carrasco**.** GWTUpload. *Google code* [online] c2009-2012. [cit. 2013-04-22]. <https://code.google.com/p/gwtupload>.

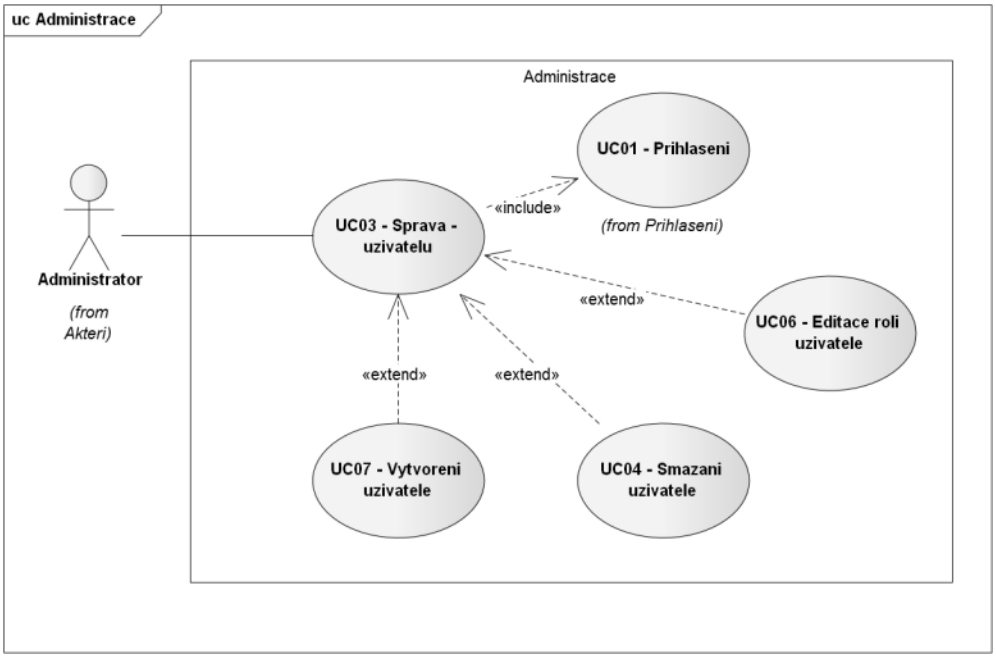
6 PŘÍLOHY

Příloha A – Případy užití

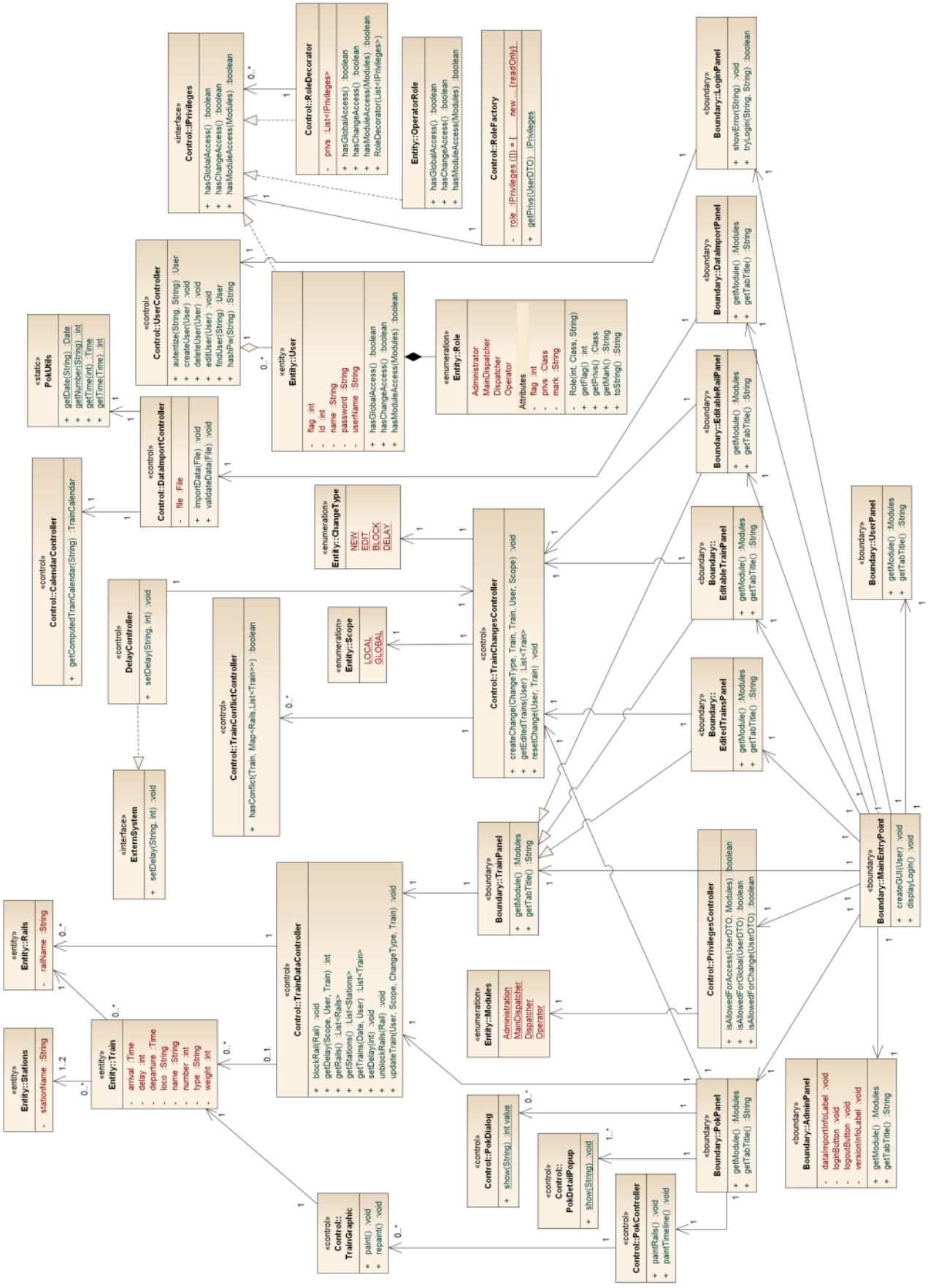



uc Editace zmen
Editace zmen
Dispecer
(from Akteri)
Hlavni dispecer
(from Akteri)
UC49 - Sprava editovanych zmen vlaku
UC50 - Editace koleje
«include»
«include»
UC01 - Prihlaseni
(from Prihlaseni)

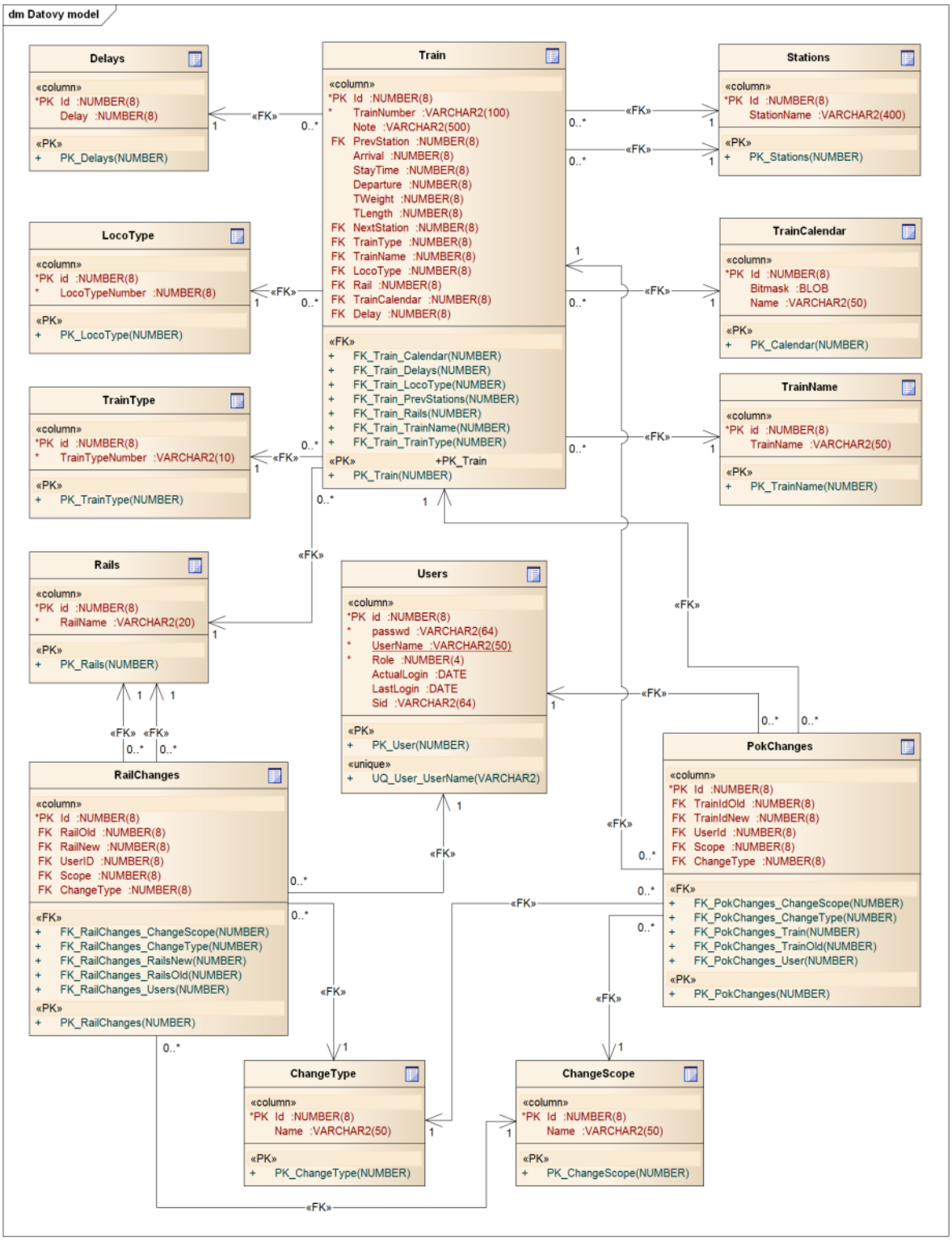

uc Prihlaseni
Prihlaseni
Administrator
(from Akteri)
Operator
(from Akteri)
UC01 - Prihlaseni
«include»
UC02_Zmena_hesla
Dispecer
(from Akteri)
Hlavni dispecer
(from Akteri)


uc Administrace
Administrace
Administrator
(from Akteri)
UC01 - Prihlaseni
(from Prihlaseni)
«include»
UC03 - Sprava - uzivatelu
«extend»
UC06 - Editace roli uzivatele
«extend»
«extend»
UC07 - Vytvoreni uzivatele
UC04 - Smazani uzivatele

Příloha B – UML analytických tříd



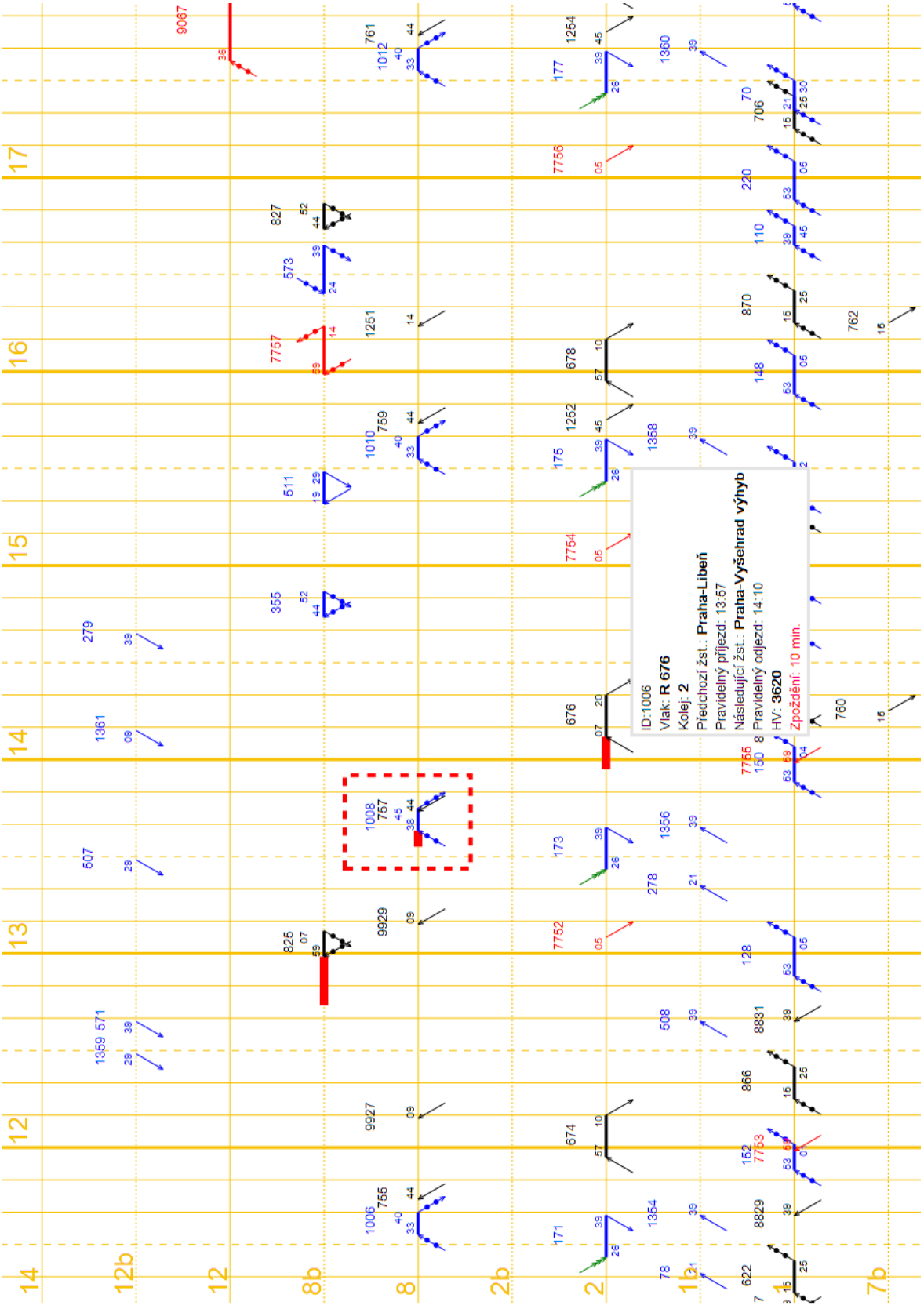
Příloha C – UML datového modelu



Příloha D – Fragment SVG elementu

```
<svg width="4600px" height="3100px">
  <g>
    ...
    <g style="stroke: #a52a2a; fill: #a52a2a;"
      transform="translate(578.75 1226.82)">
      <line style="stroke-width: 3px;" x2="9.375" y1="0"
        y2="0" x1="0"></line>
      <text style="stroke: rgba(0, 0, 0, 0); font-size: 10px;"
        x="0px" y="12px">46</text>
      <text style="stroke: rgba(0, 0, 0, 0); font-size: 10px;"
        x="-2.625px" y="24px">49</text>
      <g transform="translate(-10.3125 -30)">
        <text style="stroke: rgba(0, 0, 0, 0);" x="0px"
            y="0px">65520</text>
      </g>
      <g transform="translate(0 0) rotate(-120)">
        <g style="stroke: #008000; fill: #008000;">
          <line x2="30" y1="0" y2="0" x1="0"></line>
          <line x2="5" y1="0" y2="2" x1="0"></line>
          <line x2="10" y1="0" y2="2" x1="5"></line>
          <line x2="15" y1="0" y2="2" x1="10"></line>
          <line x2="5" y1="0" y2="-2" x1="0"></line>
          <line x2="10" y1="0" y2="-2" x1="5"></line>
          <line x2="15" y1="0" y2="-2" x1="10"></line>
        </g>
        <text style="stroke: rgba(0, 0, 0, 0);" transform=
            "rotate(180 20 -8.74228e-7)" x="9px" y="-5px"></text>
      </g>
      <g transform="translate(9.375 0) rotate(-60)">
        <g>
          <line x2="30" y1="0" y2="0" x1="0"></line>
          <line x2="25" y1="0" y2="2" x1="30"></line>
          <line x2="25" y1="0" y2="-2" x1="30"></line>
        </g>
      <text style="stroke: rgba(0, 0, 0, 0);"
        x="7px" y="-5px"></text>
      </g>
    </g>
    ...
  </g>
</svg>
```

Příloha E – Fragment vytvořeného plánu obsazení kolejí



Příloha F – Seznam příloh na přiloženém optickém médiu

Na přiloženém médiu jsou umístěny veškeré vypracované materiály.

Ve složce *Zdrojové kódy* jsou umístěny veškeré zdrojové kódy aplikace, včetně použitých verzí knihoven. Je přiloženo i použité vývojové prostředí NetBeans.

Složka *Model* obsahuje soubor projektu v programu Enterpreise Architect, v kterém byly vytvořeny veškeré diagramy.

Složka *Dokumentace* obsahuje text této práce.