Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Komponentová aplikace Řízení silniční křižovatky

Plzeň 2012 Filip Šimůnek

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů. Nemám námitek proti půjčení práce se souhlasem katedry ani proti zveřejnění práce nebo její části.

V Plzni dne 22. června 2012

Filip Šimůnek

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Tomáši Potužákovi Ph.D. za cenné a rady a připomínky při vedení bakalářské práce, které mi pomohly při jejím vypracování. Další poděkování patří pedagogům z Fakulty aplikovaných věd, kteří mně po dobu studia předávali své znalosti. Nakonec děkuji členům své rodiny za podporu během studia.

Abstract

The purpose of this thesis was to design a model of component-based application for controlling a traffic crossroad and test it for extra-functional requirements.

The main result of this thesis is an implementation of traffic crossroad model with two traffic control algorithms. It was developed by component model Spring DM, which is the extension of OSGi framework. This project was created as a bachelor´s thesis primary for Department of Computer Science and Engineering for simulation framework for testing components.

The component application contains 11 components. They evaluate dates, save statistics and control simulated traffic crossroad from traffic crossroad component by two possible algorithms. Each component has its own functionality and provides services, which are defined in interface.

Obsah

1 Úvod

Komponentově orientované programování posledních deset let nabírá na významu. Hlavní myšlenkou tohoto přístupu je dekomponování velkých projektů na menší části. Aplikace se rozdělí na elementární, ale samostatné a kompletní jednotky — komponenty. Takový přístup k tvorbě softwaru má v dnešní době, kdy se musí neustále v co nejkratším čase upravovat a doplňovat software k potřebám zákazníka, velké opodstatnění. Komponentu lze z celé aplikace snadno vyjmout a rychle opravit bez nutnosti zdlouhavého předělávání celé aplikace. Velkou výhodou komponenty je také možnost jejího znovupoužití do více jiných aplikací.

Cílem této bakalářské práce je návrh a implementace aplikace sestávající se z několika mezi sebou provázaných komponent. Vzniklá komponentová aplikace by mohla do budoucna sloužit k testování v simulačním frameworku, který je vyvíjen na KIV[1]. Proto byl tento projekt vyvíjen ve Spring DM frameworku[2], s jehož pomocí vzniká i výše zmíněný simulační framework. Simulační framework je zaměřen na testování mimofunkčních požadavků a kvalitu služeb, což jsou vedle samotné funkcionality důležité vlastnosti softwarových komponent.

Konkrétním tématem bakalářské práce je vytvoření komponentové aplikace silniční křižovatky, která obsahuje komponenty zodpovědné za správné řízení křižovatky. Tyto komponenty budou například zajišťovat detekování vozidel na křižovatce pomocí senzorů, sbírání souhrnných dat a ovládání světel křižovatky. Pro ověření správného fungování aplikace, a komunikace mezi komponentami je mým hlavním úkolem implementovat dva algoritmy pro řízení křižovatky. Prvním je statický algoritmus řízení a druhým dynamický algoritmus VASC[3], každý implementovaný jako samostatná komponenta. Dynamický algoritmus bude ke své činnosti využívat na rozdíl od statického téměř všechny komponenty. U obou komponent budeme sledovat kvalitu služeb, konkrétně jejich schopnosti efektivně řídit provoz v křižovatce. Tímto způsobem bychom měli od dynamického způsobu řízení křižovatky dostat lepší výsledek řízení křižovatky než pouze pomocí statického a zároveň tak získáme ověření správné funkčnosti a komunikace všech komponent v aplikaci.

[1] Katedra informatiky a výpočetní techniky

[2] Implementace komponentového modelu Spring Dynamic Modules, viz 2.3.3

[3] Vehicle actuated signal control viz 4.1.2

2 Komponentový přístup k tvorbě softwaru

Jak už bylo řečeno v úvodu, komponentově orientované programování, nebo také CBSE (Component-based Software Engineering), popř. CBD (Component-based Development) je v dnešní době na vzestupu. Problematika CBSE pomohla ke vzniku komponentových modelů, což jsou specifikace komponentové tvorby softwaru. Dělením aplikace na menší autonomní části vznikají komponenty. Ty jsou volány přes rozhraní a jejich implementace zůstává navenek skrytá. Takový přístup má pro dnešní dobu spoustu výhod, mezi něž patří[1]:

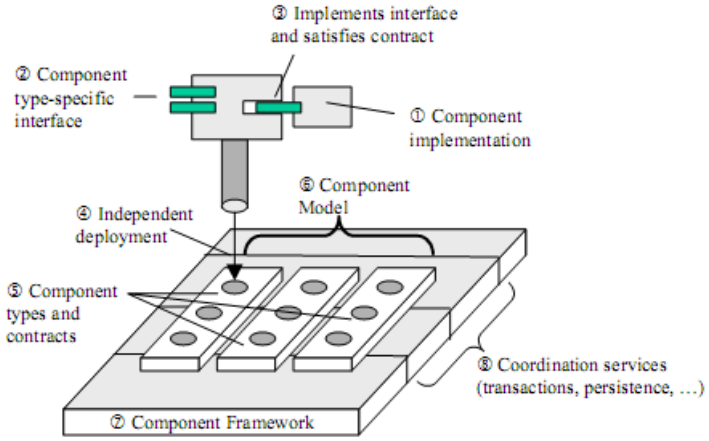
- rozdělení aplikace jednotlivé části, které jsou tím pádem snadněji opravitelné, než když se musí zasahovat do celé aplikace
- zrychlení vývoje aplikace a tudíž zlevnění ceny softwaru
- předpokladem je znovupoužití, popř. použití již otestovaných komponent třetích stran
- předvídatelnost chování komponenty, ale na druhou stranu skrytí obsahu komponent za rozhraní

Je ovšem třeba zmínit i nevýhody:

- velká režie při přechodu od jiného přístupu k programování k vývoji komponent
- existence příliš velkého množství frameworků
- vývoj komponent třetích stran je třeba sledovat, protože se rychle vyvíjejí

2.1 Základní pojmy

Na Obr.2.1 můžeme vidět, jak vypadá schéma komponentově orientovaného softwaru.



Obrázek 2.1: schéma součástí komponentového softwaru převzato z [2]

Nyní si popíšeme základní pojmy vyskytující se v souvislosti komponentově orientovaným programováním.

2.1.1 Komponenta

Komponenty jsou části softwaru s definovaným rozhraním a funkcionalitou. Každá softwarová komponenta je navrhována pro maximalizaci znovupoužitelnosti a využívání komponenty třetí stranou. Chová se podle principu černé skříňky, což znamená, že vnitřní implementace zůstává skrytá navenek[3]. Kromě samotné funkcionality mají softwarové komponenty i další vlastnosti, ovlivňující jejich použitelnost v komponentových aplikacích. Tyto vlastnosti jsou souhrnně nazývány mimofunkční požadavky a kvalita služeb. Často jsou předmětem testování[4].

2.1.2 Komponentové rozhraní

Rozhraní je tou částí komponenty, která je viditelná ostatním a popisuje, jaké služby komponenta poskytuje. Komponenty jsou na sobě implementačně nezávislé, takže rozhraní slouží k jejich komunikaci. Pro správnou komunikaci si na sebe komponenty definují závislosti(dependency[1]). Tento proces se

[1]odkaz na jiný objekt resp. komponentu

liší od běžného postupu tím, že komponenta sama hledá objekt na kterém závisí[2].

2.1.3 Kontrakty a interakce komponent

Kontrakty určují „práva a povinnosti“ zúčastněných komponent a měly by zajišťovat dobré spojení rozhraní. Teoreticky mají možnost se měnit za běhu nebo se mohou zúčastněné strany i domluvit na vypršení kontraktu[2].

Existují různé druhy vztahů komponenty viz rozdělení podle [2]. Kromě normálního spojení existuje ještě vnoření komponent, nasazení komponenty do frameworku, nebo rozšíření vnořeného frameworku komponentou.

2.1.4 Komponentový model

Komponentový model specifikuje, jak má komponenta vypadat, jak se má chovat a jak má komunikovat s ostatními komponentami Také specifikuje, zda existuje více druhů komponent a jak probíhá jejich skládání [3].

2.1.5 Komponentový framework

Komponentový framework je implementací specifikace komponentového modelu. Existuje velké množství komponentových modelů, které se dají rozdělit do různých kategorií. Frameworků pro ně existuje dokonce ještě více a jsou vyvíjeny a využívány ve výzkumné sféře i průmyslu [3].

2.2 Proces vývoje komponent

Proces vývoje komponent můžeme rozdělit na několik fází, z nichž se jednotlivé mohou v závislosti na používaném komponentovém modelu lišit[5].

2.2.1 Návrhová fáze

Zde je třeba rozdělit aplikaci na několik logických celků (komponent), z nichž každý má svojí danou funkcionalitu. Pro každou komponentu jsou jasně definovány služby, které bude poskytovat a které bude požadovat. Tento postup se opakuje, dokud nejsou jednotlivé komponenty primitivní[2]. Takové komponenty mohou být použity znovu a lze z nich sestavovat větší komplexnější komponenty, které následní utvoří celou aplikaci.[5].

2.2.2 Vývojová fáze

V druhé fázi procesu probíhá vývoj kódu pro primitivní komponenty. Komponentové modely umožňují na základě specifikace komponenty část zdrojového kódu vygenerovat. Výsledný zkompilovaný kód tvoří komponentu, která se používá pro další vývoj[5].

2.2.3 Fáze sestavení

V této fázi se všechny komponenty sestaví do assembly tree. Kořen stromu reprezentuje celou aplikaci a potomci každého uzlu reprezentují komponenty, ze kterých je jejich rodičovský uzel složen. Listy stromu tvoří komponenty používané pro sestavení aplikace. Zde se konfigurují hodnoty properties[3][5].

2.2.4 Fáze nasazení

Nasazení je poslední fází vedoucí k fungující aplikaci. Konfigurační hodnoty v assembly deskriptoru jsou zde nastaveny podle potřeb. Komponenty jsou rozděleny do DU(Deployment Units) podle toho, na jakém serveru poběží[5].

[2]dostatečně jednoduché k implementaci — neskládají se z dalších komponent

[3]defaultní konfigurační data, která mají podobu XML souboru nazývaného assembly descriptor

2.2.5 Certifikace

O certifikaci se stará někdo důvěryhodný, kdo vydá prohlášení o certifikovaném subjektu. Tím může být např. komponenta, framework nebo systém z nich složený. Je to důležitá fáze pro předvídání výsledných vlastností kompozice celku[5].

2.3 Příklady komponentových modelů

Pro implementaci tohoto projektu byl zadán komponentový model Spring DM. Používání tohoto modelu zahrnuje i potřebu znalostí o komponentových modelech Spring a OSGi, ze kterých Spring DM vychází. Proto zde budou popsány příklady těchto tří komponentových modelů resp. frameworků.

2.3.1 OSGi

OSGi[4] framework implementuje dynamický komponentový model a nabízí platformu služeb pro programovací jazyk Java. Poskytuje prostředí pro komponentově orientované aplikace a samostatné komponenty, které zde mohou být na dálku instalovány, spouštěny, vypínány, aktualizovány a odinstalovávány bez nutnosti restartu. OSGi framework je běžně používán v různých odvětvích průmyslu jako např. v autech, mobilech, PDA, atd.[6]

Komponenta se v OSGi nazývá OSGi *bundle*. OSGi bundle může obsahovat libovolný počet tříd zabalených v klasickém `.jar` souboru. Jediným rozdílem je, že soubor *manifestu* OSGi popisuje relace mezi bundly, jejich názvy, atd.[6].

OSGi bundly spolu komunikují skrze služby. Služby jsou implementací konkrétních rozhraní, které jsou registrovány v OSGi frameworku pomocí OSGi bundlů obsahujících tyto implementace. Služby jsou jednou z nejdůležitějších částí OSGi frameworku a jako komponentově orientované aplikace jsou vytvářeny z bundlů, které jsou spojené s těmito službami. Služby jsou registrovány přímo v kódu bundlu. OSGi bundle potřebuje vědět kontext bundlu, který je držen na vstupním bodu bundlu s názvem *bundle activator*.

[4]Open Services Gateway initiative — modulární systém pro Javu

Služby mohou být jednoduše registrovány, jak je vidět na Obr.2.2)[6].

Každý OSGi bundle může poskytovat svoje třídy a interfacy tím, že je zveřejní. Po jejich zveřejnění je mohou využívat ostatní bundly ve frameworku. Tato funkcionalita je další formou komponentové interakce a může být nejlépe vysvětlena porovnáním vůči sdíleným knihovnám v operačních sytémech[6].

```
IMgGen msgGen1 = new MsgGenImpl1();
context.registerService("cz.zcu.kiv.cosi.msgtalk.IMsgGen",
  msgGen1, new Hashtable("type", "MsgGenImpl1"));
```

Obrázek 2.2: Příklad registrace služby v OSGi, převzato z [6]

2.3.2 Spring

Spring framework poskytuje různé vlastností pro podporu vývoje podnikových aplikací. Mezi nejcharakterističtější vlastnosti tohoto frameworku patří vzdálený přístup, řízení transakcí, datový přístup aspektově orientovaného programování a IoC (Inversion of Control), což je proces, při kterém si komponenty definují své závislosti.[6]

Spring také poskytuje snadnou konfiguraci elementů pomocí konfiguračního XML souboru. Tento soubor musí obsahovat XML namespace pro Spring *beany*. Spring beana je základní element v konfiguračním XML souboru. Lze jí popsat také jako komponentu nebo jako implementaci konkrétní třídy. V konfiguračním souboru je možné definovat závislosti mezi komponentami, jak je vidět v jednoduchém příkladu viz Obr.2.3[6].

```
<beans>
  <bean id="foo" class="x.y.Foo">
    <constructor-arg ref="bar" />
    <property name="baz" ref="baz" />
  </bean>

  <bean id="bar" class="x.y.Bar" />
  <bean id="baz" class="x.y.Baz" />

</beans>
```

Obrázek 2.3: Příklad definice beany ve Springu, převzato z [6]

2.3.3 Spring DM

Spring DM pro platformu OSGi služeb umožňuje vyvíjet OSGi komponenty pomocí Spring frameworku. Navíc vylepšuje ovladatelnost OSGi služeb převzato z [6].

Spring DM je distribuován jako několik OSGi bundlů, které se instalují a spouští v cílovém OSGi frameworku. Jeden z bundlů pak zkontroluje každý nový bundle instalovaný do frameworku a prověří, jestli je to Spring DM bundle nebo ne. OSGi bundle je transformován na Spring DM bundle jednoduchým přidáním konfiguračních XML souborů Springu do meta-information adresáře bundlu. Nové nastavení tohoto XML souboru musí obsahovat nový namespace Spring-OSGi, aby byla zachována podpora vlastností OSGi. Příklad registrace do Spring DM je vidět na Obr.2.4[6].

```
<bean id="InstanceSC" name="SC"
         class="cz.zcu.kiv.crossroadcontrol.statisticscollector.StatisticsCollector">
 </bean>

 <osgi:service id="VolaniRozhraniSC" ref="InstanceSC"
        interface="cz.zcu.kiv.crossroadcontrol.statisticscollector.IStatisticsCollector" />
```

Obrázek 2.4: Příklad registrace služby ve Springu DM

Použitím Spring DM tedy dostáváme základ modelu OSGi a možnost zjednodušené práce se službami a vlastnosti Springu při vývoji komponentové aplikace.

3 Řízení křižovatek v silniční dopravě

V této kapitole se podíváme na možnosti řízení křižovatek v silniční dopravě. Získáme teoretické povědomí o tom, jaké existují aspekty pro řízení křižovatek. A tyto informace budeme moci využít při návrhu a porovnávání s praktickou částí projektu.

3.1 Historie

Řízení křižovatek v silniční dopravě má svou bohatou historii, na kterou je účelné při hledání nových optimalizovaných řešení navázat. První řešení vznikala začátkem minulého století a jsou spojena s rozvojem automobilového průmyslu. Zpočátku řídili křižovatky strážníci, později se začaly na vybraných křižovatkách používat mechanické a poté i semafory světelné. Jejich použití bylo převzato z železniční a tramvajové dopravy. V češtině získalo světelné řízení dopravy označení „Světelná signalizační zařízení“ se zavedenou zkratkou SSZ.

Významná historická data zavádění SSZ ve světě:

- r.1868 Londýn, první instalace světelného semaforu v dopravě[7]
- r.1918 Salt Lake City, realizovaná první „zelená vlna“[1][7]
- r.1960 Toronto, bylo představeno první integrované řízení dopravy ve městě[7]

3.2 Bezpečnost

Z důvodů bezpečnosti silniční dopravy musí projektování a provozování SSZ splňovat Zákon o provozu na pozemních komunikacích (č.361/2000 Sb.) a

[1] pojem pro synchronizované fáze semaforů tak, aby vozidlo jedoucí doporučenou rychlostí zastihlo na všech semaforech signál volno

také řadu technických norem. V ČR se jedná o souhrn ČSN norem vzniklých přeložením evropských norem EN[8].

3.3 Druhy křižovatek

Křižovatka je místo, kde se pozemní komunikace v půdorysném průmětu protínají nebo stýkají a alespoň 2 z nich jsou vzájemně propojeny. Druhy křižovatek[9]:

- úrovňové
- mimoúrovňové(MÚK)

Z hlediska SSZ nás zajímají úrovňové křižovatky, které rozdělujeme na:

a) styková (kolmá / šikmá)

b) průsečná

c) vidlicovitá

d) odsazená

e) hvězdicovitá (alespoň 5 paprsků)

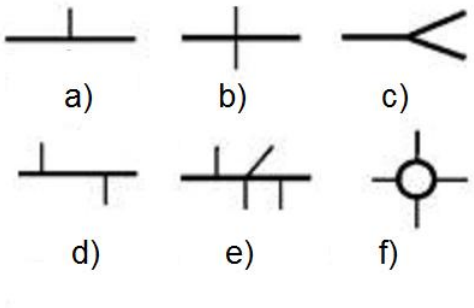
f) okružní

Schémata těchto křižovatek jsou vidět na Obr.3.1.

3.4 Technická řešení SSZ

3.4.1 Světelné semafory

Základem technického řešení SSZ příslušné křižovatky je řadič většinou umístěný ve skříňce v blízkosti křižovatky. Většinou existuje možnost ovládat signalizaci křižovatky manuálně policistou. Modernější SSZ bývá napojeno



Obrázek 3.1: Ilustrační schéma druhů křižovatek, převzato z [9]

dálkově na centrální dispečink, z něhož může dopravu řídit příslušný pracovník na dálku – zpětnou vazbu mu poskytuje kamerový systém. Střídání světel na semaforech se sestavuje v souladu s technickými předpisy, které respektují rychlost reakce řidičů, vozidel a chodců. Rychlost změny signálu musí počítat zejména s časem pro opuštění křižovatky. Současně příliš dlouhý čas čekání na signál může svádět řidiče k předčasné jízdě na červenou[10].

3.4.2 Detekce vozidel

Starší SSZ používají časově pevně nastavené cykly, které se v předem stanoveném pořadí střídají. Tyto cykly se maximálně jen několikrát za den mění dle předpokládaného provozu v příslušné denní době. I tato starší technologie umožňuje vytváření zelené vlny, což je sladění signálů na semaforech na navazujících křižovatkách. Novější SSZ používají dynamické řízení provozu, které nastavuje délku a pořadí světelných cyklů na základě pasivní i aktivní detekce účastníků provozu. Pro pasivní detekci vozidel se používají tlaková, optická nebo kamerová bezkontaktní čidla, počítače náprav, tramvaje a trolejbusy jsou většinou registrovány pomocí trolejových kontaktů reagujících na elektrický sběrač, autobusy MHD radiovým signálem. Pro chodce a cyklisty se užívá aktivní detekce pomocí tlačítka na semaforových sloupcích. Moderní semafory jsou současně vybaveny zvukovou signalizací pro nevidomé[10].

3.5 Základní pojmy řízení křižovatek

3.5.1 Pojem „fáze“

V současné době jsou křižovatky řízené výhradně SSZ, tj. souborem semaforů, které mají v reálném čase vůči sobě navzájem pevně nastaveny světelná označení (červená = „stůj“ , zelená = „volno“) v jednotlivých „dopravních proudech“ tak, aby nemohlo dojít ke kolizi vozidel, případně vozidel s chodci. Toto vzájemné nastavení světel se nazývá světelná „fáze“. Střídání těchto „fází“ postupně umožní automobilům průjezd křižovatkou všemi směry bez ohrožení vzájemnou kolizí (u složitějších křižovatek musí řidič současně se signálem „volno“ (zelená) respektovat při odbočování základní pravidla silničního provozu (přednost chodců na přechodu nebo přednost automobilů jedoucích v protisměru na zelenou rovně). Každá křižovatka má svůj „signální program“ střídání „fází“. Minimální počet fází jedné křižovatky jsou dvě a dále platí, že současně v činnosti může být pouze jedna fáze[11].

3.5.2 Druhy „signálních programů“

V provoze rozlišujeme „statický“ „signální program“, který nereaguje na dopravní proud a „dynamický“ „signální program“, který reaguje na okamžité potřeby provozu. Podle požadavků na řízení křižovatky existuje normální automatický „fázový“ režim, ruční přepínání „fázového“ režimu policistou, blikání oranžové na všech semaforech a vypnutá SSZ. „Signální program“ „statického“ řízení provozu pracuje se změnou jednotlivých „fází“. Ty musí probíhat s časovou prodlevou, která respektuje vyklizení křižovatky automobily před další fází. Pro tuto časovou prodlevu se využívá ve světelném označení oranžová = „připrav se“ resp. „dokonči průjezd“. Doba potřebná na časovou prodlevu se stanovuje pro každou křižovatku a pro každou „fázi“ samostatně a z bezpečnosti provozu vyplývá, že se signály „připrav se“ a „dokonči průjezd“ úplně nepřekrývají. Kromě stavu změny jednotlivých „fází“ je nezbytné i signalizačně ošetřit spuštění funkce SSZ a ukončení funkce SSZ, tj. blikající semafory, případně vypnutí semaforů. Tyto přechody je nezbytné řešit přes nastavení signálu červené na všech světlech a po dostatečné časové prodlevě nastavit blikání oranžové a poté zcela vypnout SSZ[11].

3.5.3 „Dynamické“ řízení křižovatky

„Signální program“ „dynamického“ řízení provozu navíc pracuje s detekcí přítomnosti vozidel v jednotlivých „dopravních proudech“. Údaje z detekcí jsou zpracovávány a samočinně upravují sled a délku jednotlivých „fází“. Tento proces se označuje jako volná tvorba „signálního programu“. „Dynamické řízení“ SSZ funguje na principu spouštění „fází“ na žádost. Tato žádost je vyvolaná aktivací příslušného detektoru. Ovlivňuje-li detektor více „fází“, je logickou funkcí vybraná ta, která vyhovuje požadavkům nejvíce směrů. V „dynamickém řízení“ má „fáze“ proměnnou délku trvání, která je omezená zdola i shora. Minimální délka trvá 5 sec., maximální délka pro tři fáze bývá 20 sec. V případě detekce vozidla MHD nebývá dodržování maximálního limitu tak striktní a je snaha, aby vozidlo zelenou stihlo. Vedle „statického“ a „dynamického“ řízení SSZ existuje ještě modernější řešení tzv. „adaptivní“ řízení. V něm nemá řídící systém křižovatky žádný konkrétní signální program a každá křižovatka je řízena komplexně v rámci rozlehlé dopravní oblasti[11].

3.6 Strategie řízení křižovatek

Pro optimální řízení konkrétní křižovatky je důležité zvolit správnou strategii[7].

3.6.1 Strategie ze získaných dopravních dat

Podle vlivu dopravních dat získaných v reálném čase pro řízení dopravy na tuto charakteristiku existují dvě strategie[7]:

- strategie s hotovým programem. „Signální program“ je vypočítán předem s použitím statistických dat.
- Strategie s programem v reálném čase. Data o dopravní situaci získaná v reálném čase jsou použita k řízení křižovatky případně k modifikaci hotového programu

3.6.2 Strategie výkonnostních ukazatelů

Nejčastěji používané ukazatele jsou[7]:

- *pro řízení dopravy na izolované křižovatce* — všechny hodnoty „zpoždění“ (součet všech „zpoždění“ ve všech směrech křižovatky během určitého časového intervalu, obvykle se jedná o čas jednoho „cyklu“); počet signálů „stůj“; vážený součet hodnot „zpoždění“ a počtu signálů „stůj“ ; součet všech „zelených“ časů v průběhu „cyklu“; všechny „intenzity“ dopravního „proudu“ přeplněné křižovatky během „cyklu“; počet nehod; doba trvání „cyklu“ atd.
- *pro řízení dopravní tepny* — rozpětí intervalu dopravního pásu, tj. interval, ve kterém je možné projet křižovatkou na „zelenou vlnu“ bez zastavení
- *pro řízení dopravy sítě vzájemně navazujících křižovatek* — součet všech hodnot „zpoždění“ v síti křižovatek; součet signálů „stůj“ v síti křižovatek; vážený součet hodnot „zpoždění“ a počet signálů „stůj“všech spojení v síti křižovatek; součet spotřeby paliva všech vozidel v síti křižovatek; úroveň znečištění vzduchu „zplodinami“; úroveň hladiny hluku; počet nehod atd.

3.6.3 Strategie dle dopravní hustoty

Dle této charakteristiky hodnotíme strategie řízení dopravy na[7]:

- Strategie pro slabou dopravu
- Strategie pro normální dopravu
- Strategie pro přetíženou dopravu v síti křižovatek
- Strategie pro speciální účely (např. dání přednosti MHD, dání přednosti hasičům atd.)

3.6.4 Strategie dle systému řízení dopravy

Pro tuto charakteristiku existují dva typy řízení dopravy[7]:

- Strategie, která používá centrální systém řízení dopravy, kde jsou všechny řídící funkce obsluhovány z jednoho centrálního počítače
- Strategie postavená na dekompozici problematiky řízení – část řídícího problému je řešena mikroprocesorovými ovladači jednotlivých křižovatek, zbytek problematiky je řízen centrálním počítačem.

3.6.5 Strategie dle rozvržení dopravy

Na základě této charakteristiky můžeme uvést dvě strategie[7]:

- Strategie, která vychází z toho, že rozvržení dopravy v síti křižovat je nezávislé na řízení dopravních světel
- Strategie, která vychází z toho, že oboje(nastavení signálních parametrů a dopravní tok v daném směru) nejsou fixní, tj. strategie, která optimalizuje vybraný výkon indexu a má vliv na rozvržení dopravy

4 Komponentová aplikace křižovatky

4.1 Analýza

Ze zadání projektu je patrné, že před zahájením samotného vytváření komponentové aplikace řízení silniční křižovatky pro komponentový model Spring DM bylo nutné se seznámit s prostředím, ve kterém bude tento projekt vyvíjen a seznámit se také s konfigurací a prací s komponentami. Poté bude potřeba navrhnout, jak bude aplikace fungovat a jak se bude postupovat při jejím vytváření.

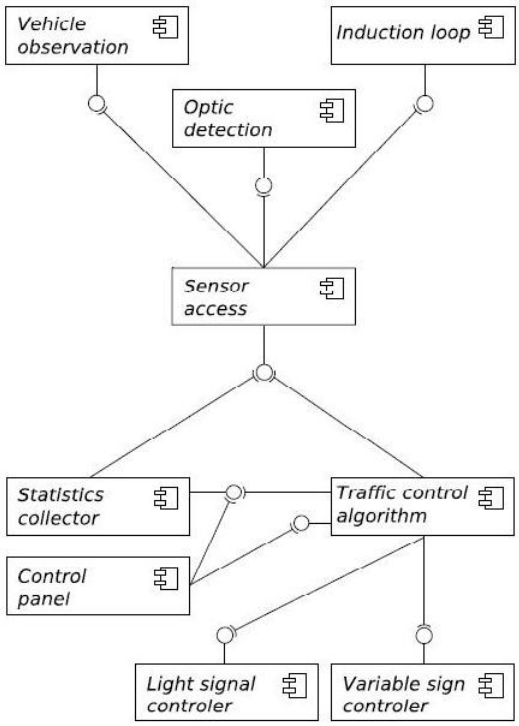
Pro ponoření se do problému komponentového programování bylo vhodné vyzkoušet si práci s komponentami na dvou vzorových příkladech podle návodu, který mi byl dodán. Po zkušenosti s fungováním komponent pak bylo potřeba vybrat si mezi dvěma způsoby komunikace mezi komponentami. Byl zvolen způsob komunikace s využitím rozhraní, neboť pro potřeby aplikace plně dostačoval a byl implementačně jednodušší než druhý způsob, který využíval správce události pro předávání zpráv.

4.1.1 Návrh křižovatky

Jak již bylo uvedeno v kapitole 3.4, existuje mnoho elementů, které mohou mít vliv na řízení silniční křižovatky a cílem bylo přiblížit se co možná nejvíce skutečné situaci na reálné křižovatce. První návrh křižovatky tedy vypadá následovně viz Obr.4.1.

Na obrázku není zobrazena komponenta Traffic crossroad jejíž funkcí je simulování provozu reálné křižovatky viz na předchozím schématu. Tato komponenta byla dodána zadavatelem a celá komponentová aplikace ze schématu bude využívat služby této komponenty poskytované přes její rozhraní. Je tedy potřeba navrhnout, jak celá aplikace bude fungovat, resp. jakým způsobem bude probíhat komunikace mezi komponentami ze schématu a dodanou komponentou Traffic crossroad. Vzhledem k tomu, že Traffic crossroad obsahuje informace o všech elementech[1] křižovatky nutných pro její řízení, bude se každá z komponent viz Obr.4.1 starat o tyto jednotlivé elementy. Aplikace

[1] např. pruhy, semafory, auta, atd..



Obrázek 4.1: Schéma komponentové aplikace křižovatky bez komponenty Traffic crossroad simulující křižovatku, převzato z [12]

se takto dle principů komponentového programování rozdělí na menší části, které je snadnější později nahradit. Této vlastnosti využijeme například u komponenty Traffic control algorithm[2] viz 4.1.1, zajišťující algoritmus řízení křižovatky. Lehce ji můžeme vyměnit za jinou komponentu s jiným algoritmem aniž by bylo nutné zasahovat do dalších částí aplikace.

Nyní popíšeme návrhy funkčností konkrétních komponent tak, aby odpovídaly reálné křižovatce a zároveň byly schopny spolupracovat na řízení simulované křižovatky komponenty Traffic crossroad.

[2]tento název budeme zatím používat obecně pro komponentu implementující blíže nespecifikovaný algoritmus řízení křižovatky

Traffic crossroad

Externě dodaná křižovatka, která simuluje běh reálné křižovatky. S touto komponentou budou komunikovat téměř všechny komponenty aplikace.

Vehicle observation

Tato komponenta funguje jako senzor vozidel zajištěný přímým pozorovatelem. Reálně by její funkci zastával například člověk, který by zaznamenával počet vozidel projíždějících pruhem. Tímto senzorem bychom pak měli dostat úplnou informaci o počtu aut v pruhu.

Optic detection

Komponenta zastávající funkci senzoru na principu optické detekce. V praxi by se takto detekoval počet aut pomocí rozpoznávání obrazu z dat zachycených kamerou, která je umístěna na křižovatce. Tento senzor podle možností rozpoznávání obrazu podá informace a počtu aut v pruhu až do vzdálenosti, kam je schopen auta rozpoznat.

Induction loop

Komponenta Induction loop plní funkci indukční smyčky. Tento typ senzoru je v praxi nainstalován pod vozovkou a je schopen rozpoznat, zda nad ním stojí, nebo nestojí nějaké vozidlo. Indukční smyčka se může nacházet v různé vzdálenosti od hranice křižovatky a v závislosti na tom je možné usoudit, že až do tohoto místa stojí fronta vozidel. Nicméně toto není relevantní údaj, protože mezi autem nad smyčkou a křižovatkou může být prázdná mezera. Smyčka tedy podává informaci pouze o tom, stojí-li nad ní auto.

Sensor access

Sensor access je komponenta, která má přístup ke všem senzorům na křižovatce. Těmito senzory jsou Vehicle observation, Induction loop a Optic

detection. Komponenta shromažďuje informace ze senzorů a plní roli prostředníka mezi senzory a komponentami požadujícími data o počtu aut ze senzorů. V reálu si jej lze představit jako zařízení ukládající data ze senzorů na datové úložiště.

Statistics collector

Komponenta shromažďující a vyhodnocující data z křižovatky. Nahrává shromážděná data ze Sensor access. Může vyhodnocovat dlouhodobé statistiky z běhu křižovatky a podle toho posílat komponentě traffic control algorithm návrhy na změny v algoritmu řízení křižovatky.

Traffic control algorithm

Hlavní řídící komponenta křižovatky. Obsahuje algoritmus pro měnění fází křižovatky a dává pokyny komponentě Light Signal controller pro změnu světel na semaforech. Od Sensor access dostává informace o přítomnosti vozidel v pruzích a od Variable sign controller může dostat zprávu o změně povolené rychlosti v pruhu. Těmto datům pak může být dočasně přizpůsoben algoritmus řízení.

Control Panel

Komponenta, která se dá charakterizovat jako ovládací panel křižovatky. Odtud je možné nastavovat senzory pro sběr dat, vypínat SSZ a zobrazovat statistiky shromážděné v Statistics collectoru.

Light signal controller

Light signal controller je komponenta, která plní funkci prostředníka mezi Traffic control algorithm a Traffic crossroad. Její funkcí je měnit světla na křižovatce.

Variable sign controller

Tato komponenta není běžnou součástí každé křižovatky. Lze si ji představit jako informativní dopravní značku jako displej. Ta by zobrazovala např. informaci o snížení maximální povolené rychlosti. Toto řízení mohou využívat některé algoritmy řízení křižovatky.

4.1.2 Postup vytváření aplikace

Sestavování komponent pro statický algoritmus řízení

Nejrychlejší cestou k dosažení první funkční aplikace řízení nám stačí vytvořit komponentu Traffic control algorithm pro statický algoritmus řízení. Pro tento případ není potřeba zatím vytvářet další komponenty, protože algoritmus je zde řešen staticky a nepotřebuje pro svou činnost externí data z jiných komponent. Komponenta je zatím propojena pouze s Traffic crossroad a přes interface této komponenty bude jen nastavovat změny světel pro jednotlivé fáze.

V dalším kroku je možné aplikaci rozšířit o další komponentu — Light signal controller. Ta se stane prostředníkem mezi komponentou Traffic crossroad a komponentou Traffic control algorithm a převezme od ní přepínání světel na konkrétních pruzích křižovatky. Připojením komponenty Control panel pro ovládání spouštění a vypínání algoritmu získáme první funkční komponentovou aplikaci pro řízení křižovatky. Připojení dalších komponent pro statické řízení křižovatky nemá zatím moc význam.

Sestavování komponent pro dynamický algoritmus řízení VASC

Již vytvořená komponentová aplikace nám usnadňuje práci při vývoji celé komponentové aplikace podle schéma z Obr.4.1. Pro implementaci dynamického algoritmu řízení bude potřeba zapojit celý zbytek komponent z obrázku kromě komponenty Light signal controller, která nemá v dodané komponentě Traffic crossroad dle svého popisu uplatnění.

Vytvoří se tři komponenty senzorů s podobnou funkčností — zjišťováním počtu, resp. přítomnosti vozidel v jednotlivých pruzích křižovatky. Po získání

těchto informací z Traffic crossroad jsou tyto informace odeslány do Sensor access. Komponenta Sensor access se spojí s novou komponentou Traffic control algorithm VASC, která nahradila místo původní komponenty Traffic control algorithm. Této nové komponentě pro dynamické řízení jsou odeslány informace o přítomnosti vozů v pruzích a ta je vyhodnocuje v algoritmu a podle toho dává pokyn ke změně fáze světel Light signal controlleru.

Sensor access posílá data ještě poslední komponentě ze schématu Statistics collectoru. Ten sbírá statistiky počtu vozidel v pruzích. Tyto statistiky a možnost zvolit si typ senzoru pro sběr statistik jsou zobrazeny v pomocí komponenty Control panel.

Do této komponentové aplikace je možné kdykoliv vložit zpět původní komponentu Traffic crossroad algorithm se statickým algoritmem řízení. Tento algoritmus sice funkce nově vzniklých komponent neovlivňují, ale je nově umožněno sbírat ve Statistics collectoru data i se senzorů jako u Traffic control algorithm VASC. Pak lze dělat alespoň statistiku projíždějících vozidel pro porovnání obou algoritmů.

4.2 Implementace

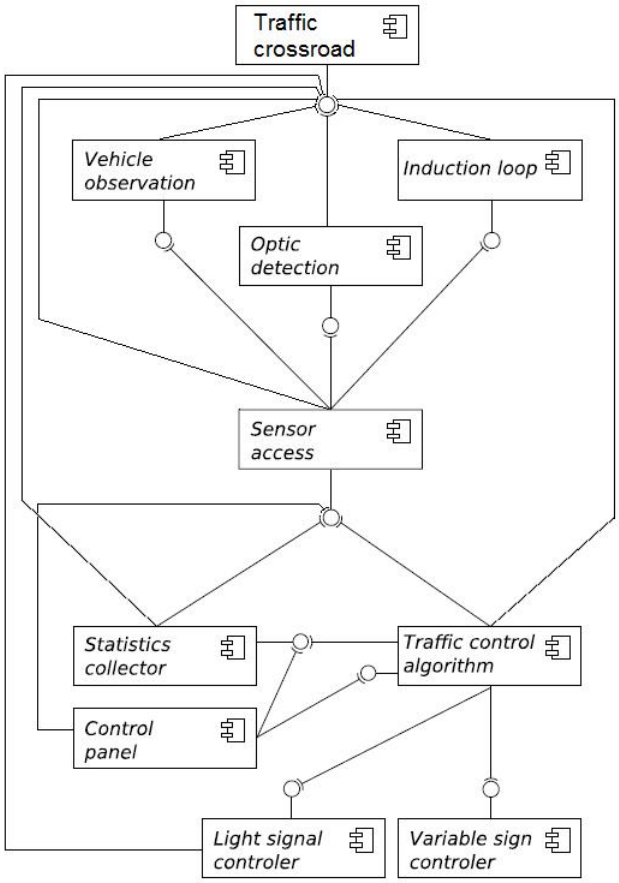
4.2.1 Seznámení s prostředím

Pro vývoj aplikace bylo použito vývojové prostředí Eclipse pro J2EE verzi „Eclipse IDE for Java EE Developers“. Toto prostředí bylo třeba nakonfigurovat pro práci se Spring DM, k čemuž stačilo doinstalovat Spring Source Tool Suite, importovat aktuální verzi balíků Spring Dynamic Modules a importovat z nich bundly.

Pro vytvoření grafů statistik byla do Control Panelu importována opensourcová knihovna JFreeChart pro programovací jazyk Java.

4.2.2 Schéma komponentové aplikace

Konečné schéma implementace komponentové aplikace je vidět na Obr.4.2. Komponenta Traffic control algorithm je znázorněna obecně pro oba algoritmy.



Obrázek 4.2: Výsledný implementovaný komponentový model aplikace

Jsou na něm zobrazeny všechny implementované komponenty a znázorněna komunikace, která probíhá pomocí interfaců mezi nimi. Kromě Control Panel každá komponenta implementuje svůj interface, kde jsou jasně definovány metody, které komponenta poskytuje. Téměř všechny komponenty komunikují s rozhraním hlavní komponenty Traffic crossroad, která jim poskytuje všechny potřebné metody pro jejich funkci.

4.2.3 Traffic crossroad

Zde je obsaženo grafické uživatelské rozhraní, které ze souboru načte a zobrazí křižovatku a pravděpodobnosti generování vozidel na jednotlivých pruzích. Po spuštění simulace se rozeběhne vlákno generující běh křižovatky.

Traffic crossroad poskytuje téměř všem komponentám rozhraní metod postačujících pro řízení křižovatky. jako jsou počty aut v pruzích, nastavení světel na semaforech, aktuální čas.

4.2.4 Traffic control algorithm

Obsahuje spoustu metod načtení pruhů, správné přepínání oranžové na semaforu, ale hlavně obsahuje algoritmus — definici fází pro přepínání světel. Běží na samostatném vlákně a pomocí načítání času z vlákna komponenty Traffic crossroad přepíná fáze semaforu v přesných intervalech.

4.2.5 Traffic control algorithm VASC

Stejně jako předchozí komponenta běží na samostatném vlákně a mění fáze podle aktuálního času z Traffic crossroad. Narozdíl od předchozí komponenty, využívá zprávy ze Sensor Access. Vybere si jeden ze senzorů a dostane informaci o tom, zda se vozidla nachází v pruzích, tuto informaci pak využívá při prodlužování intervalů.

4.2.6 Light signal controller

Implementuje nastavení konkrétních světel na konkrétních semaforech, které mu poskytuje Traffic Crossroad. Obsahuje metodu pro získání doby trvání oranžové na semaforu podle toho, jestli je před zelenou, nebo před červenou.

4.2.7 Variable sign controller

Tato komponenta je součástí komponentové aplikace a je připravena k použití, nicméně neobsahuje žádnou implementaci, protože pro současnou implementaci Traffic crossroad nemá uplatnění.

4.2.8 Induction loop

V Induction loop je implementována metoda na zjištění, zda je vozidlo v pruhu a metoda počítání vozidel.

Pro potřeby Statistics collectoru mají všechny tři senzory metodu pro resetování zaznamenaných počtů aut.

4.2.9 Optic detection

Obsahuje metody pro zjištění počtu vozidel a detekování přítomnosti alespoň jednoho vozidla na silnici. Metoda umí rozpoznat jedno, dvě nebo tři vozidla. Při větším počtu vozidel rozpozná metoda maximálně 4.

4.2.10 Vehicle observation

Obsahuje metody stejné jako Optic Detection s tím rozdílem, že umí detekovat všechna vozidla na vozovce. Předpokládá se, že by zde bylo možné udělat funkční součinnost s jedním či více pozorovateli.

4.2.11 Sensor access

Obdrží data z komponent senzorů. Je spojen s Control panelem pro výběr nastavení senzoru pro algoritmus a pro výběr nastavení senzoru pro statistiky. Těmto komponentám pak počty resp. informaci o obsazení pruhu.

4.2.12 Statistics collector

Tato komponenta běží na vlákně, sčítá auta na každém pruhu díky datům ze Sensor Access a načítá si aktuální čas z Traffic crossroad. Tato data ve formátu identifikátor pruhu, počáteční čas, konečný čas a počet aut ukládá do souboru XML. Ukládání probíhá pomocí knihovny `java.beans.XMLEncoder`. Toto ukládání je zajištěno ve třídě `OutputData.java`. Do jména XML souboru se přidává datová značka pomocí níž se pak zobrazuje graf Control

Panel, každé uložení se provede do nového souboru.

4.2.13 Control panel

Obsahuje grafické uživatelské rozhraní, které umožňuje na začátku před spuštěním simulace křižovatky uživateli nastavit intervaly pro 4 fáze algoritmu VASC. Umožňuje vypnout řízení křižovatky, čímž začnou po půl vteřině blikat oranžově světla. Umožňuje nastavit interval po jakém se budou automaticky ukládat data. Má i tlačítko na zapnutí nebo vypnutí sběru statistik a tlačítko na uložení konkrétních statistik do zvoleného souboru. Na panelu je možné nastavit ze kterého senzoru budou sbírány statistiky a ze kterého bude probíhat detekce vozidla na pruhu pro běh algoritmu.

5 Algoritmy řízení křižovatky

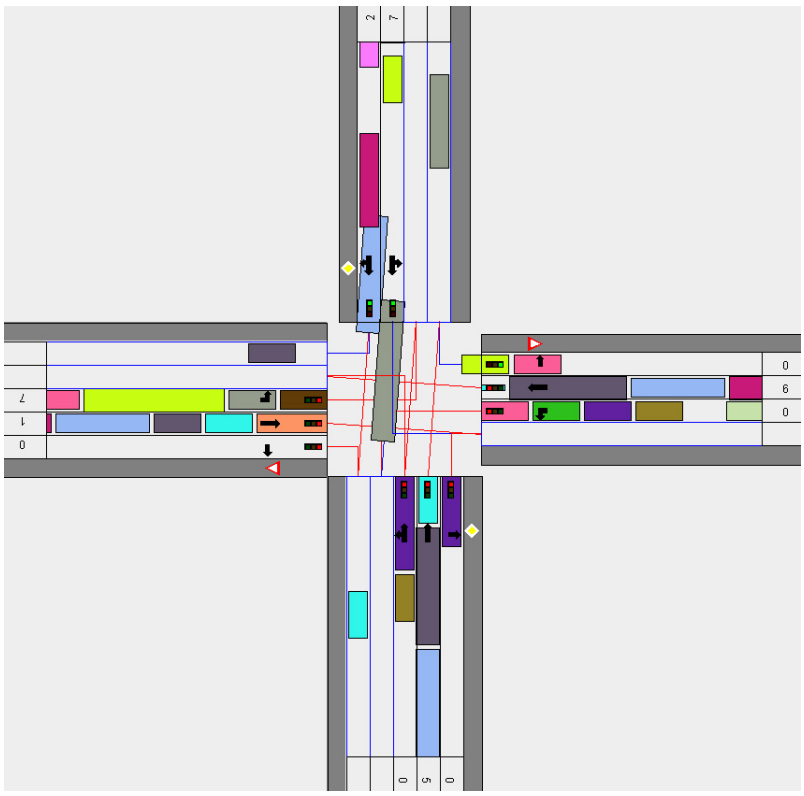
Komponentová aplikace zahrnuje dvě komponenty, které obsahují implementaci algoritmu pro řízení křižovatky. Tyto komponenty nemohou být v aplikaci spuštěny najednou, ale jedna se dá plnohodnotně nahradit druhou. Zadáním bylo implementovat statický algoritmus do jedné komponenty Traffic control algorithm a algoritmus VASC pro řízení vybrané silniční křižovatky do druhé komponenty Traffic control algorithm VASC. Testovací křižovatka, na které jsme měli vyzkoušet implementaci zmiňovaných dvou algoritmů řízení provozu, byla získána ze souboru `map22.xml` dodaného k projektu.

5.1 Seznámení se s křižovatkou

Vybraná křižovatka se nachází v Plzni v místě kde se křižují ulice Klatovská a Sukova. Pro lepší pochopení řízení této křižovatky jsem ji byl pozorovat ještě předtím než jsem začal s implementací a zaznamenal jsem si, jakým způsobem se v reálu řídí. Bohužel se ukázalo, že skutečná křižovatka neodpovídá úplně křižovatce viz Obr.5.1, kterou simuluje komponenta Traffic crossroad. Počet pruhů v každém směru byl sice totožný, ale odbočovací směry se ve třech pruzích lišily. Takový detail je při navrhování změny fází křižovatky poměrně zásadní. I tak jsem se pro návrh způsobu řízení křižovatky inspiroval.

5.2 Statický algoritmus

Jak statický, tak i dynamický algoritmus vycházejí ze stejného principu řízení křižovatky, jak je popsáno v 3.5.1. Tímto principem je střídání několika fází křižovatky — v každé jsou nastaveny všechny semafory(v našem případě jich je 11) na takovou barvu, aby nemohlo dojít ke kolizi vozidel. Všechny fáze běží v časové smyčce a v případě statického řízení mají nastavenou pevnou dobu trvání, která zůstává konstantní. Počet fází řízení křižovatky je obvykle sudý — používají se 2 nebo 4 fáze. Pro naši křižovatku byl na základě pozorování reálné křižovatky navržen počet fází na 4. Jednotlivé fáze pro Traffic control algorithm můžeme vidět na čtyřech obrázcích viz Příloha A.2. Jsou na nich vyobrazena rozsvícená zelená světla v konkrétních pruzích v každé ze čtyř fází. V každém stavu jsou rozsvíceny 3 nebo 4 zelené. Přesně podle



Obrázek 5.1: Ilustrační obrázek simulace provozu na testované křižovatce

toho, jak je vidět na obrázcích, jsou jednotlivé fáze implementovány stejně v obou komponentách. Délka trvání jedné fáze může být různá. Není dobré nechávat čekat jednotlivé pruhy dlouhou dobu. Na druhou stranu ani nastavení příliš krátké doby intervalu — cca do 10 sekund není ideální. Simulovaná křižovatka byla implementována tak, že vozidla nemohou vyjet do křižovatky na oranžovou. To znamená, že 3 sekundy křižovatka stojí, než uplyne jednosekundová oranžová před zelenou a dvousekundová oranžová před červenou. To sice zabraňuje tomu, že na křižovatku nevjedou dvě vozidla najednou do kolize, nicméně při krátkém intervalu trvání fáze jsou při fungování algoritmu 3 sekundy stání vůči například 5 sekundám zelené znát. Navíc když jsou vozidla často zastavována, musí se pomalu rozjíždět, což je oproti plynulejšímu průjezdu na zelenou další ztráta.

Algoritmus by proto měl být horší než dynamický VASC, který byl implementován v komponentě Traffic control algorithm VASC.

5.3 VASC algoritmus

VASC je dynamický algoritmus, který se v ničem neliší od statického algoritmu až do chvíle, kdy vyprší minimální interval fáze. V tomto okamžiku přichází na řadu senzor, který ve smyčce periodicky detekuje, zda je v pruzích, které mají právě zelenou, alespoň jedno vozidlo. Takto kontroluje až dokud neodjedou všechna vozidla z pruhů, nebo dokud nevyprší maximální interval trvání fáze. V případě, že ani v jednom pruhu není po vypršení minimálního intervalu trvání fáze, přepne algoritmus na další fázi. Délka kroku — času po kterém se senzory snaží detekovat se nastavuje na minimální dobu, neboť by nemělo význam čekat ve fázi na prázdné pruhy. V případě VASC si můžeme dovolit nastavit minimální interval klidně jen na 5 sekund, protože je předpoklad, že v pruhu bude alespoň v jednom z pruhů detekováno alespoň jedno vozidlo, které svou přítomností prodlouží délku trvání intervalu. Nemělo by se tak stát jako u statického algoritmu, že bude při nastavení malého minimálního intervalu trvání fáze oranžové tvořit velkou procentuální část trvání fáze.

Nastavení světel jednotlivých fází v komponentě Traffic control algorithm VASC vypadá stejně jako u statického, opět viz Příloha A.2.

5.4 Nastavení parametrů pro algoritmy

Provoz na křižovatce je generován exponenciálně pravděpodobnostním rozdělením ze třídy ExponetialTrafficGenerator v komponentě Traffic crossroad. Zde je velmi důležité nastavit citlivě parametr lambda, protože na něm může záviset úspěšnost algoritmů. Tento parametr mi bylo doporučeno nastavit na hodnotu lambda=0,01, která by měla odpovídat skutečné hustotě provozu. Tímto by mělo být zajištěno, že algoritmus bude moci efektivně řídit provoz a nedojde k zácpě jen kvůli špatnému nastavení hustoty provozu.

Správné nastavení intervalů jednotlivých fází není vůbec jednoznačnou záležitostí. Tomuto problému se budeme více věnovat v kapitole 6.

Pro statický algoritmus lze v komponentě Traffic control algorithm nastavit v atributech třídy konstantní intervaly trvání fází. Je doporučeno nastavit délku intervalu přibližně 25 sekund a v závislosti na konkrétní fázi sekundu nebo dvě ubrat kvůli malému vytížení pruhu.

Pro VASC algoritmus je možné v komponentě Control panel nastavit v grafickém uživatelském rozhraní před spuštěním simulace křižovatky ručně maximální a minimální délku jednotlivých fází. Doporučená minimální délka intervalu je, jak už dříve bylo uvedeno — 5 sekund. Maximální délka intervalu by měla být nastavena delší než ve statickém algoritmu(přibližně 27 sekund), aby bylo umožněno projet více autům v případě, že je pruh přeplněný.

Je víceméně jasné, že rozdíly mezi intervaly pro jednotlivé fáze nemohou být nastaveny velké, protože by se část křižovatky ucpala. Maximem by měli být přibližně 3 sekundy mezi nejdelším a nejkratším intervalem fáze u statického algoritmu a 3 sekundy rozdílu mezi maximálním intervaly fází u VASC. Hodnoty byly odzkoušeny na simulované křižovatce.

6 Testování kvality mimofunkčních požadavků

U této komponentové aplikace nás samozřejmě zajímá nejvíce testování dvou implementovaných algoritmů pro řízení křižovatky.

6.1 Úspěšnost řízení křižovatky

Necháme-li běžet simulaci provozu křižovatky několik tisíc sekund, zjistíme, že křižovatka se při nastavení parametru pro hustotu provozu lambda=0,01 většinou zahltí[1].Nejnaplněnějšími pruhy naší křižovatky byly téměř vždy prostřední pruhy a naopak nejprázdnější jsou ty, co mohou odbočit pouze doprava, ty jsou totiž obsaženy ve více fázích.

Nemyslím si však že je to vinnou implementace a ani algoritmů řízení. Je to spíše tím, že křižovatka je řízena náhodnou veličinou, která může způsobit, že se zahltí více pruhů a z tohoto stavu už se algoritmus nedostane. Pokud je nastavena lambda=0,009, je už provoz moc řídký a řízení dopravy de facto permanentně plynulé. Přesto jsem testoval většinu spuštění s hodnotou lambda=0,01. Takto vypadají statistiky aut ve spojovém grafu pomocí řízení statického a VASC algoritmu viz příloha A.5 a .

6.2 Rozdíl mezi statickým algoritmem a VASC

Rozdíly mezi grafy obou algoritmů nejsou příliš velké. To je z důvodu velké hustoty provozu, tudíž VASC algoritmus neuplatní svoji vlastnost zkrácení intervalu a umožnění přepnutí do další fáze v jejich prospěch. Při velkém provozu se tedy rozdíl projeví až po dlouhé době, kdy mi vyšlo, že po 34000 sekundách je algoritmus VASC má o jedno procento projetých počtů aut více viz tabulka6.1

Při hustém provozu se tedy VASC algoritmus začíná chovat podobně jako

[1]Na křižovatce jsou auta za pruhy generována do počtu 51. Po dosažení takového počtu vozidel ve více pruzích z jiných fází můžeme považovat křižovatku za zahlcenou.

Pruh	VASC algoritmus				Statický algoritmus			
	Čas [10 000s]		Čas [34 000s]		Čas [10 000s]		Čas [34 000s]	
	Počet aut	Průměr	Počet aut	Průměr	Počet aut	Průměr	Počet aut	Průměr
1.	1015	101,5	3400	100	1033	103,3	3426	100,7
2.	991	99,1	3439	101,1	1000	100	3383	99,5
3.	935	93,5	3355	98,7	1016	101,6	3313	97,4
4.	959	95,9	3318	97,6	922	92,2	3194	93,9
5.	1002	100,2	3370	99,1	929	92,9	3204	94,2
6.	967	96,7	3338	98,1	979	97,9	3426	100,7
7.	967	96,7	3310	97,3	1019	101,9	3379	99,3
8.	1004	100,4	3356	98,7	968	96,8	3334	98,0
9.	1002	100,2	3355	98,7	999	99,9	3444	101,2
1.	970	97	3283	96,6	939	93,9	3218	94,6
11.	969	96,9	3340	98,2	927	92,7	3192	93,8
$\sum$	10781	98,0	36864	98,6	10731	97,6	36513	97,6

Obrázek 6.1: Tabulka k porovnání obou algoritmů při sečtení celkového počtu projetých aut při 10000 sekundách a 34000 sekundách

statický algoritmus. Z odkázaných grafů A.5 a v příloze ale přece jenom můžeme něco málo vyčíst. Je trochu patrné, že statický algoritmus má o něco více výchylek, což by mohlo poukazovat na dočasné zahlcení a tudíž menší plynulost v dopravě. V zásadě ale více vyčteme z běhu aplikace, kde má opravdu ve většině případů spuštění křižovatky VASC rovnoměrnější rozdělení počtů aut v pruhu. Je to dáno tím, že když má VASC plný pruh, pustí co nejvíc aut může, zatímco statický zapne předčasně červenou. Z toho plyne i to, že se statický algoritmus velmi špatně dostává ze zahlcení. U VASC je daleko větší šance, že se mu povede uvolnit zahlcený pruh.

Můžeme tedy celkově prohlásit, že VASC je tak teoreticky lepší algoritmus, byť jen o velmi málo co se týká celkového počtu projetých aut křižovatkou za časový interval.

6.3 Možnosti vylepšení

Protože senzory v komponentové aplikaci mají poměrně jednoduchou implementaci, bylo by nejspíš možné dosáhnout větší úspěšnosti zlepšením detekce vozidla na konkrétní pozici ve vozovce. Tím bychom zabránili ztrátám

času, když je za autem dlouhá mezera, ale přesto je v pruhu detekováno další auto. Z toho vyplývá, že kvalita služeb detekce všech senzorů by měla být větší, ačkoliv senzory plní svoji primární funkci spolehlivě. Dozvěděli jsme se také, že data zobrazující údaje o počtu aut, nevypovídají úplně stoprocentně o plynulosti provozu.

6.4 Možnosti dalšího vývoje

Vzhledem k tomu, že tato komponentová aplikace pracuje na více vláknech, je třeba se zaobírat problémem využívání paměti. Může nastat situace, že aplikace bude použita na křižovatce v rámci nějakého zastaralého hardwaru a bude na ní vznesen mimofunkční požadavek na schopnost omezené práce s pamětí.

Další komponenty nemají až takový vliv pro samotné algoritmické řízení křižovatky, aby je bylo nutné podrobit nějakým smysluplným testům kvality.

7 Závěr

V rámci této bakalářské práce jsem se seznámil se základy komponentového programování a podařilo se mi implementovat navrženou komponentovou aplikaci silniční křižovatky. Vývoj aplikace probíhal ve dvou etapách. Nejdříve jsem vytvořil jednodušší aplikaci sestávající se jen ze tří komponent obsahujících komponentu se statickým algoritmem pro řízení silniční křižovatky. V druhé etapě jsem rozšířil aplikaci na 10 komponent. Komponenty s algoritmem řízení křižovatky jsou dvě a každou z nich lze ve výsledné aplikaci použít. Druhá z nich implementuje dynamický algoritmus VASC, který využívá ke své činnosti ostatní komponenty. Jeho správnou činnost, kterou můžeme vidět v grafickém uživatelském rozhraní simulované křižovatky, potvrzuje správnou komunikaci komponent. Současně s rozhraním křižovatky se spouští ovládací panel, ve kterém je nastavení parametrů algoritmu VASC nebo sběru statistik z křižovatky.

Hlavní cíl — implementace dvou algoritmů řízení křižovatky byl splněn. Obě komponenty obsahující algoritmy jsou při správném nastavení parametrů schopny plynule řídit provoz. Při větší hustotě provozu se výhody algoritmu VASC vůči statickému algoritmu smazávají. Nicméně dlouhodobě zůstává algoritmus VASC o něco efektivnější. Tyto poznatky jsem načerpal při testování různých možností nastavení intervalů VASC a sledováním chování křižovatky buď z grafu, nebo z animace křižovatky. Při dalším vývoji aplikace by se dalo pokračovat například vylepšením citlivosti senzorů, který by podával konkrétnější informace o aktuální poloze vozidel pro větší zefektivnění algoritmu VASC.

Výsledná komponentová aplikace by mohla být v budoucnu využita k testování v simulačním frameworku vyvíjeného KIV.

Seznam obrázků

Seznam zkratek

- **KIV** — Katedra informatiky a výpočetní techniky
- **CBD** — *Component-based Develompment* — komponentově orientovaný vývoj softwaru.
- **CBSE** — *Component-based Software Engineering* — Komponentově orientované softwarové inženýrství
- **VASC** — *Vehicle actuated signal control* — Dynamický algoritmus pro řízení křižovatky
- **SSZ** — Světelná signalizační zařízení

Literatura

[1] Clemens Szyperski: *Component Software Beyond Object—Oriented Programming (2nd Edition)*, Addison—Wesley Professional, November 23, 2002
Dostupné z `http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage`

[2] Bachmann, Felix; Bass, Len; Buhman, Charles; Comella—Dorda, Santiago; Long, Fred; Robert, John; Seacord, Robert; Wallnau, Kurt.: *Volume II: Technical Concepts of Component—Based Software Engineering, 2nd Edition*, Software Engineering Institute, Carnegie Mellon University, 2000. Dostupné z `http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage`

[3] Szyperski, C., Gruntz, D., Murer, S.:*Component Software – Beyond Object—Oriented Programming* , ACM Press, New York (2000)

[4] Becker, S., Koziolek, H., Reussner, R.:*The Palladio component model for model—driven performance prediction.*, In: Journal of Systems and Software, Vol. 82, No. 1, pp.3–22 (2009)

[5] Crnkovic, I., Larsson, M.: *Building Reliable Component—Based Software Systems [Hardcover]*, Artech House; 1st edition (July 15, 2002) Dostupné z `http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage`

[6] Rubio, D.:*Pro Spring Dynamic Modules for OSGiTM Service Platform*, Apress, USA, (2009)

[7] Slobodan Guberinič, Gordana Šenborn, Bratislav Lažič: *Optimal Traffic Control Urban Intersections*, CRC Press 2008

[8] Projektanti: *Systémy silniční dopravní technologie*,web technologie.vhd.cz/ssz-projekt.php

[9] Tomáš Padělek: *Základy dopravního inženýrství* Dostupné z `http://www.fd.cvut.cz/personal/padeltom/.../ZDIR\%2009\%20T6.pdf`

[10] Dostupné z `http://preference.prazsketramvaje.cz`

[11] Vladimír Faltus: *Aplikace Petriho sítě při modelování dynamického řízení křižovatek.*, Automatizace – Ročník 48 – Číslo 2 – únor 2005 Dostupné z `http://www.automatizace.cz/article.php?a=530`

[12] Tomas Potuzak, Richard Lipka, Jaroslav Snajberk, Premek Brada, Pavel Herout: *Design of a Component-based Simulation Framework for Component Testing using SpringDM*, University of West Bohemia, Pilsen, Czech Republic

A Přílohy

A.1 Uživatelská dokumentace

A.1.1 Instalace workspace

Ke spuštění programu stačit připojit workspace do Eclipse, na CD je soubor map22.xml, ve kterém jsou uloženy data o křižovatkách

A.1.2 Konfigurace komponenty

Na přepínání mezi statickým a VASC algoritmem pro řízení křižovatky je nutné v komponentě 1.ControlPanel zakomentovat všechny dosud používané metody a atributy daného rozhraní Konkrétně při přepnutí z VASC na statický algoritmus zakomentujeme:

import cz.zcu....ITrafficControlAlgorithmVASC public ITrafficControlAlgorithmVASC itca metodu public void setTrafficCvasc těla metod všech setrů intervalu a odkomentujeme

public ITrafficControlAlgorithm itca metodu public void setTrafficCA v případě opačného přepnutí postupujte opačně

2. V manifestu.mf u ControlPanelu v záložce Dependenciespřidáme Imported package cz.zcu.kiv.crossroadcontrol. + jmeno požadovaného algoritmu řízení a odebereme používaný algoritmus

3. Poté opět ve složce META-INF v komponentě ControlPanel v podsložce spring otevřeme CPconfig.xml kde přidáme značky komentářů(<!– –>)k <property name=”trafficCvasc”ref=”RozhraniTCAVASC”/> <osgi:reference id=”RozhraniTCAVASC”interface=”cz.zcu....ITrafficControlAlgorithmVASC”/> a odeberem u

<osgi:reference id=”RozhraniTCA”interface=”cz.zcu.....ITrafficControlAlgorithm”/> <property name=”trafficCA”ref=”RozhraniTCA”/>

4. V ControlPanel properties přidat v java buildy požadovanou kompo-

nentu a odebrat současnou

5. Control panel – run as – run configuration – nastavit požadovanou komponentu a odebrat současnou

6. spustit program

A.1.3 Ovládání aplikace

Probíhá intuitivně pomocí tlačítek z gui křižovatky a dále pomocí ovládacího panelu obsluhovat funkce komponent:

Před spuštěním

Nastavení Intervalů — Před spuštěním křižovatky si uživatel může nastavit velikosti jednotlivých fází.

Interval uložení — Slouží k nastavení času v sekundách, po kterých probíhá automatické ukládání statistik do souboru.

Nastaveni senzorů pro algoritmus řízení — Toto nastavení je pak uloženo a je s ním počítáno při chodu programu.

Nastavení senzorů pro sběr statistik — Zvolí senzor pro senzor pro sběr statistik do souboru.

Za běhu

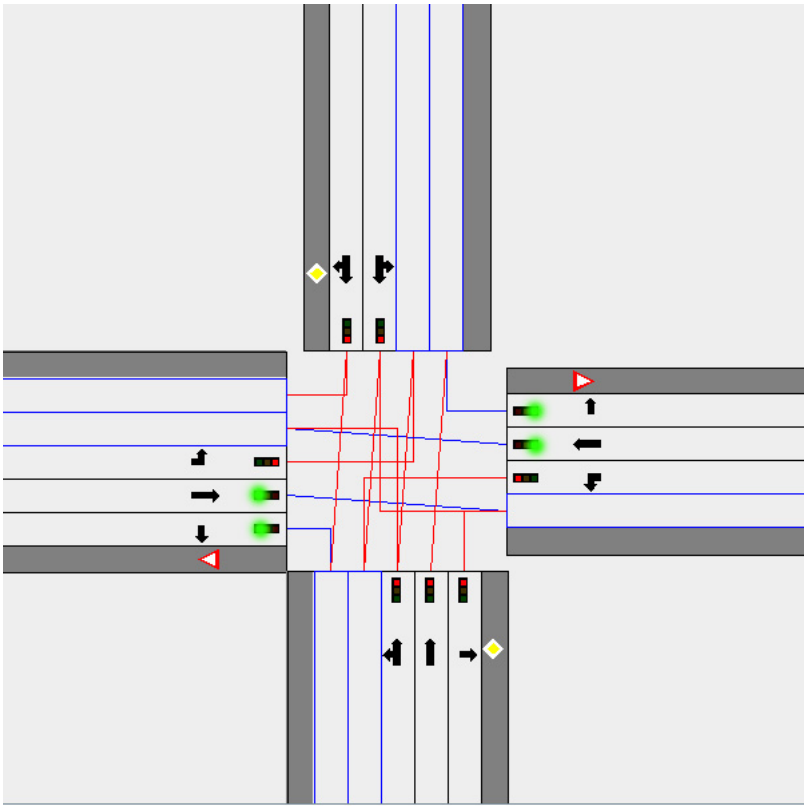
Zapnout/Vypnout řízení křižovatky — Při běhu programu si uživatel může vypnout řízení křižovatky, na semaforech začne blikat oranžová. Na zapnutí se všude nastaví červená a pokračuje fáze, která měla běžet.

Vypnutí/Zapnutí sběru dat — Od spuštění probíhá ukládání dat v intervalech podle intervalu uložení. Data se ukládají do adresáře XMLstats ve formátu XML a do jména souboru je vložena datová značka času uložení souboru, která pak slouží pro čtení.

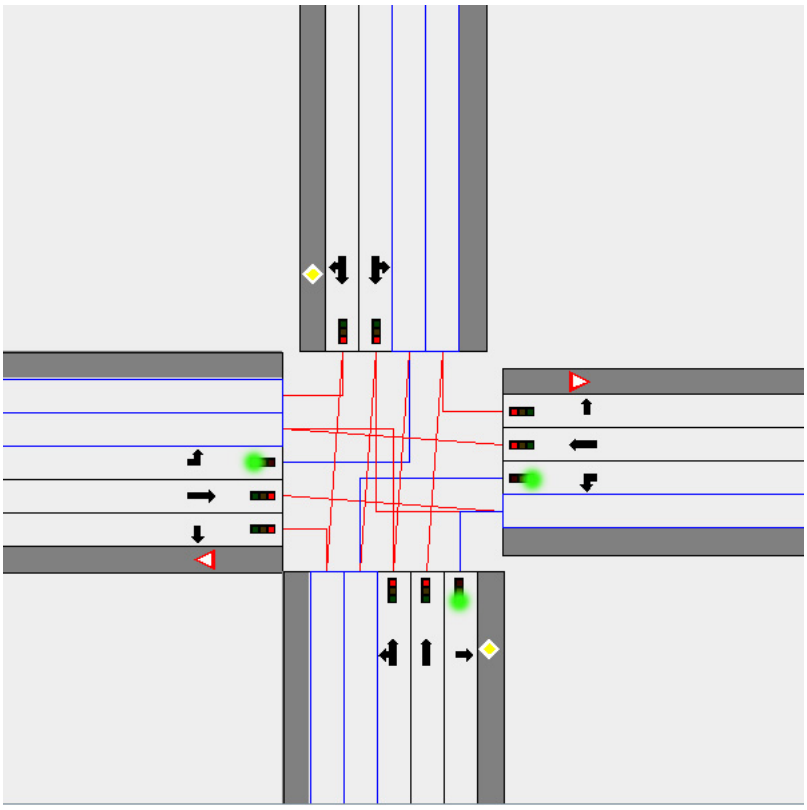
Ulož data — Uloží soubor do uživatelem zvoleného adresáře s údaji od

posledního automatického uložení, pro zahrnutí do statistik, musí být soubor uložen do adresáře XMLstats *Zobraz statistiku* — Zobrazí spojnicový graf s počtem projetých aut v každém pruhu v čase

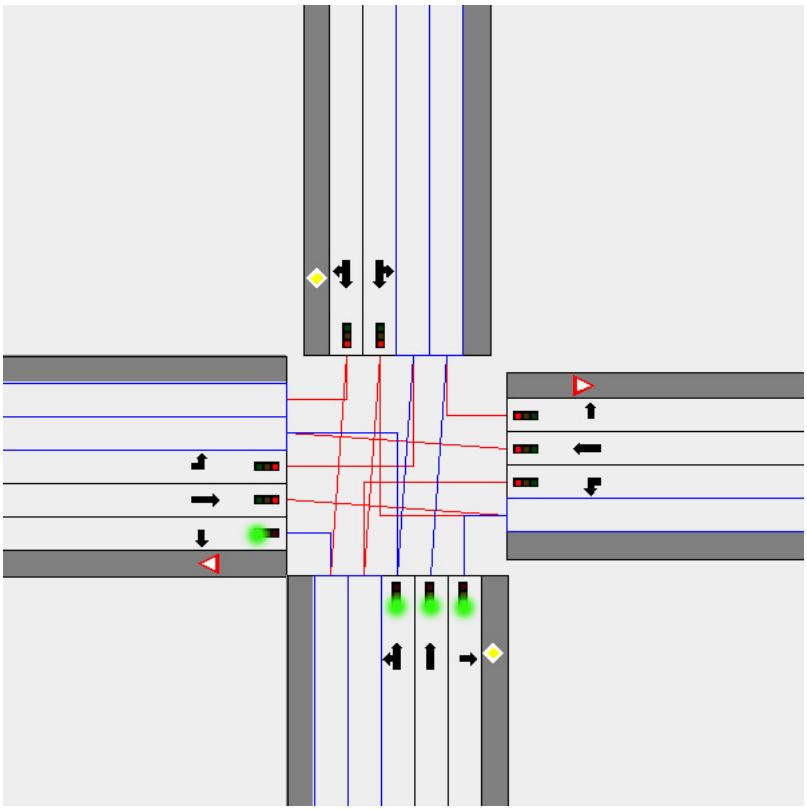
A.2 Fáze řízení křižovatky



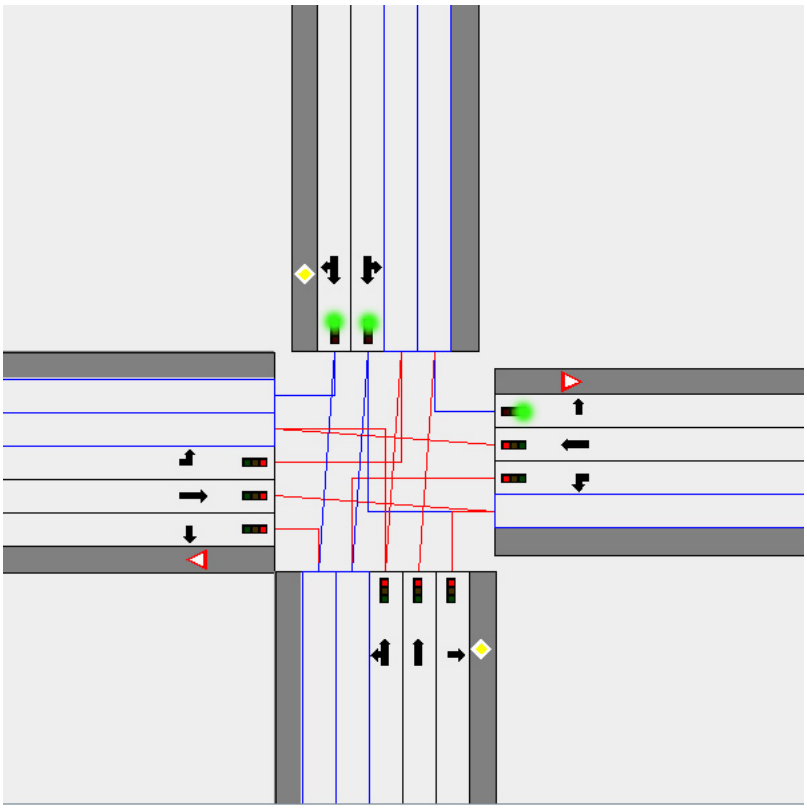
Obrázek A.1: první fáze nastavení světel na křižovatce



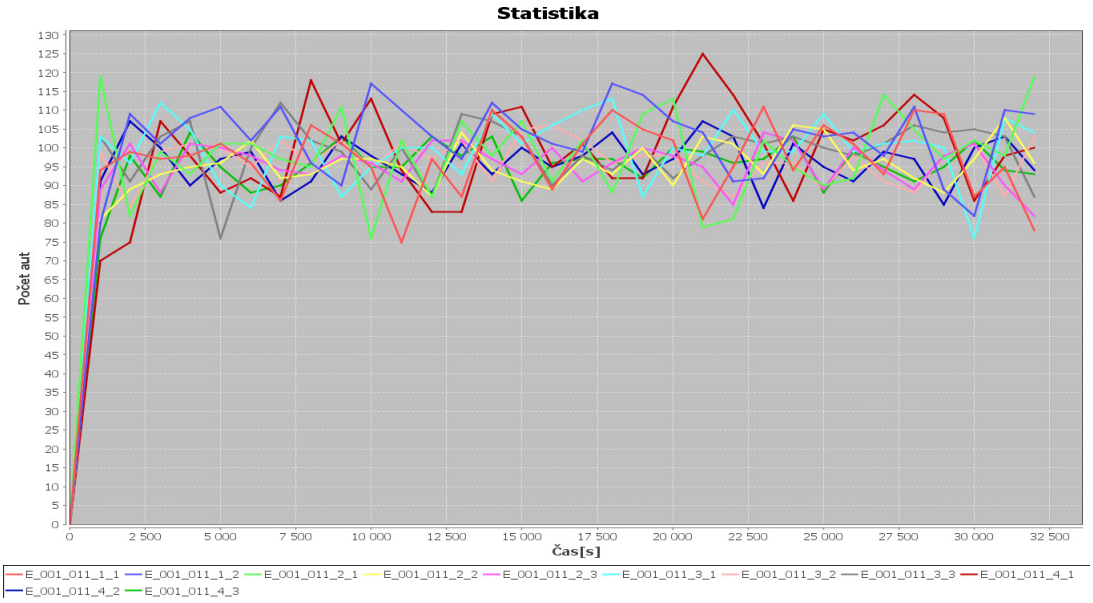
Obrázek A.2: druhá fáze nastavení světel na křižovatce



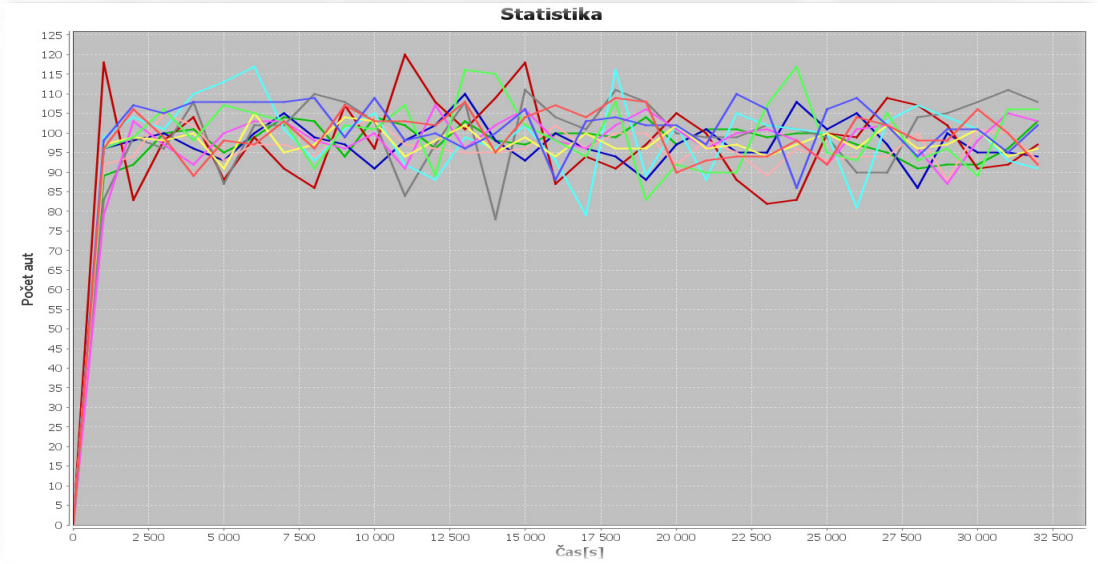
Obrázek A.3: třetí fáze nastavení světel na křižovatce



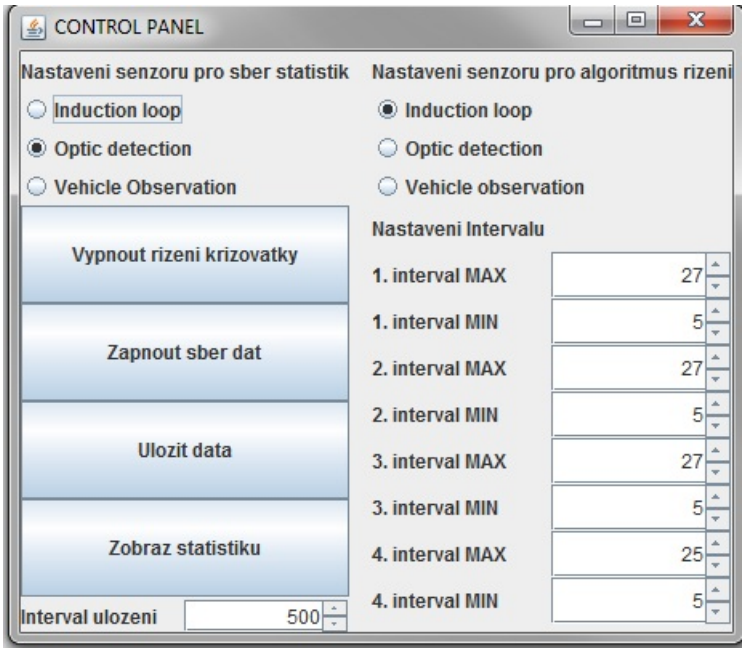
Obrázek A.4: čtvrtá fáze nastavení světel na křižovatce



Obrázek A.5: Statistika počtu aut v jednotlivých pruzích během 34000 sekund statického algoritmu



Obrázek A.6: Statistika počtu aut v jednotlivých pruzích během 34000 sekund algoritmu VASC



Obrázek A.7: ovládací panel