# Geoprostorna vizualizacija standardiziranih prometnih podataka

**Jakovljević, Dinko**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# GEOPROSTORNA VIZUALIZACIJA STANDARDIZIRANIH PROMETNIH PODATAKA

**Diplomski rad**

**Dinko Jakovljević**

**Osijek, 2016.**

# Sadržaj

# 1. UVOD

Korištenjem modernih ICT (*informacijska i komunikacijska tehnologija*) rješenja moguće je prikupiti različite prometne podatke sa senzora koji se nalaze na prometnicama te dobiti točniji prikaz stanja na prometnicama. Prikupljeni podaci su korisni vladama, sustavima za prometni nadzor, znanstvenim istraživanjima, službama za spašavanje, policiji itd. Integracija senzora za prikupljanje prometnih podataka čini jezgru ITS (engl. *Intelligent Transportation System*) informacijskog sustava koji omogućava dohvaćanje podataka sa servera i njihovo slanje klijentima koji su registrirani u sustavu [1]. Podaci sa senzora, kao što je brojilo prometa, se prikupljaju, analiziraju i vizualiziraju te se ovisno o anomalijama u podacima mogu pokrenuti različite akcije koje će za rezultat imati odziv sustava, bilo da se radi o pozivu žurne službe ili saniranju kolnika. Uzimajući u obzir navedeno moguće je razviti platformu za donošenje odluka unutar sustava za upravljanje prometom. Sustav bi mogao pokrenuti odgovarajuće rutine na osnovu podataka sa senzora. Postojeći ITS format podataka karakterizira raznovrsnost i kompleksnost koja ne može odgovoriti zahtjevima [1]. Spremanje prometnih podataka zahtjeva izradu standarda koji bi omogućio jedinstvenu komunikaciju između različitih sustava i njihovo spremanje. Povećanje potražnje za podacima u stvarnom vremenu dovelo je do razvoja DATEX II standarda za razmjenu prometnih podataka. DATEX II je standardiziran od strane CEN-a (engl. *European Committee for Standardization*) te ga koristi nekoliko zemalja unutar Europe [1], [2], [3]. Postoji potreba za centraliziranim sustavom koji će omogućiti programerima integraciju novih sustava u postojeći te korištenje standardiziranog oblika spremanja i slanja prometnih podataka. U diplomskom radu je opisan razvijeni programski dodatak (engl. *Plugin*) koji omogućuje dohvaćanje, obrađivanje i vizualiziranje prometnih podataka u DATEX II formatu dobivenih od NTIS (engl. *National Traffic Information Service*) sustava. Vizualizacija podataka se vrši uz pomoć ResQMap GISa (*Geografski informacijski sustav*) koji je integriran sa C&C (engl. *Command and Control*) sustavom CoordCom [4], [5]. Podaci sa NTIS sustava se prikazuju u stvarnom vremenu (engl. *Real time*) na klijentskoj strani što omogućava brže i jednostavnije upravljanje prometnicama.

# 2. STANDARDI RAZMJENE PODATAKA

Prometni centri čine jezgru velikog broja ITS (engl. *Information Technology Service*) primjena. Centri su u većini slučajeva odgovorni za upravljanje velikom količinom prometa (na granici kapaciteta) te im usklađena i standardizirana razmjena osigurava efikasnosti pri izvođenju i menadžmentu. Informacijska tehnologija je postala sveprisutna u svakodnevnom životu od automobila i domova sve do prometnica, te je postala sama po sebi alat za upravljanje prometom. Postoji velika raznolikost izvora podataka kojima moderne prometne mreže raspolažu. Razmjena podataka bi trebala pratiti međunarodne standarde kako bi se izbjeglo fragmentiranje informacija. Većina standarda koji se koriste za razmjenu prometnih podataka dovode u pitanje komunikaciju i razmjenu podataka između različitih sustava više nego njihovo pohranjivanje unutar sustava. Standardi koji se većinom koriste za razmjenu podataka između sustava [6] su:

- Riječnik podataka za upravljanje prometom TMDD (engl. *The Traffic Management Data Dictionary*)
- Riječnik podataka za P1512 upravljanje incidentima (engl. *The P1512 Incident Management Data Dictionary*)
- Jezik za opisivanje prometnog modela TMML (engl. *Traffic Model Markup Language*)
- Geografski opisni jezik GML (engl. *Geographic Markup Language*)
- Prometni podaci univerzalnog formata UTDF (engl. *Universal Traffic Data Format*)
- Digitalni geoprostorni metapodaci FGDC-STD-001-1998 (engl. *Digital Geospatial Metadata*)
- Europski standard za razmjenu prometnih informacija DATEX II (engl. *European Traffic Information Exchange Standard*)

Dostupnost podataka je ograničena više nego što bi trebala biti. Razlog ograničavanja dostupnosti podataka je spremanje velike količine podataka koja ovisi o veličini prometne mreže i mjestima mjerenje. Dnevna količina podataka kroz prometnu mrežu može biti jako velika, te može predstavljati ozbiljan financijski problem. Kako bi se smanjili troškovi skladištenja velike količine podataka, podaci se filtriraju te se spremaju samo relevantni podaci ovisno o projektu [6]. Primjerice kod ultrazvučnih brojila prometa se podaci spremaju u vidu srednjeg protoka vozila umjesto da se spremaju podaci za svako vozilo koje je prošlo pored brojila. Prikazivanje svakog vozila zahtjeva veliku količinu memorijskog prostora za spremanje podataka i vrlo brzi odziv sustava što za posljedicu ima veće financijske troškove.

# 3. DATEX 2 – ITS STANDARD

Datex 2 je dizajniran i razvijen od strane Europske radne skupine kao mehanizam za razmjenu prometnih podataka koji će standardizirati sučelja između prometne kontrole, informacijskih centara, pružatelja usluga, prometnih operatora i medijskih partnera kao što su radijski servisi, i sl. Postao je referenca za sve aplikacije koje se razvijaju i implementiraju u Europi, a vezane su uz razmjenu prometnih podataka. Standard omogućuje usklađen način prijenosa podataka između zemalja na sustavskoj razini što dovodi do većeg upravljanja Europskom mrežom prometnica. Datex 2 pokriva širok spektar podataka vezanih uz promet i transport [3]. Jedan od najvećih uspjeha Datex 2 standarda je uspostavljanje logičkog modela koji je široko podržan od strane korisnika iz cijele Europe.

## 3.1. Modeli podataka

Datex 2 pruža model podataka pod nazivom „*Level A data model*“ koji je nastao kao rezultat istraživanja podataka od strane korisnika diljem Europe. Postoje situacije u kojima koncept podataka nedostaje u rječniku podataka za određenu skupinu korisnika zbog konteksta u kojem se podaci koriste (npr. unutar neke države). U ovakvim slučajevima korisnicima se pruža nadogradnja (engl. *Extension*) modela u „*level B*“ koji će pružiti nepostojeće koncepte. Korisnici mogu primijeniti ograničeni set dobro definiranih UML (engl. *Unified Modeling Language*) mehanizama za nadogradnju B razine koji će sačuvati međusobno djelovanje sa standardnim Datex 2 sustavima. Drugim riječima, standardni sustavi (model razine A) će biti u mogućnosti obrađivati publikacije stvorene iz proširenih modela (model razine B) bez mogućnosti obrade nadograđenog sadržaja.

Model „A“ je vrlo sadržajan i obuhvaća veliki broj podataka za široku primjenu. U zadnje vrijeme pojavljuju se nove primjene na državnim i međunarodnim razinama koje žele dodati nove koncepte i svojstva u postojeće modele. Za nove primjene koje trebaju nadogradnju na postojeći „A“ model razvili su se B i C modeli. Ovakav pristp je omogućio razvoj specifičnih modela koji će nadopuniti „A“ model sa dodatnim i specifičnim informacijama i svojstivma. Novostvoreni modeli će imati mogućnost međudjelovanja sa „A“ modelom i sustavima koji koriste standardni model.

Nakon predstavljanja razine „A“ i razine „B“ modela podataka određeni korisnici još uvijek ne mogu pronaći način kako bi prilagodili još specifičnije podatke u model. Podaci se previše razlikuju od razine „A“ modela podataka što je dovelo do razvoja razine „C“ modela podataka. Implementacija razine „C“ modela podataka se smatra neskladna sa postojećim Datex 2 razinama

modela „A“ i „B“. Zaključak je da sustavi koji su usklađeni sa razinom „C“ ne mogu razmjenjivati informacije sa sustavima koji rade sa razinom „A“.

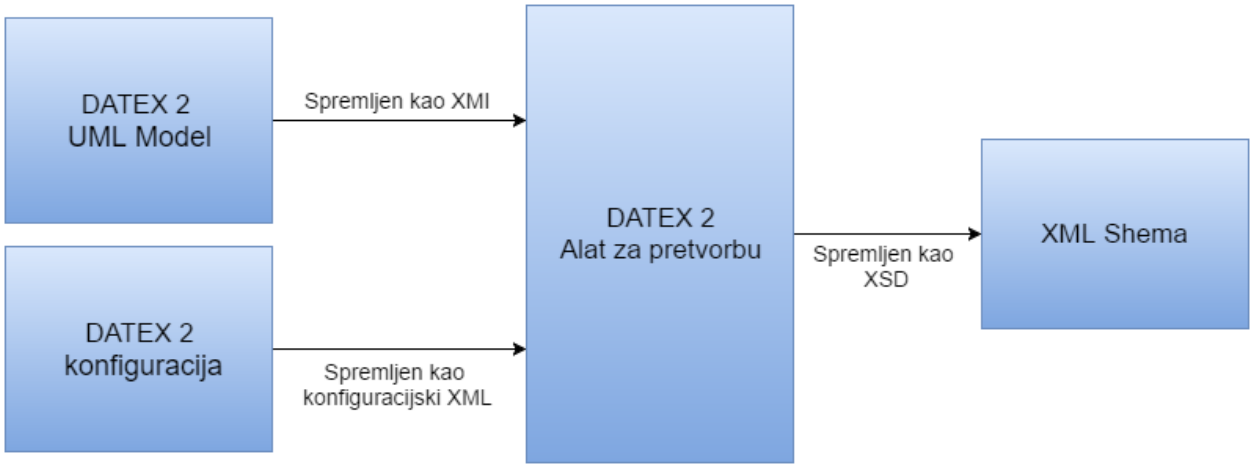
Prethodni Datex standard je pružao definicije koje su bile razumljive prometnim inženjerima i IT stručnjacima. Definicije podataka i sadržaj u Datex II standardu su spremljene u UML modelu koji nije lako dostupan korisnicima koji nisu IT stručnjaci. Razumljiviji riječnik podataka je dostupan kao programski alat koji automatski „izvadi“ definicije iz UML modela. Riječnik podataka je dostupan putem Datex 2 Internet stranice (http://www.datex2.eu). Datex 2 model obuhvaća sljedeće podatke:

- razinu usluge na prometnicama kao poruka za određenu situaciju ili kao dugoročno stanje u prometu,
- vrijeme putovanja, bilo da se radi između neposrednih ili udaljenih čvorova,
- sve vrste incidenata i nesreća,
- radovi na prometnicama,
- izvještaj o stanju prometnica,
- zatvorenost, blokade i začepljenja na prometnicama,
- vremenske prilike na prometnicama,
- sve vrste mjerenja na prometnicama (brzina, protočnost, itd.),
- javni događaji koji imaju utjecaj na promet [7].

## 3.2. XML shema

Danas je XML (engl. *Extensible Markup Language*) jedan od najčešćih načina za opis i razmjenu informacija između različitih aplikacija i sustava. Osnovni dio ovakvog pristupa je XML shema. U ovisnosti o modelu podataka i riječniku podataka XML shema je razvijena kako bi upotpunila određeno područje primjene. Shema je alat koji pomaže razumjeti i prikazati sadržaj podataka koji se razmjenjuju između sustava. Razvijen je alat koji omogućava pretvorbu UML Datex 2 modela podataka u XML shemu. UML model se pretvara iz UML alata za modeliranje (engl. *UML modelling tool*) u XMI datoteku (engl. *XML Metadata Interchange*) koja se zatim pretvara u XML shemu uz pomoć alata za pretvorbu kao što je prikazano slikom 3.1. XMI je OMG standard (engl. *Object management group*) za razmjenu informacija iz metapodataka putem XML

jezika. Može se koristiti za sve metapodatke čiji se metamodel može prikazati pomoću MOF (engl. *Meta-Object Facility*).




**Slika 3.1.** Pretvorba UML modela u XSD shemu

XMI se najčešće koristi kao format za razmjenu kod UML modela. Također se može koristiti za stavljanje u seriju (engl. *Serialization*) modela drugih jezika i metamodela [8].

## 3.3. Mehanizmi za razmjenu

Datex 2 omogućava dva načina razmjene podataka, a to su slanje (engl. *Push*) i dohvaćanje (engl. *Pull)*. Push način omogućava opskrbljivaču (engl. *Supplier*) slanje informacija klijentu dok pull način omogućava klijentu slanje zahtjeva (engl. *Request*) za preuzimanjem informacija od opskrbljivača. Razmjena podataka između izdavačkog sustava (engl. *Publisher system*) i klijenta može se postići na 3 načina:

- Način rukovanja 1 – sustav izdavač šalje podatke na događaj
    - Izdavač isporučuje podatke kada dođe do promjene u podacima
- Način rukovanja 2 – sustav izdavač periodički šalje podatke
    - Izdavač isporučuje podatke u cikličkom vremenu
- Način rukovanja 3 – Klijent dohvaća podatke
    - Klijent šalje zahtjev za preuzimanjem podataka na koji sustav izdavač odgovara

Svaki od načina rukovanja mogu raditi izvanmrežno (engl. *Offline*) i putem mreže (engl. *Online*). Postoje dva profila implementacije za „klijent dohvaća“ (engl. *Client Pull*) način rukovanja putem interneta. To su direktna upotreba HTTP/1.1 protokola ili pomoću mrežnog servisa (engl. *Web Service*) po HTTPu. Za način rukovanja „izdavač isporučuje“ podatke razvijena je platforma pomoću mrežnog servisa po HTTPu [7].

## 3.4. Klijent dohvaća podatke po HTTPu

Dohvaćanje podataka putem HTTP protokola je najjednostavnija implemetacija Datex 2 standarda iz nekoliko razloga:

- Ne postoji pretplata (engl. *Subscription*) na servis
- Izdavač periodično pruža podatke
- Podaci su dostupni putem URL poveznice za sve autorizirane klijenta

Različitim setovima podataka se može pristuputi putem različitih URL adresa na izdavačkoj strani. Opskrbljivački podsustav razrađuje podatke u skladu s Datex 2 XML shemom te ih dostavlja izdavačkom podsustavu koji ih čini dostupnima klijentima putem URL poveznice. Klijenti opskrbljivačkih sustava mogu biti:

- Klijentski sustav sa jednostavnim HTTP GET zahtjevom
- Klijentski sustav koji koristi mrežni servis

Da bi klijentski sustav mogao međusobno međudjelovati sa mrežnim servisom potrebno je podesiti sučelja za komunikaciju. Opskrbljivački sustav mora pružiti statičku SOAP omotnicu (engl. *Simple Object Access Protocol Wrapper*) definiranu sukladno SOAP imeniku. Zbog veće sigurnosti serveri najčešće zabranjuju korištenje većine TPC vrata (engl. *Port*). Jedina slobodna TCP vrata za upotrebu su pod brojem 80. Kako bi se osigurala velika razina međudjelovanja i upravljanje vatrozidom (engl. *Firewall*) potrebno je implementirati standardni HTTP port 80 [7].

## 3.5. Klijent dohvaća podatke pomoću mrežnog servisa

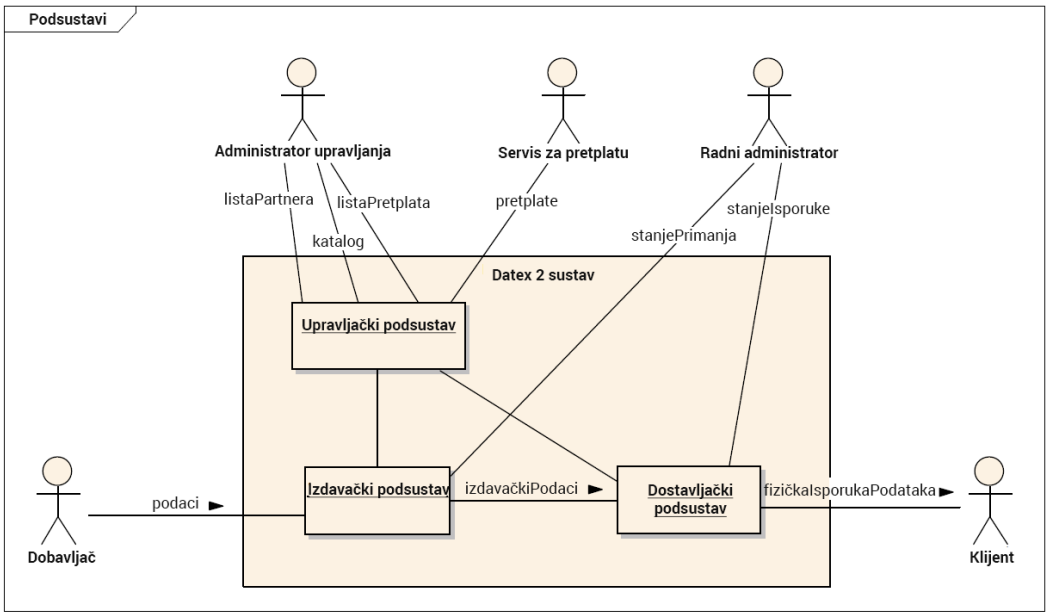
Prilikom dohvaćanja podataka putem mrežnog servisa izdavač periodički pruža podatke te ih dostavlja opskrbljivačkom podsustavu (engl. *Supplier subsystem*). Ovakav pristup je učinkovitiji od pružanja podataka svaki puta kada ih klijent zatraži. Dostavljački podsustav (engl. *Delivery subsystem*) je izrađen u skladu s Datex 2 „Pull WSDL“ specifikacijom. Postoje dvije razvojne cjeline (engl. *Framework)* koje podržavaju mrežni servis, a bazirani su na Apache foundation Axis 1.4.1 i Microsoft .Net 2.0. Prilikom razvijanja programskog dodatka za Resqmap sustav korišten je mrežni servis razvijen na Microsoft .NET razvojnoj cjelini. Međudjelovanje izdavačkog podsustava i mrežnog servisa se može izvesti na dva načina: mrežni servis je pretplaćen na izdavački podsustav i bez pretplate [7].

## 3.6. Arhitektura Datex 2 podsustava

Datex 2 sustav se sastoji od različitih publikacija koje se mogu isporučiti klijentima različitim načinima rada. Datex 2 omogućava svakom korisniku mogućnost definiranja profila prema vlastitim zahtjevima sve dok je omogućeno međusobno djelovanje zajedničkih dijelova (publikacije, načina rada) sa ostalim korisnicima. Logička struktura Datex 2 standarda podijeljena je u 4 paketa:

- Razmjena (engl. *Exchange*) sadrži dijagram razmjene i definicije prebrojavanja,
- Publikacija nosivosti (engl. *Payload Publication*) sadrži opise podataka koji se razmjenjuju,
- Općenito (engl. *General*) sadrži „općenite klase“; tipove podataka, klase lokacija, prebrojavanje nosivosti (engl. *Payload Enumeration*), klase za višekratnu upotrebu (engl. *Reusable Classes*),
- Upravljanje (engl. *Management*) sadrži upravljanje situacijama u načinima rada „Izdavač šalje podatke na događaj“ (engl. *Publisher Push on occurrence*),
- Nadogradnja (engl. *Extension*) je rezervirana za elemente koji sadrže moguće nadogradnje.

Arhitektura Datex 2 podsustava je opisana UML dijagramom koji je prikazan na slici 3.2. Između tri postjeća sustava, Datex 2 opisuje samo dostavljački podsustav što daje slobodu svakom programeru da razvija sučelja na klijentskoj strani.




**Slika 3.2.** Arhitektura Datex 2 podsustava prikazana UML dijagramom [7]

Datex 2 standard na dostavljačkoj strani sadrže sljedeće dijelove:

- izdavački podsustav koji razrađuje podatke koji se isporučuju,
- način dostavljanja podataka izdavačkom podsustavu,
- upravljački podsustav
    - sadrži popis partnera za razmjenu podataka (autorizacija klijenata),
    - upravlja katalogom (tipovi dostupnih publikacija, dostupne lokacije),
    - upravljanje pretplatama,
- administraciju za razmjenu.

## 3.7. Datex II logički model

Datex II logički model „*d2LogicalModel* " je osnovni model u kojem se nalaze svi podaci definirani Datex II standardom. Njegova struktura je definirana XML shemom te se sastoji od dvije komponente, a to su razmjena (engl. *Exchange*) i nosivost (engl. *PayloadPublication*). Klasa za razmjenu „*Exchange*" se koristi za razmjenu parametera koji su vezani uz dostavu podataka klijentu. „*Payload*" pod model se sastoji od različitih publikacija koje primarno sadrže podatke koji su vezani uz promet i putovanje. Neke od publikacije se također koriste za razmjenu statičkih tablica kao što su one korištene za referenciranje lokacija ili određivanje položaja opreme na prometnicama. Svi stvarni podaci i informacije vezane uz putovanje se nalaze unutar „*Payload*" pod modela.

Datex 2 standard obuhvaća 5 glavnih publikacija (objava):

- Publikacija koja sadrži situacije (engl. *Situation publication*),
- Publikacija koja sadrži razrađene podatke (engl. *Elaborated data publication*),
- Publikacija koja sadrži izmjerene podatke (engl. *Measured data publication*),
- Publikacija koja sadrži informacije o stanju u prometu (engl. *Traffic View publication*),
- VMS publikacija (engl. *VMS Publication*).

Postoje 3 dodatne publikacije koje definiraju reference na postojeće publikacije podržane Datex 2 standardom. Svaka publikacija mora sadržavati:

- vrijeme stvaranja publikacije (engl. *Publication Time*),
- jezik koji će se koristiti,

- tvorac publikacije koji ima međunarnodni identifikator koji se sastoji od dva slova za enumeraciju zemlje plus identifikator za svaku državu. Tvorac može biti drugačiji od opskrbljivača publikacije (engl. *Publication Supplier*) [7].

## 3.8. Publikacija koja sadrži situacije

Glavne značajke publikacije su mogućnost grupiranja nekoliko situacija. Svaka situacija je definirana kao identificirajuća klasa *traffic/travel* situacije koja se sastoji od jedne ili više okolnosti koje su povezane jednom ili više veza. Svaka traffic/travel okolnost je zastupljena u zapisu situacija (engl. *Situation Record*). Svojstva (engl. *Attributes*) publikacije mogu se isporučiti u *InformationHeader*-u koji daje pojedinosti o upravljanju informacijama klijentu kao što su područje interesa (engl. *Area of interest*), povjerljivost (engl. *Confidentiality*), hitnost (engl. *Urgency*), status informacije (engl. *Information status*). Situacija može biti sastavljena od nekoliko zapisa.

### 3.8.1. Zapisi situacija

Zapis situacije (engl. *Situation Record*) se definira kao identificirajuća klasa jednog zapisa odnosno elementa unutar situacije. Ova klasa je apstraktnog tipa što znači da se ne može instancirati direktno. Svaki zapis situacije ima jedinstveni verzonizirani identifikator. Prvi dio identifikatora je ostvaren onog trenutka kada je zapis situacije prvi puta stvoren unutar Datex II baze podataka. Drugi dio identifikatora čini njegova verzija. Vrijednost zapisa situacije sačinjavaju slijedeće komponente:

- status dostupnosti koji definira trenutno stanje zapisa koje može biti aktivno ili suspendirano. Status se može definirati prema različitim vremenskim periodima,
- jedno obavezno (engl. *Mandatory*) globalno definirano prvo početno vrijeme „overallStartTime“,
- jedno neobavezno (engl. *Optional*) stvarno vrijeme završetka „overallEndTime“,
- jedan ili više perioda provjere gdje svaki ima početno i krajnje vrijeme (datum) za koje je zapis situacije aktivan odnosno važeći. Mogućnost detaljnijeg definiranja vremena pomoću sati/dana/tjedana/mjeseci,
- jedan ili više perioda iznimaka gdje svaka ima početno i krajnje vrijeme (datum) za koje je zapis situacije aktivan odnosno važeći. Mogućnost detaljnijeg definiranja vremena pomoću sati/dana/tjedana/mjeseci,

- neobavezno „prepisivanje“ (engl. *Overrunning*) označava aktivnost odnosno akciju koja još uvijek napreduje u izvođenju prepisujući njeno planirano trajanje koje je označeno prijašnjom verzijom zapisa [7].

Postoje 3 glavne kategorije zapisa situacije:

- Prometni element (engl. *Traffic Element*) je neplanirani događaj od strane prometnog operatera koji utječe ili ima mogućnost utjecaja na protočnost u prometu.
- Akcije operatera (engl. *Operator action*) su akcije koje stvaraju prometni operateri kako bi spriječili ili pomogli u rješavanju opasnih i loših uvijeta u prometu, uključujući održavanje cestovne infrastrukture.
- Informacije vezane uz ne cestovne događaje (engl. *Non-road event information*) su informacije o događaju koji nije vezan uz cestu već može utjecati na ponašanje vozača što ima izravan utjecaj na prometnu protočnost [7].

### 3.8.2. Prometni elementi

Klasa kojom su definirani prometni elementi je apstraktna. Postoje 6 vrsta prometnih elemenata definirani Datex II standardom:

- prepreke (engl. *Obstruction*),
- nenormalan promet (engl. *Abnormal traffic*),
- nesreće (engl. *Accident*),
- kvarovi sustava ili opreme (engl. *Equipment or system fault*),
- aktivnosti (engl. *Activities*),
- uvjeti (engl. *Conditions*)

Prepreke podrazumjevaju sve pokretne i nepokretne objekte koji su podjeljene u 5 kategorija:

- prisutnost životinja (mogu biti žive ili mrtve),
- prirodne prepreke,
- prepreke uzrokovane oštećenjima infrastrukture,
- općenite prepreke,
- prepreke koje uzrokuju vozila

Svaki tip prepreka sadrži brojčano izraženu vrijednost koliko se prepreka nalazi na cesti (npr. dva drveta blokiraju prometnicu). Nenormalan promet označava stanje koje je neuobičajeno za promet

te obuhvaća nekoliko atributa kao što je tip, broj vozila u koloni, dužina kolone, relativna protočnost, itd [7].

### 3.8.3. Akcije operatera

Datex II standardom su definirana 4 tipa akcija koje mogu poduzeti operateri:

- radovi na cestama (engl. *Roadworks*),
- postavljanje znakova (engl. *Sign setting*),
- upravljanje prometnom mrežom (engl. *Network management*),
- pomoć na cesti (engl. *Roadside assistance*).

Svaka situacija objavljena u publikaciji sadrži sva nabrojana svojstva i opise za svaku lokaciju na kojoj se prikupljaju podaci. Primjerice kada se klijentu pošalje objekt napravljen prema klasi „Situation“ on će sadržavati informacije o svakom zapisu i njegovim atributima [7].

## 3.9. Publikacija koja sadrži razrađene podatke

Svrha ove publikacije je pružanje informacija korisniku o podacima koji su na neki način obrađeni ili izvedeni. Uglavnom su to podaci koji su obrađeni od strane prometnih centara te se čuvaju u njihovim bazama podataka. Publikacija sadrži jedan ili više setova obrađenih podataka. Putem publikacije klijent dobiva set zadanih (engl. *Default*) vrijednosti koje uključuju prognozu (engl. *Forecast*), period i vrijeme. Ovakav pristup omogućava smanjenje veličine publikacije kada se radi o velikom broju vrijednosti koje dijele zajedničke parametre. Kako bi se dodatno smanjio obujam potrebnih vrijednosti koje se šalju klijentu koristi se *ReferenceSettings*. Klasu čine reference na predefinirani set neodređenih lokacija i zadane vrijednosti koje klijenti mogu pretpostaviti za sve lokacije ako one nisu dostupne. Na taj način dobavljač (engl. *Supplier*) treba poslati vrijednosti samo za one lokacije čije se vrijednosti razlikuju od zadane vrijednosti. Svaki razrađeni podatak (engl. *Elaborated Data*) se sastoji od vrijednosti osnovnih podataka (engl. *Basic Data*). Vrijednosti osnovnih podataka se dobivaju iz osnovne apstraktne klase koja ima slijedeće atribute:

- vremenska preciznost (preciznost mjerenja ili računanja vremena),
- period (u sekundama),
- vrijeme (vrijednost osnovnog podataka kada je izmjeren ili razrađen).

Svaka vrijednost osnovnog podatka ima pripadajuću lokaciju (engl. *Pertinent Location*). Pripadajući lokaciju čini grupa lokacija koje se identificiraju putem referenci. Vrijednosti osnovnih podataka mogu biti jedan od sljedećih tipova:

- vrijeme putovanja (engl. *Travel time data*),
- prometni status (engl. *Traffic status*),
- prometni podaci (engl. *Traffic data*),
- informacije o vremenskim prilikama (engl. *Weather data*) [7].

## 3.10. Publikacija koja sadrži izmjerene podatke

Publikacija sadrži mjerenja sa jedne ili više mjernih lokacija koje su definirane u publikaciji koja sadrži predefinirane lokacije. Mjerenja se vrše putem različitih uređaja kao što su brojila prometa, uređaji za mjerenje brzine i sl. Izmjereni podaci iz publikacije moraju sadržavati referencu na tablicu sa mjernim lokacijama (engl. *Measurement site table*). Svaki set podataka sa vrijednostima mjerenja moraju imati referencu na lokacije mjerenja i zadano vrijeme mjerenja (engl. *Measurement Time Default*) osim ako nije drugačije zadano. Set podataka o mjernim lokacijama ima poredanu (indeksiranu) listu mjerenih vrijednosti čiji redoslijed odgovara indeksima u tablici koja sadrži lokacije mjerenja. Ovakav pristup ne zahtjeva dodatnu identifikaciju svake mjerene vrijednosti na pojedinoj lokaciji. Vrijednost mjerenja (objekt klase) može odrediti tip opreme koja se koristila za dobivanje vrijednosti [7].

## 3.11. VMS publikacija

Publikacija sadrži informacije o trenutnom stanju i postavkama jedne ili više promjenjivih znakovnih poruka (engl. *Variable Message Signs*, VMS). Promjenjive znakovne poruke se sastoje od:

- osnovne stavke (ako VMS radi) i opisu intervala slijednih poruka u sekundama (ako je primjenjivo),
- postavke za prikaz područja za tekst (engl. *Text Display Area Settings*),
- poredani set postavki za prikaz područja za piktogram (engl. *Pictogram Display Area Settings*), jedna postavka za svako područje za prikaz (engl. *Display Area*). Svako područje za prikaz piktograma VMSa je referencirano pomoću „*pictogramDisplayAreaIndex*“ uz detalje o relativnoj poziciji u odnosu na tekst ili o apsolutnoj poziciji koja se zadaje pomoću „*VmsPictogramDisplayCharacteristics*“.

- Poredani set VMS poruka. Indeks poruka pruža mogućnost organiziranja sekvence prikazanih prouka. VMS prikazuje svaku poruku u ovisnosti o predanom indeksu svake poruke.

U slučaju da VMS nije prethodno definiran u publikaciji VMS tablica, potrebno je dostaviti statičke karakteristike o:

- njenoj lokaciji u pogledu pozicije (*vmsLocation* – definirana kao točka) i u pogledu upravljive logične lokacije kao što je prometna dionica (definirana kao grupa lokacija) gdje je VMS primnjenjiv.
- Njegove fizičke karakteristike kako o tekstu tako i o piktogramu. Kako VMS može biti nekoliko područja koje sadrže piktograme, svako od njih je referencirano pomoću „*pictogramDisplayAreaIndex*“ [7].

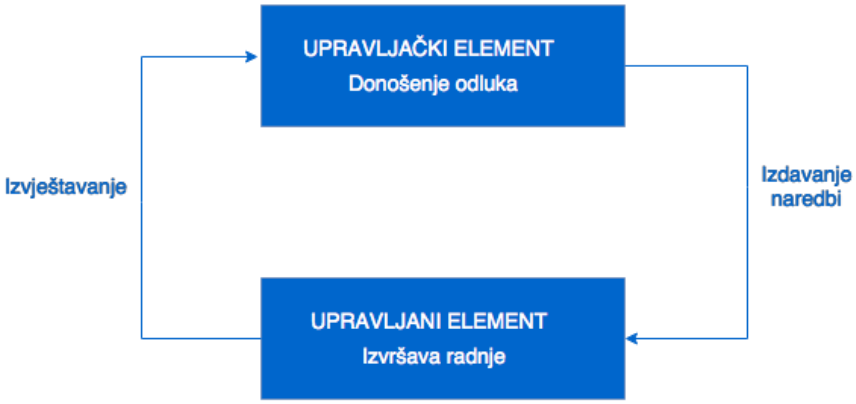
Svaku VMS poruku čini nekoliko elemenata. Osnovne informacije uključuju:

- tip informacije koje VMS poruka sadrži (upozorenje, vrijeme putovanja, upravljanje prometom, datum i vrijeme, temperaturu, upute, buduće informacije),
- vrijeme (kada je poruka zadnji puta postavljena),
- kodirani razlog postavljanja (situacija, kampanja, upravljanje prometom, vrijeme putovanja) i odgovarajući način upravljanja ili plan diverzije,
- vlast, organizacija ili sustav koji zahtjeva postavljanje poruka,
- reference prema odgovarajućim situacijama i zapisima situacija (engl. *Situation records*),
- udaljenost VMS-a od lokacije odgovarajućeg zapisa ili elementa.

Publikacija koja sadrži VMS tablice sadrži jednu ili više VMS jedinica (engl. *VMS unit*) od koje se svaka sastoji od seta zapisa koji sadržavaju detalje o toj jedinici. VMS tablica i karakteristike odgovarajućih VMS jedinica su referencirane VMS publikacijom što omogućuje efikasno slanje velikog broja prikazanih poruka [7].

# 4. SUSTAVI ZA NADZOR I UPRAVLJANJE

Sustavi za upravljanje i nadzor (engl. *Command & Control Systems*) su relativno novo polje za istraživanje i primjenu iako nam je koncept upravljanja i nadzora poznat od prije. Nakon desetljeća upotrebe u kritičnim situacijama, vladinim organizacijama i vojsci, mogućnosti upravljanja i nadzora nalaze primjenu u naprednim „stroj za stroj“ (engl. *Machine-to-machine M&M*) komunikacijama i njihovoj primjeni u javnim djelatnostima, istraživanjima i komercijalnom tržištu. Takvi sustavi pružaju optimalne upravljačke operacije koje odgovaraju traženom kontekstu kako bi olakšali proces upravljanja korisnicima. Upravljanje i nadzor mora sadržavati akciju upravljanja i upravljani element kao što je prikazano na slici 4.1. Osnovna funkcija upravljačkog elementa je odlučivanje (engl. *Decision making*) zasnovano na procjeni upravljanog elementa koji izvržava naredbe upravljačkog elementa [9].



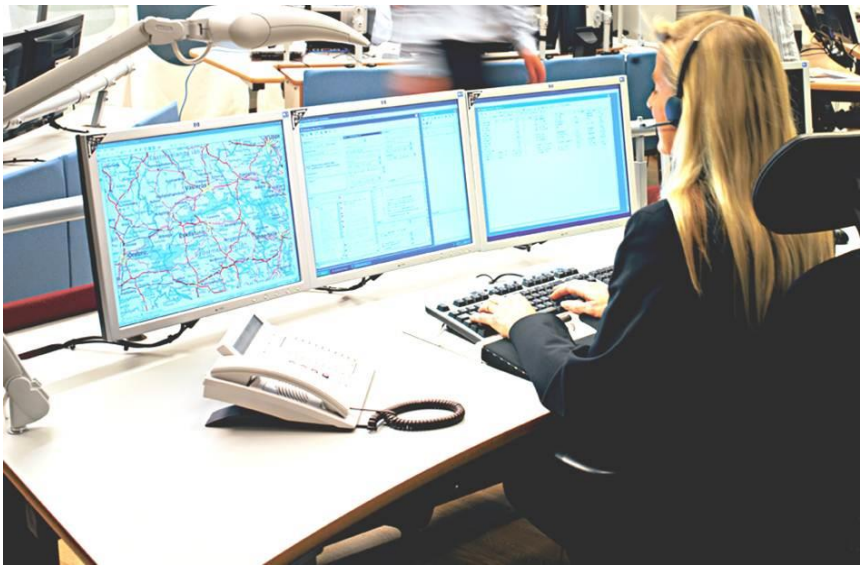

**Slika 4.1.** Primjena sustava za upravljanje i nadzor u zatvorenoj petlji

Odlučivanje je jedan od važnijih zadataka sustava za upravljanje i nadzor te se može implementirati ako imamo jasnu predodžbu procesa. Jasna predodžba se zasniva na ažuriranim informacijama i podacima unutar akcija dok upravljanje mora biti sposobno odrađivati akcije u stvarnom vremenu. Sustave za upravljanje i nadzor karakterizira koordinacija resursa koji se nalaze raspršeni na geografskom području. Ona uključuje prikupljanje podataka, korelaciju između podataka, prikaz informacija, odlučivanje, slanje naredbi i njihova implementacija [9].

## 4.1. CoordCom

CoordCom [4], [5] je sustav za upravljanje i nadzor koji omogućava efikasniju povezanost službi spašavanja, analizu i slanje hitnih poziva korisnika koji trebaju pomoć policije, vatrogasaca ili hitne pomoći odnosno sustava 112. CoordCom je vrlo skalabilno i fleksiblino „*end-to-end*“ rješenje. Sustav upravlja i koordinira cijeli lanac žurnih aktivnosti od primanja i identifikacije

žurnog poziva do slanja odgovarajućih resursa kao što je policija, vatrogasna služba i hitna pomoć. CoordCom integrira funkcije za telefoniju, radio prijem kao i razmjenu podataka između osobe koje je prijavila incident, operatera koji radi na CoordComu i službi spašavanja. Sustav omogućava analizu napora žurnih sužbi kako bi omogućio unapređenje u njihovom poslovnom procesu te kako bi primjenili odgovarajući pristup različitim situacijama. Mogućnost razmjene podataka i informacija sa drugim vanjskim sustavima kao što je ResQMap sustav je velika prednost ovakvog sustava za nadzor i upravljanje. Najčešća postava CoordCom sustava i operatora je pomoću tri monitora kao što je prikazano slikom 4.2. Na dva monitora se prikazuje CoordCom sustav sa trenutno otvorenim slučajevima i njihovim popisom. Drugi monitor se koristi za prikaz geografskog područja pomoću ResQmap sustava.



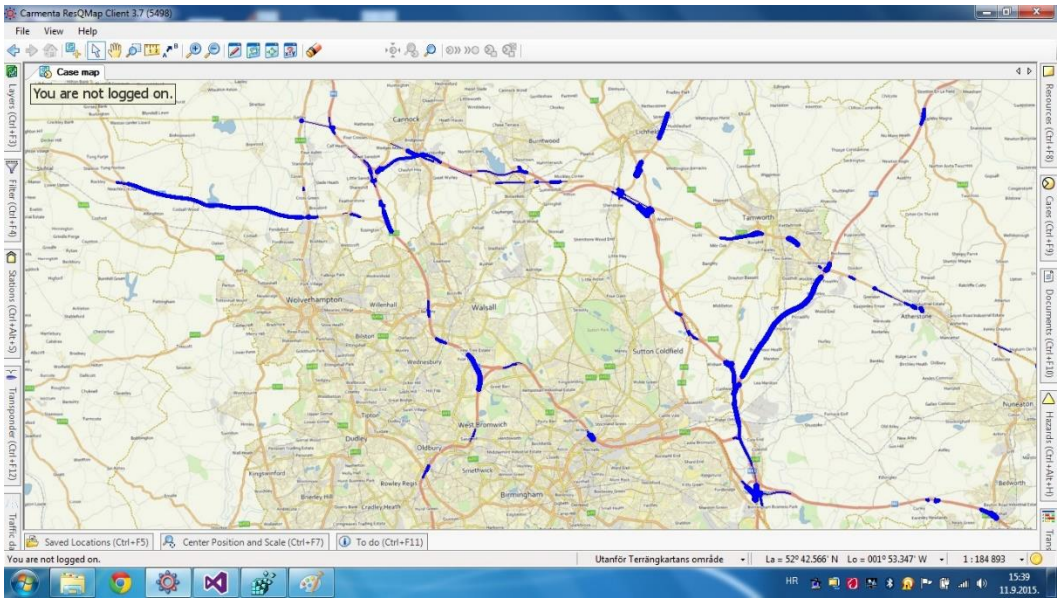
**Slika 4.2.** Postava CoordCom sustava na tri monitora s operaterom

Izvor: http://archive.ericsson.net/service/internet/picov/get?DocNo=1/28701-FGD101032&Lang=EN&HighestFree=Y

## 4.2. ResQmap

ResQmap [4] je sustav za prikazivanje geoprostornih karti za veliki broj korisnika te je dizajniran kako bi ispunio rastuće potrebe službi za spašavanje. Karakteristična primjena ResQmap sustava je u centrima za nadzor i upravljanje kao potpora prihvaćanju žurnih poziva te slanje ambulantnih vozila na mjesto nesreće ili integracija sa video i senzorskim sustavima. ResQmap podržava sve tipove GIS podataka te je prikaz (prezentacija) podataka jasna, sažeta i točna. Sustav može koristiti sve kombinacije rasterske slike i vektorske podatke, te jednostavno upravlja s podacima u različitim koordinatnim sustavima. ResQmap sustav je dizajniran kao

otvorena i fleksibilna arhitektura kako bi se mogao jednostavno integrirati s drugim sustavima kao što su dispečerski sustavi, vanjske baze i sl. Komunikacijska sučelja koriste XML standard za razmjenu podataka. Prikaz karti sadrži različite slojeve informacija kao što je pozadina, ulice, hidranti i objekti nacrtani od strane korisnika. Sva vozila opremljena GPS uređajima i uređajima za komunikaciju (pametni telefoni) su ucrtani na kartama. ResQmap je povezan sa CoordCom sustavom te može grafički prikazati informacije prikupljene sa različitih prometnih izvora kao što su senzori uključujući i lokacije poziva kao što je prikazano snimkom ekrana (engl. *Screenshot*) na slici 4.3. Digitalne karte sa naprednim mogućnstima mogu brzo locirati mjesto nesreće. ResQmap pruža mogućnost vizualnog praćenja slučaja kako bi dobili više podataka o samom slučaju.




**Slika 4.3.** Prikaz digitalnih karti na vizualnom sučelju

# 5. TEHNIČKI OPIS PROJEKTA

Kao što smo spomenuli prije u radu Datex 2 je standard za razmjenu podataka između upravljačkih prometnih centara i pružatelja usluga. U radu je korišten NTIS (*England National Traffic Information System*) pružatelj usluga koji implementira u potpunosti Datex 2 standard u svim izvorima iz kojih dohvaća podatke te ih distribuira klijentima. Izvori uključuju nekoliko upravljačkih prometnih centara i njihove prometne senzore koji zajedno čine jedinstevni servis za distribuciju prometnih informacija. NTIS je dobar primjer kako bi se Datex 2 trebao primjeniti i integrirati svoje podatkovne izvore sa C&C platformom (engl. *Command and Control*). Na ovaj način će se u budućnosti povezati svi državni i regionalni prometni podaci i njihovi izvori jer će koristiti isti standard.

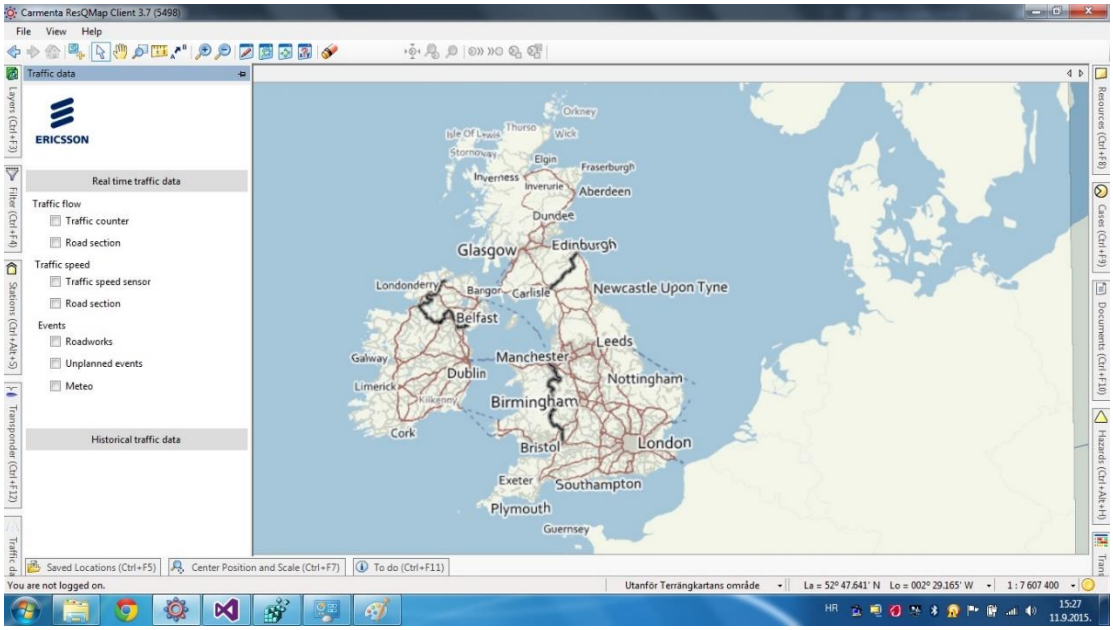
Postojeći sustavi za upravljanje prometom ne koriste standardizirane prometne podatke sa izvora u stvarnom vremenu već se oslanjanju na različite nestandardizirane izvore. Drugim riječima, sustav napravljen u jednoj državi se ne može primijeniti na druge zemlje. U Hrvatskoj postoji najmanje 10 dionika (engl. *Stakeholder*) koji pružaju podatke vezane uz promet te ne postoji jedan centar koji bi prikupio sve podatke i distribuirao ih prema klijentima. Idjea iza diplomskog rada je proširiti trenutne sustave za upravljanje i nadzor kako bi mogli raditi sa standardiziranim Datex 2 podacima koji će dodati nove mogućnosti za postojeće sustave.

Cilj diplomskog rada je izraditi dodatak za ResQmap GIS sustav koji će vizualizirati standardne prometne podatke i integrirati ih sa platformom za nadzor i upravljanje CoordCom koji će na osnovu podataka okidati alarme za različite anomalije u podacima. Rezultat ovog rada je povećanje upravljanja na prometnicama kao i brže odzivno vrijeme za različite akcije. Ovaj sustav je primjenjiv u svim zemljama Europske Unije koje koriste isti standard.

## 5.1. Ideja

Uključuje izradu pretplatničkog servisa (engl. *Subscriber Service*) koji je pretplaćen na NTIS sustav i dodatka na ResQmap sustavu. Pretplatnički sustav dohvaća Datex 2 podatake sa NTISa te ih sprema u MS SQL bazu podataka koja je povezaa sa svakim klijentom. Za potrebe testiranja razvijen je lokalni način rada koji omogućava da se podaci sa NTIS sustava prenesu lokalno te učitaju u sustav. Podaci koje NTIS sustav pruža spremljeni su u XML dokumentu po Datex 2 standardu. Razvijeni dodatak ima mogućnost raščlanjivanja (engl. *Parse*) sadržaja iz XML dokumenta te ih vizualizira pomoću slojeva na ResQmap klijentu. Dodatak omogućava izbor tipa informacija koji se želi prikazati na ResQmap klijentu. Postoji dva tipa informacija za prikaz

prometne brzine i protoka. Prvi tip podataka je točka (engl. *Point*) koji sadrži vrijednosti sa senzora na točnoj lokaciji dok drugi tip podataka uključuje liniju (engl. *Line*) te sadrži vrijednosti za određenu dionicu ceste. Vremenski podaci su prikazani pomoću mnogokutnih objekata (engl. *Polygonal Object*) kao što je prikazano snimkom ekrana na slici 5.1.




**Slika 5.1.** Razvijeni dodatak integriran sa ResQmap sučeljem

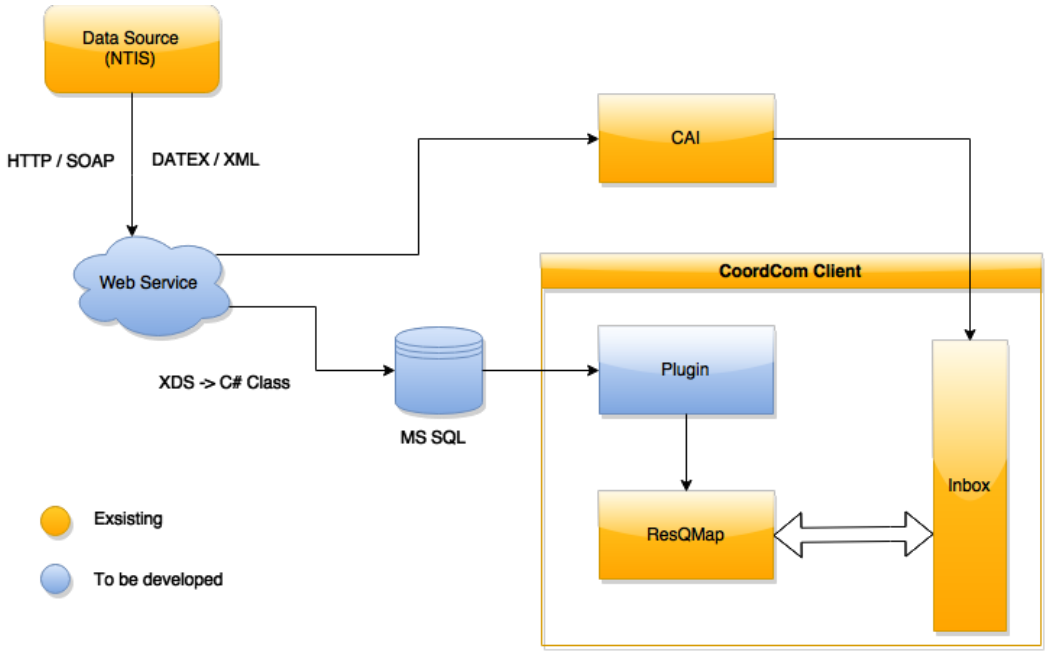
Dodatak pruža korisnicima usporedbu između povijesnih i trenutnih podataka. Postoje 3 tipa događaja koji se mogu prikazati na ResQmap klijentu a to su:

- radovi na cesti koji su planirani događaji koji sadrže informacije o tome koliko je prometnih traka zahvaćeno radovima, koliko će radovi potrajati, itd.
- Neplanirani događaji kao što su prometne nesreće koje se događaju na prometnicama
- Meteorološki podaci koji sadrže mnogokutne objekte koji predstavljaju zahvaćeno područje uz opis vremenskih prilika odnosno neprilika.

## 5.2. Pregled projekta

Projekt se sastoji od nekoliko dijelova koji čine izvor podataka (NTIS sustav) koji prikuplja sirove podatke sa senzora te ih strukturira i oblikuje prema Datex 2 standardu. NTIS šalje podatke putem SOAP protokola prema pretplatničkom servisu koji je mrežni servis. Mrežni servis dohvaća i procesira pristigle podatke te ih sprema u MS SQL bazu podataka koja je povezana na svakog CoordCom i ResQmap klijenta kao što je prikazano na slici 5.2. Unutar ResQmap GIS klijenta nalazi se razvijeni dodatak koji dohvaća podatke sa mrežnog servisa te ih vizualizira na

korisničkom sučelju. Ako sustav otkrije anomalije u podacima tada CoordCom može okinuti alarm pomoću CAI sučelja (engl. *CoordCom Alarm Interface*) koji će stvoriti slučaj sa posebnim setom rutina za određenu anomaliju ili događaj [5].



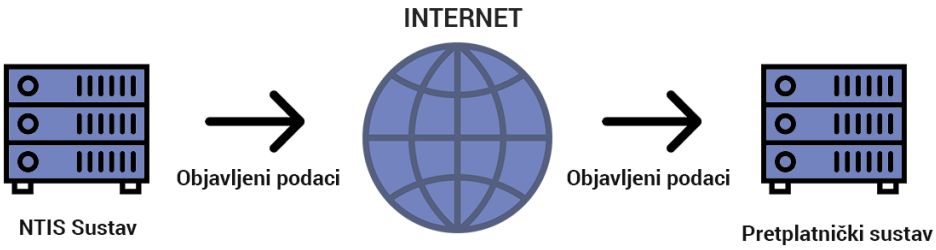

**Slika 5.2.** Arhitektura projekta

Dodatak ima mogućnost raščlanjivanja podataka koji pristižu izravno sa mrežnog servisa ili lokalno učitane XML dokumente. Drugim riječima dodatak ima mogućnost mrežnog i izvanmrežnog rada.

## 5.3. Implementacija mrežnog servisa

NTIS sustav šalje podatke svim pretplatničkim sustavima na odabranom načinu rada. Podaci koji se dobiju od strane izdavačkog sustava se filtriraju i prilagođavaju u ovisnosti o opcijama i postavkama koje su definirane pretplatničkim sustavom. Kako be se primili objavljeni podaci, pretplatnički sustav mora implementirati serversku krajnju točku (engl. *Server-side endpoint*) mrežnog servisa. Mrežni servis je definiran uz pomoć Datex 2 Push WSDL standarda (engl. *Web Service Description Language*). WSDL definira sučelje i metodu „*putDatex2Data*“ na strani mrežnog servisa [10]. Metoda omogućava dohvaćanje podataka sa NTIS sustava na način da šalje poruku koja sadrži informacije o tipu podataka koje treba poslati izdavački sustav. Tijek rada NTIS sustava i pretplatničkog sustav je prikazan slikom 5.3. Pretplatnički sustav koji je implementiran pomoću mrežnog servisa dohvaća podatke od izdavačkog sustava (NTIS). Ovakav centralizirani

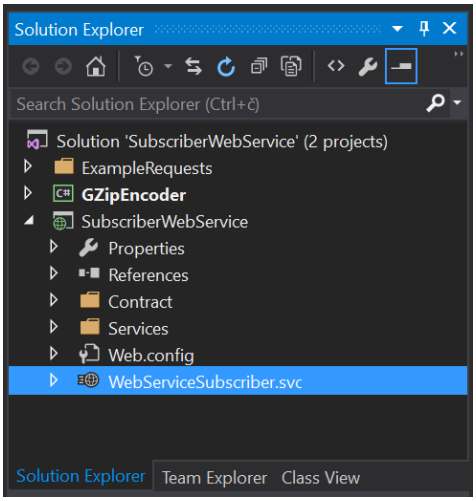
pristup omogućava jednostavnu distribuciju podataka do svakog klijenta koji je spojen preko pretplatničkog sustava.




**Slika 5.3.** Tijek rada od NTIS sustava prema pretplatničkom sustavu

Svaka promjena u podacima je proslijeđena svakom klijentu kako bi svi dobili ažurirane podatke u isto vrijeme. Državni servis za prometne informacije (engl. *National Traffic Information Service*, skraćeno NTIS) je razvio pretplatnički servis (engl. *Subscriber Service*) za Java i .NET platformu [10]. Pretplatnički servis korišten za potrebe implementacije dodatka u diplomskom radu je razvijen na .NET platformi koristeći Visual Studio 2015 razvojno okruženje i .NET radni okvir (engl. *Framework*) pod verzijom 4. Prema zadanim postavkama mrežni servis (engl. *Web Service*) može primiti 512 MB sažetih podataka (engl. *Compressed data*). Unutar mrežnog servisa se nalazi automatski stvorena datoteka pod nazivom „*supplierPush.cs*“ koja se stvara pomoću WSDL (engl. *Web Services Description Language*) i XSD (engl. *XML Schema Definition*) datoteka. Naredba koja se koristila pri izgradnji „*supplierPush.cs*“ datoteke se može pronaći odabiranjem „*SubscriberWebService*“ svojstva pod karticom „*Build Events*“ unutar programskog rješenja.

Pretplatnički mrežni servis se sastoji od ugovora (engl. *Contract*), implementacije sučelja mrežnog servisa, XML datoteke za konfiguraciju mrežnog servisa, programskog koda koji stvara mrežni servis kao što je prikazano projektnim stablom na slici 5.4.




**Slika 5.4.** Projektno stablo pretplatničkog mrežnog servisa

Unutar datoteke nalaze se WSDL i WSD datoteke pomoću kojih pretplatnički mrežni sustav izgrađuje sučelje na klijentskoj strani kako bi se komunikacija mogla nesmetano odvijati. Osim navedenih datoteka postoji još „*supplierPush.cs*“ datoteka koja sadrži programski kod koji definira način na koji se podaci šalju prema klijentima. Unutar datoteke servisi se nalaze implementacije sučelja prema raznim publikacijama koje dolaze od strane dostavljačkog sustava odnosno izvora podataka.

### 5.3.1. Prilagođavanje podataka .NET okruženju

Mrežni pretplatnički servis od dobavljačkog sustava dobiva podatke u XML formatu strukturirane prema Datex II standardnu. Kako su mrežni servis i programski dodatak na klijentskoj strani izrađeni na .NET platformi potrebno je dobivene podatke prilagoditi okruženju u kojem se navedeni programi napisani. Jednom kada podaci stignu od izvora podataka (u ovom slučaju NTIS) putem SOAP protokola potrebno ih je pretvoriti iz XSD sheme u C# klase na osnovu kojih ćemo praviti objekte s kojima će se vršiti upravljanje i obrada kao što je prethodno prikazano slikom 5.2. Pretvaranje se može napraviti pomoću alata „*XML Schema Definition Tool*“ koji iz XSD datoteka stvara C# klase. Alat je dostupan unutar radnog okvira .NET platforme. Alat se koristi pomoću konzole u koju je potrebno unijeti naredbe i putanju kao što je prikazano programskim kodom 5.1.

```
xsd file.xsd {/classes | /dataset} [/element:element]
[/language:language] [/namespace:namespace]
[/outputdir:directory] [URI:uri]
```

**Programski kod 5.1.** Prikazuje naredbu za pretvorbu XSD datoteke u C# datoteku

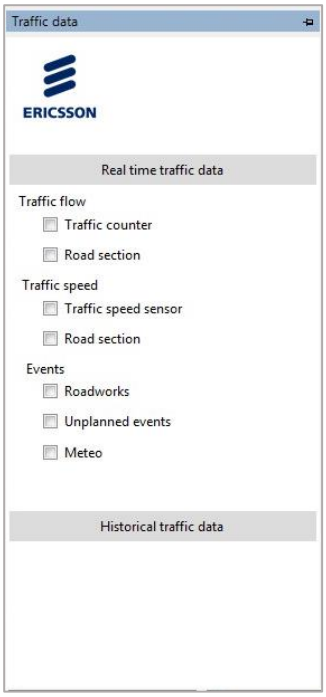
Naredbi „*xsd*“ potrebno je u argument dati XSD datoteku iz koje želimo stvoriti C# datoteku „*file.xsd* “. Rezultat pretvaranja je C# datoteka kao što je prikazano programskim kodom 5.2. Ako želimo da se rezultat spremi na drugu lokaciju to je potrebno navesti (npr. *xsd myFile.xml /outputdir:myOutputDir*). Kako bi se omogućilo upravljanje podacima svih publikacija potrebno ih je pretvoriti u odgovarajući oblik (C# klase). Iz dobivenih C# klasa vrlo lako se uz pomoću „*Entity frameworka*“ mogu stvoriti Microsoft SQL baze podataka.

```
    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.Xml", "4.6.81.0")]
    [System.SerializableAttribute()]
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true, Namespace =
"http://datex2.eu/schema/2/2_0")]
    [System.Xml.Serialization.XmlRootAttribute(Namespace = "http://datex2.eu/schema/2/2_0",
IsNullable = false)]
    public partial class d2LogicalModel
    {
        private d2LogicalModelExchange _exchange;

        private d2LogicalModelPayloadPublication _payloadPublication;

        private byte _modelBaseVersion;

        public d2LogicalModel()
        {
            this._payloadPublication = new d2LogicalModelPayloadPublication();
            this._exchange = new d2LogicalModelExchange();
        }
        //get I set metode konstruktora
        public d2LogicalModelExchange exchange{…}
        //get I set metode konstruktora
        public d2LogicalModelPayloadPublication payloadPublication{…}
        //get I set metode konstruktora
        [System.Xml.Serialization.XmlAttributeAttribute()]
        public byte modelBaseVersion{…}
    }
```

**Programski kod 5.2.** Rezultat pretvorbe XSD datoteke u C# datoteku

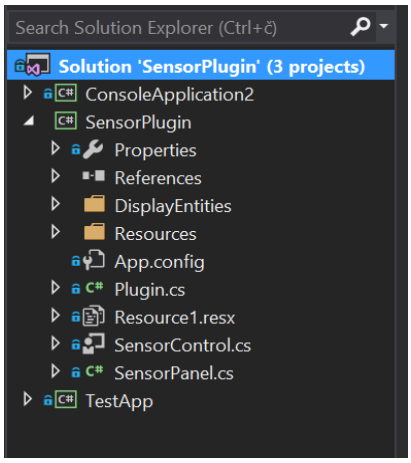
## 5.4. Programski dodatak

Programski dodatak je integriran u ResQmap sustav te omogućava rasčlanjivanje Datex 2 podataka iz XML dokumenta. Obrađeni podaci se vizualiziraju na korisničkom sučelju ResQmap sustava u nekoliko slojeva. Nekoliko dijelova čine programski dodatak. Osnovna klasifikacija dodatka je korisničko sučelje koje je prikazano na slici 5.5 i programski kod u pozadini.



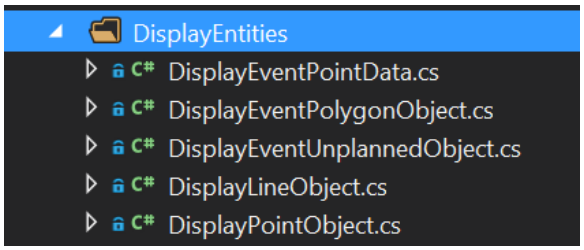

**Slika 5.5.** Korisničko sučelje programskog dodatka

U projektnom stablu na slici 5.6. vidljivi su svi dijelovi programa i datoteke koje čine programski dodatak. Datoteka *SensorControl.cs* sadrži definicije elemenata sučelja i biblioteke za iscrtavanje tih elemenata. *Resource1.resx* je datoteka koja sadrži ikonu programskog dodatka koja se učitava prilikom pokretanja ResQmap programa. Datoteka *Plugin.cs* registrira programski dodatak u ResQmap sustav. Unutar *SensorPanel.cs* datoteke se nalaze funkcije koje omogućavaju rasčlanjivanje, procesiranje i vizualizaciju podataka. Dodatne biblioteke za implementaciju određenih funkcija se nalaze u referencama.




**Slika 5.6.** Projektno stablo programskog dodatka

Kako bi uspješno testirali lokalne podatke napravljeni su modeli po Datex 2 shemi koji se nalaze unutar mape *DisplayEntities*. Mapa sadrži klase koje implementiraju tipove informacije koje dohvaćamo putem NTIS sustava kao što je prikazano slikom 5.7.




**Slika 5.7.** Klase koje implementiraju tipove podataka

Programski dodatak podržava implementaciju podataka po Datex 2 standardu koji se nalaze u *d2LogicalModel* datoteci. Ulazna točka programskog dodatka je metoda *OnAttached* koja se pokrene prilikom inicijalizacije ResQmap programa. Unutar *OnAttached* metode dodaju se slojevi koji će vizualizirati podatke na GIS sustavu. Svaki tip podataka se prikazuje pomoću posebnog sloja koji se aktivira kroz korisničko sučelje programskog dodatka. Podaci su spremljeni u memorijske setove koji sadrže funkcije za procesiranje i obradu podataka dobivenih iz XML datoteka. Nakon što se svaki set podataka dodjeli odgovarajućem sloju tada se podaci spremaju u objekte radi lakše obrade kao što je prikazano programskim kodom 5.3.

```
public override void OnAttached(ResQMapModel resQMapModel)
        {
            // Remember the ResQMapModel for use later on
            this.resQMapModel = resQMapModel;

            this.dataSetFlowPoint = AddFlowPointDataLayer();
            this.dataSetFlowPointHistorical = AddFlowPointHistoricalDataLayer();
            this.dataSetFlowLine = AddFlowLineDataLayer();
            this.dataSetSpeedPoint = AddSpeedPointDataLayer();
            this.dataSetSpeedPointHistorical = AddSpeedPointHistoricalDataLayer();
            this.dataSetSpeedLine = AddSpeedLineDataLayer();
            this.dataSetEventRoadworks = AddEventRoadworksDataLayer();
            this.dataSetEventUnplanned = AddEventUnplannedDataLayer();
            this.dataSetEventPolygon = AddEventPolygonDataLayer();

            FillPointObjectList();
            FillHistoricalPointObjectList();
            FillLineObjectList();
            FillEventRoadWorksObjectList();
            FillEventUnplannedObjectList();
            FillEventWeatherObjectList();
        }
```

**Programski kod 5.3.** *OnAttached* metoda programskog dodatka

Za potrebe testiranja učitavanja podataka iz XML dokumenata napravljane su funkcije koje stvaraju i popunjavaju liste objekata. S obzirom da je riječ o prototipu aplikacije funkcije su

učitavale dokumente lokalno te se nisu prikazivali svi podaci koji su podržani Datex II standardom već samo oni koji se odnose na brzinu, protočnost, događaje i vremenske prilike na prometnicama. Funkcije koje učitavaju i obrađuju vrijednosti iz XML dokumenata su:

- *FillPointObjectList()* – učitava XML dokument sa MIDAS (engl. *Motorway Incident Detection and Automatic Signalling*) podacima
- *FillHistoricalPointObjectList()* – učitava XML dokument sa MIDAS podacima
- *FillLineObjectList()* – učitava XML dokument koji sadrži spojene i senzorske podatke
- *FillEventRoadWorksObjectList()* – učitava XML dokument koji sadrži podatke sa događajima na prometnicama
- *FillEventUnplannedObjectList()* – učitava XML dokument koji sadrže podatke sa događajima na prometnicama
- *FillEventWeatherObjectList()* – učitava XML dokument koji sadrže podatke sa događajima na prometnicama

Učitavanje podataka iz dokumenata se radi na isti princip u svakoj metodi. Potrebno je stvoriti objekt u koji će se spremiti sve vrijednosti iz XML dokuemnta. Učitavanje se vrši funkcijom *Load()* pomoću *XDocument* objekta koji predstavlja XML dokument. Prema Datex II strukturi XML dokument (*d2LogicalModel*) se sastoji od dva dijela, a to su razmjena i teret (engl. *Payload*). Podaci koji se prenose su spremljeni u „*PayloadPublication*“ dijelu koji se odvaja iz dokumenta posebnom varijablom tipa *XDocument*. Princip izdvajanja podataka iz XML dokumenta se svodi na traženje oznake (engl. *Tag*) koja odgovara podatku koji želimo izdvojiti. Primjerice ako želimo izdvojiti vrijednosti brzine za svaku lokaciju tada je potrebno napraviti kolekciju objekata koji će spremiti podatke za odgovarajuću lokaciju kao što je prikazano programskim kodom 5.4.

```
///Loading XML document from local folder through StreamReader
var xDocMidas = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/MIDAS_Loop_Traffic_Data_3793828281116175.xml"));

/// Detecting payloadPublication descendant of d2LogicalModel via Linq
XElement payloadPublication = xDocMidas.Descendants().SingleOrDefault(p => p.Name.LocalName ==
"payloadPublication");

///Extracting id's for measurementSite location located within NTISModel-MeasurementSites.xml
var measurementSiteReference = payloadPublication.Descendants().Where(d => d.Name.LocalName ==
"measurementSiteReference");

///Contains speed value of every mesauredValue object.
IEnumerable<XElement> speed = payloadPublication.Descendants().Where(d => d.Name.LocalName == "speed");
```

**Programski kod 5.4.** Izdvajanje podataka o brzini za pojedinu lokaciju

Isti princip izdvajanja i spajanja XML dokumenata se koristi za sve ostale funkcije koje imaju zadaću izdvojiti podatke u odgovarajuće liste objekata koji su spremni za obradu i prikazivanje.

Svaki objekt je definiran vlastitom klasom koje se nalaze unutar datoteke „*DisplayEntities*“ u projektnom stablu. Nakon izdvajanja podataka iz XML dokumenata i prilagođavanja u odgovarajući oblik potrebno je podatke grupirati u memorijske podatkovne setove (engl. *Memory Data Set*). Potrebno je izraditi posebni memorijski podatkovni set za svaku geometrijsku vrstu podataka (točka, linija, mnogokut) i za svaki tip (brzina, protočnost, događaji, vremenske prilike) kao što je prikazano programskim kodom 5.5.

```
public void FillFlowPointDataSet(MemoryDataSet dataSet){…}

public void FillFlowPointHistoricalDataSet(MemoryDataSet dataSet){…}

public void FillFlowLineDataSet(MemoryDataSet dataSet){…}

public void FillSpeedPointDataSet(MemoryDataSet dataSet){…}

public void FillSpeedPointHistoricalDataSet(MemoryDataSet dataSet){…}

public void FillSpeedLineDataSet(MemoryDataSet dataSet){…}

public void FillEventRoadworksDataSet(MemoryDataSet dataSet){…}

public void FillEventUnplannedDataSet(MemoryDataSet dataSet){…}

public void FillEventPolygonDataSet(MemoryDataSet dataSet){…}
```

**Programski kod 5.5.** Funkcije za izradu memorijskih podatkovnih setova

Unutar funkcija za punjenje memorijskih podatkovnih setova definirani su objekti „*feature*“ kojima upravlja ResQmap sustav. Potrebno je stvoriti „*feature*“ objekt koji prihvaća geometrijski tip podatka (točka, linija, mnogokut) te format zapisa koordinata u koordinatnom sustavu kao što je prikazano programskim kodom 5.6. Nakon stvaranja „*feature*“ objekta moguće je postaviti svojstva kao što je tekst za prikaz na karti, boja, veličina teksta i slično.

```
public void FillFlowPointDataSet(MemoryDataSet dataSet)
{
     for (int i = 0; i < PointObjectList.Count; ++i)
     {
          // Create a feature with a point geometry at the position given by the long/lat values.
          Feature feature = new Feature(new PointGeometry(PointObjectList[i].Longitude,
PointObjectList[i].Latidute), Crs.Wgs84LongLat);

          //adding info about speed and vechileFlow on map
          feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue("Current" + "\n" + "Traffic flow: " + PointObjectList[i].VehicleFlow)));

           dataSet.Insert(feature);

           if (i % 50 == 0)
                resQMapModel.CarmentaEngine.MapControl.View.Update();
      }
}
```

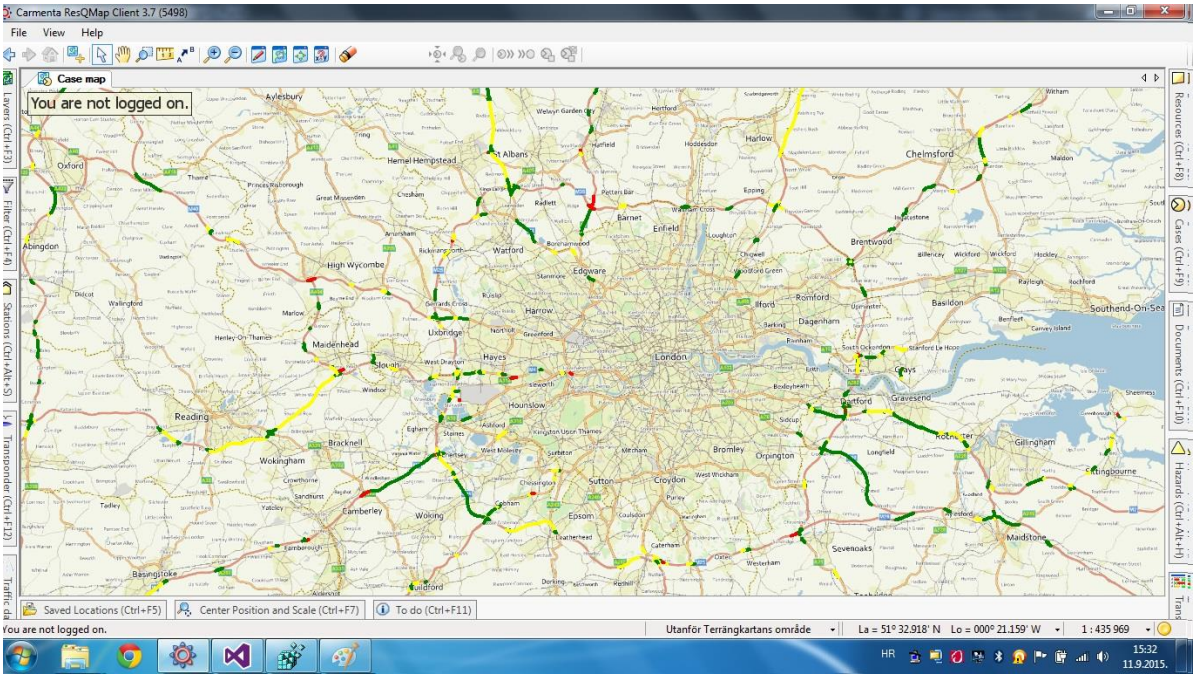
**Programski kod 5.6.** Popunjavanje objekta „*feature*“ podacima

Prikaz podataka i njihovo vizualiziranje od presudne je važnosti za operatere. Na osnovu prikazanih podataka operater vrši odgovarajuće akcije pomoću sustava za upravljanje i nadzor. Primjerice za različite vrijednosti brzina na pojedinoj prometnici potrebno je vizualizirati dionicu odgovarajućom bojom. Tako se u finkciji „*FillSpeedLineDataSet*“ stvaraju „*feature*“ objekti kojima možemo mjenjati svojstva boje u ovisnosti o različitim parametrima kao što je prikazano programskim kodom 5.7.

```
public void FillSpeedLineDataSet(MemoryDataSet dataSet)
{
    for (int i = 0; i < LineObjectList.Count; ++i)
    {
        // Create a feature with a point geometry at the position given by the long/lat values.
        Feature feature = new Feature(new LineGeometry(LineObjectList[i].listPoint),
Crs.Wgs84LongLat);

        double value = double.Parse(LineObjectList[i].Speed, CultureInfo.InvariantCulture);

        if (value < 50)
             feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("COLOR", new
AttributeValue("Red")));
        else if (value >= 50 && value < 80)
             feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("COLOR", new
AttributeValue("Yellow")));
        else if (value >= 80)
              feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("COLOR", new
AttributeValue("Green")));

        feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(LineObjectList[i].Speed)));

        dataSet.Insert(feature);

        if (i % 5000 == 0)
            resQMapModel.CarmentaEngine.MapControl.View.Update();
    }
}
```

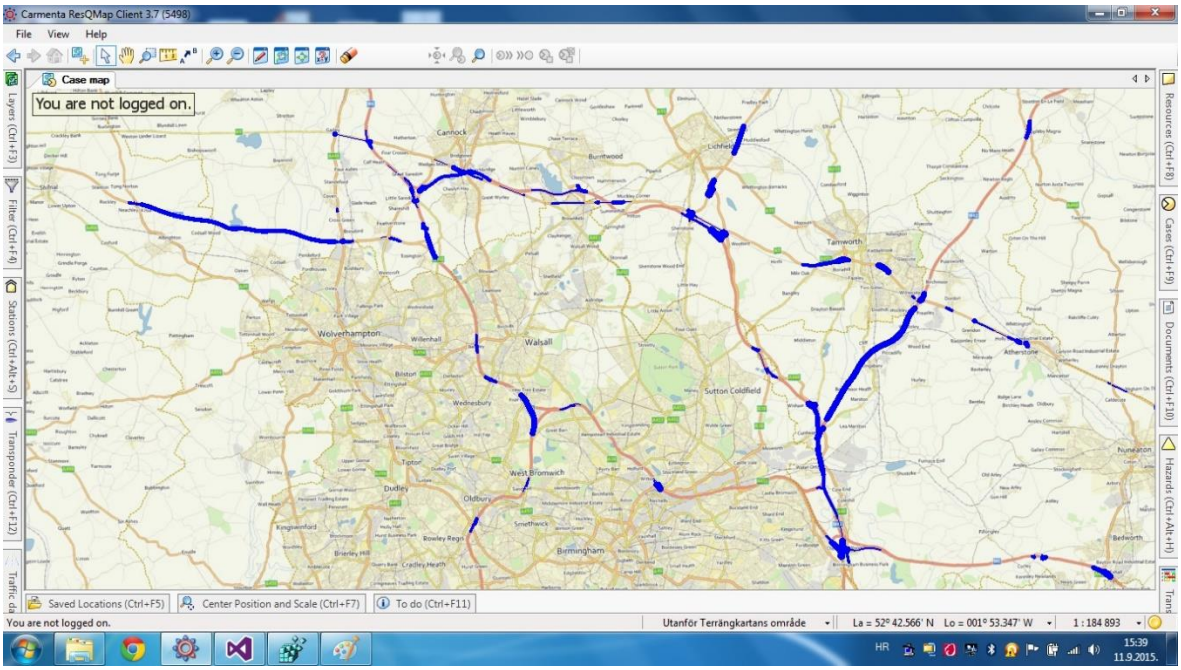
**Programski kod 5.7.** funkcije *FillSpeedLineDataSet*

Za vrijednosti brzine veće od 50 km/h na dionici svojstvo objekta „*feature*“ će poprimiti crvenu boju što znači da se na toj dionici vozi usporeno (kolona, prometna nesreća). Vrijednosti između 50 km/h i 80 km/h se označavaju žutom bojom dok se vrijednosti iznad 80 km/h označavaju zelenom bojom kao što je prikazano snimkom ekrana na slici 5.8. Opisane vrijednosti se odnose samo na prometnice koje su kategorizirane kao autoceste. Sustav omogućava definiranje posebnih pravila u odnosu na tip i kategoriju prometnice. Vrijednosti brzine se mogu odnositi na dionicu ceste (linijski podatak) i na jedno mjesto (točkasti podatak, mjesto mjerenja). Pomoću objekta „*feature*“ moguće je tekstualno prikazati vrijednost brzine uz boje. Kombinacija točkastih i linijskih podataka osigurava veću pouzdanost informacija te na taj način sustav može donijeti odgovarajuću i pravovaljanu akciju.




**Slika 5.8.** Prikazivanje vrijednosti „brzina“ na karti

Isti princip vrijedi i za prikazivanje gustoće prometa na dionicama ceste. Za prikaz gustoće koristi se svojstvo debljine linije. Ako je na određenoj dionici ceste mala količina prometa tj. protočnost onda se ta dionica prikazuje tankom linijom dok se prometnice sa velikom količinom prometa prikazuju debelom linijom kao što je prikazano snimkom ekrana na slici 5.9.



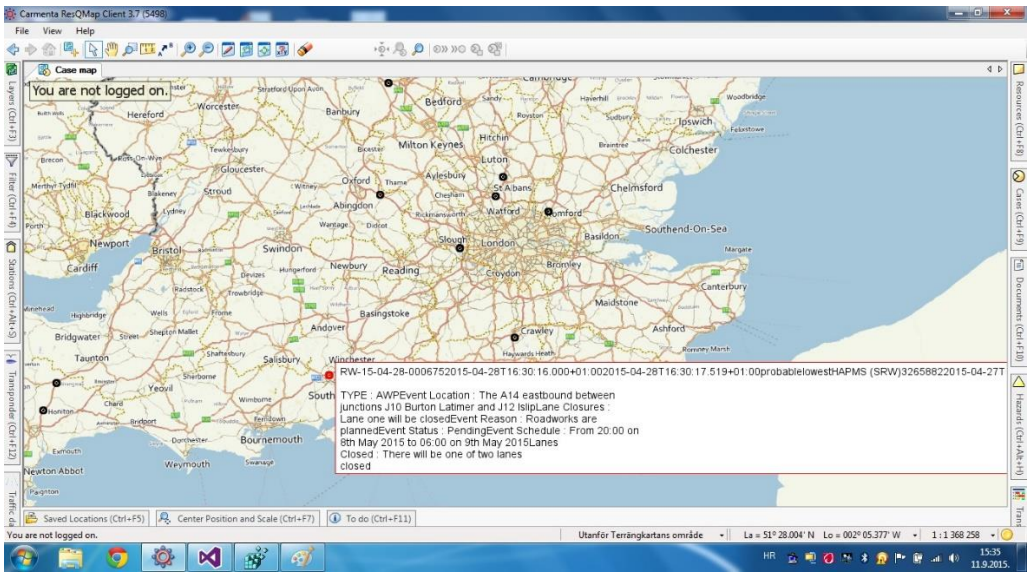

**Slika 5.9.** Prikaz vrijednosti „protočnosti“ na karti

Izrađeni prototip ima mogućnost prikaza vrijednosti brzine, protočnosti, planirane i ne planirane događaje i vremenske prilike odnosno neprilike. Događaj se prikazuje u ovisnosti o lokaciji na kojoj se dogodio uz tekstualni i slikovni opis (engl. *Icon*). Tekstualni opis može sadržavati sve informacije koje se isporučuju unutar publikacije koja sadrži događaje, a to mogu biti vrijeme stvaranja, trajanje, broj kolničkih traka zahvaćenih događajem i slično. Programski kod 5.8. prikazuje izradu objekta „feature“ koji je zadužen za prikaz događaja u ResQmap sustavu.

```
public void FillEventRoadworksDataSet(MemoryDataSet dataSet)
{
    for (int i = 0; i < EventRoadworksObjectList.Count; ++i)
    {
        Feature feature = new Feature(new PointGeometry(EventRoadworksObjectList[i].Longitude,
EventRoadworksObjectList[i].Latitude), Crs.Wgs84LongLat);

        int length = EventRoadworksObjectList[i].StartOfPeriod.Length - 9;
        string lines = string.Join(Environment.NewLine, EventRoadworksObjectList[i].Comment.Split()
             .Select((word, index) => new { word, index })
             .GroupBy(x => x.index / 9)
             .Select(grp => string.Join(" ", grp.Select(x => x.word))));

        feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(EventRoadworksObjectList[i].Condition + "\n\n" + lines)));

        dataSet.Insert(feature);
    }
    resQMapModel.CarmentaEngine.MapControl.View.Update();
}
```

**Programski kod 5.8.** funkcije *FillEventRoadworksDataSet*

Prilikom ispisa tekstualnog sadržaja potrebno je formatirati zapis pomoću znakovne varijable „*lines*“ kao što je vidljivo u programskom kodu 5.8. Događaji su prikazani crnim kružićem na karti kao što je prikazano snimkom ekrana na slici 5.10. Pritiskom na crni kružić korisnik aktivira područje koje sadrži tekstualni opis događaja.




**Slika 5.10.** Prikaz informacija o događaju unutar ResQmap sučelja

Podaci o vremenskim prilikama se prikazuju kao mnogokuti sa znakovnim opisima u ResQmap sustavu. Objektu „*feature*“ potrebno je definirati model tipa „*PolygonGeometry*“ koji prima listu točkastih objekata kao argument na osnovu kojih će ResQmap iscrtati mnogokut kao što je prikazano programskim kodom 5.9.

```
public void FillEventPolygonDataSet(MemoryDataSet dataSet)
{
    for (int i = 0; i < EventPolygonObjectList.Count; ++i)
    {
        Feature feature = new Feature(new PolygonGeometry(EventPolygonObjectList[i].listPoint),
Crs.Wgs84LongLat);

        string lines = string.Join(Environment.NewLine, EventPolygonObjectList[i].Comment.Split()
             .Select((word, index) => new { word, index })
             .GroupBy(x => x.index / 9)
             .Select(grp => string.Join(" ", grp.Select(x => x.word))));

        feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(lines)));

        dataSet.Insert(feature);
     }
     resQMapModel.CarmentaEngine.MapControl.View.Update();
}
```

**Programski kod 5.9.** funkcije *FillEventPolygonDataSet*

Svaki set podataka (engl. *Data Set*) se prikazuje na zasebnom sloju koji se aktivira pomoću kontrola na korisničkom sučelju programskog dodatka.

## 5.5. Registriranje programskog dodatka u ResQmap sustavu

Registriranje programskog dodatka u ResQmap sustavu vrši se pomoću datoteke „*Plugin.cs*“. Funkciji „*RegisterCustomPanels*“ potrebno je predati informacije o nazivu, tipu objekta (SensorPanel), orijentaciji u programskom oknu ResQmap sustava i sličicu kao što je prikazano programskim kodom 5.10.

```
public class Plugin : Carmenta.ResQMapAPI.Plugins.Plugin
{
     protected override void RegisterCustomPanels(CustomPanelRegistrar registrar)
     {
         registrar.Register(
             "Traffic data",
             typeof(SensorPanel),
             CustomPanelOrientation.Left,
             Resource1.traffic_icon);

          base.RegisterCustomPanels(registrar);
     }
}
```

**Programski kod 5.10.** Funkcije *RegisterCustomPanels*

## 5.6. Ocjena učinkovitosti

Upravljanje performansama se općenito odnosi na nadgledanje i mjerenje korištenja resursa prilikom odrađivanja posla. Programski dodatak je testiran sa XML dokumentima različitih veličina te se pratilo ponašanje sustava. Dva seta podataka su korištena kod mjerenje performansa sustava:

- XML dokument koji sadrži sve događaje koji su se dogodili u određenom trenutku (2232 događaja)
- XML dokument koji sadrži točke mjerenja za određenu regiju s opisom svake lokacije prema Datex 2 standardu (1656 lokacija)

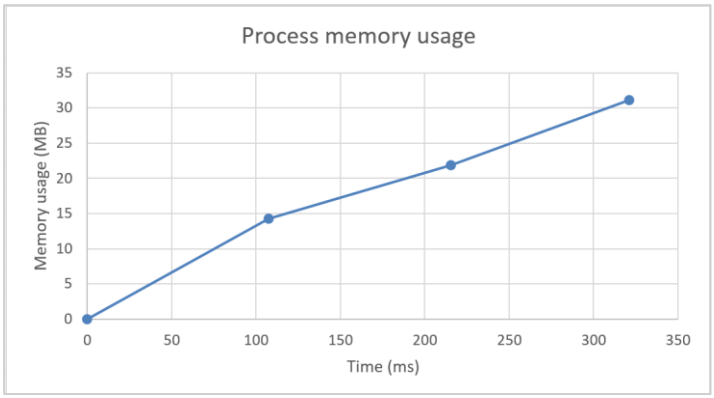
Programski dodatak je testiran na Lenovo Yoga 2 Ultrabook računalu sa Windows 10 operacijskim sustavom. Specifikacije sustava su:

- Procesor: Intel® Core™ i5-421OU CPU 1.70 GHZ
- RAM memorija: 8.00 GB
- Tip operacijskog sustava: 64-bitni operacijski sustav
- Tvrdi disk: 265 GB SSD

Mjerenje performansi sustava je odrađeno u Visual Studio 2015 Enterprise Edition programskom okruženju. Nadzor performansi uključuje korištenje procesne memorije kroz određeno vrijeme kao što je prikazano grafom na slici 5.11. Vrijednosti za navedeni graf mogu se pronaći u tablici 5.1. Promatrani podaci i izmjerene vrijednosti se odnose na korištenje procesne memorije prilikom učitavanja i vađenja podataka iz XML dokumenta koji sadrži podatke o događajima na prometnicama u vremenu t.

**Tab. 5.1.** Rezultati testiranja memorije u vremenu

| **Naziv dokumenta** | **Event_Data_-_Full_Refresh.xml** | | | |
|---|---|---|---|---|
| **Veličina dokumenta** | 11.1 MB | | | |
| **Vrijeme (ms)** | 0 | 107.395 | 215.248 | 321.029 |
| **Memorija (MB)** | 0 | 14.3 | 21.9 | 31.1 |

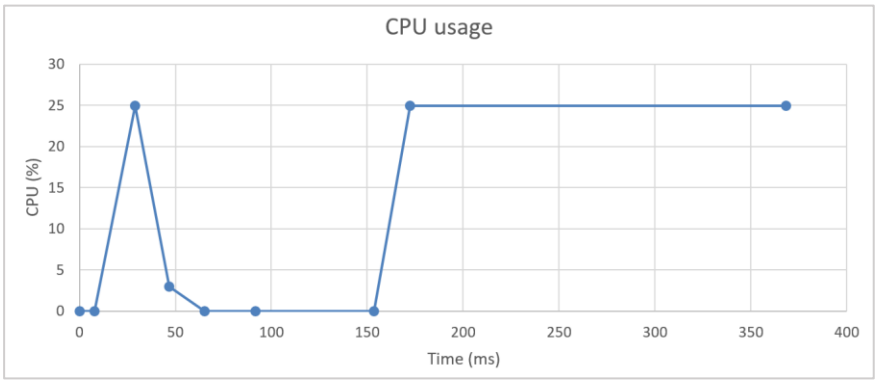



**Slika 5.11.** Graf korištenja memorije prilikom učitavanja XML dokumenta sa događajima

Testiranje je provedeno na procesorskoj jedinici (CPU) uz isto opterećenje koje se koristilo kod testiranja procesne memorije. Grafom na slici 5.12. prikazano je korištenjem procesora prilikom učitavanja XML dokumenta sa događajima u prometu. Vrijednosti za navedeni graf mogu se pronaći u tablici 5.2.

**Tab. 5.2.** Rezutati testiranja CPU-a u vremenu

| **Naziv dokumenta** | **Event_Data_-_Full_Refresh.xml** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Veličina dokumenta** | 11.1 MB | | | | | | | | |
| **Vrijeme (ms)** | 0 | 7.81 | 28.931 | 46.66 | 65.239 | 91.622 | 153.515 | 172.2 | 368.23 |
| **CPU (%)** | 0 | 0 | 25 | 3 | 0 | 0 | 0 | 25 | 25 |



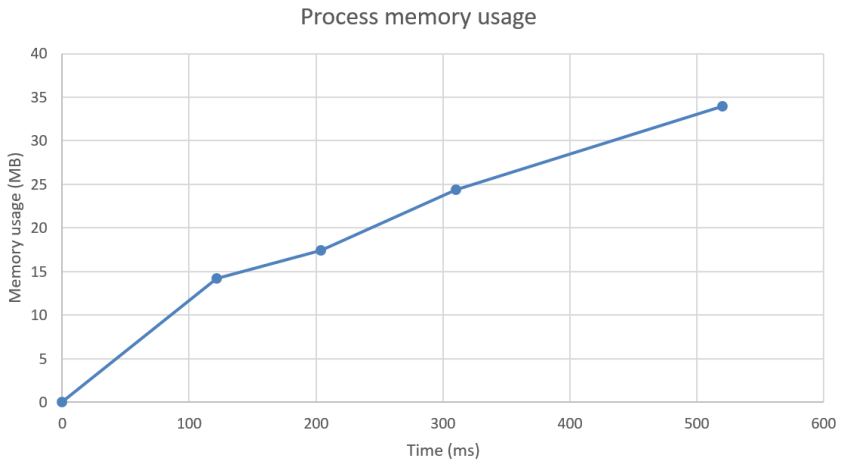

**Slika 5.12.** Graf korištenja CPU-a prilikom učitavanja XML dokumenta sa događajima

Veličina XML dokumenta korištenog tijekom testiranja je 11.1 MB. Korištenje procesne memorije ovisi o količini podataka koje sustav dohvaća od pretplatničkog servisa. Da bi učitali XML dokument od 11.1 MB u memoriju potrebno je 321.029 ms sa spomenutom konfiguracijom

računala. Korištenje CPUa i vrijeme izvršenja ovisi o veličini XML dokumenta i o procesorskim performansama. Korištenje CPU prilikom učitavanja dokumenta iznosi 25% procesorskog kapaciteta što je dobar rezultat s obzirom na memorijsko korištenje. Potrebno je 368.258 ms kako bi CPU obradio XML dokument. Povećanjem računalnih performansi rezultati učitavanja XML datoteka će biti bolji. Iz rezultata se može zaključiti da navedeno računalo korišteno za testiranje nije sposobno izvoditi složenije vizualizaciju podataka zbog ograničenosti resursima. Na većim XML datotekama računalo bi ispalo iz rada što je nepoželjno vladanje. Testiranje sa XML dokumentom koji sadrži mjerne lokacije i njihove opise također uključuje procesnu memoriju i CPU te su rezultati prikazani grafovima na slikama 5.13 i 5.14. Vrijednosti testiranja za navedeni dokument su prikazane tablicama 5.3 i 5.4. Veličina XML dokumenta sa mjernim lokacijama koji se koristio prilikom testiranja iznosi 24.3 MB.

**Tab. 5.3.** Rezultati testiranja memorije u vremenu

| **Naziv dokumenta** | **NTISModel-MeasurementSites-Workaround.xml** | | | | |
|---|---|---|---|---|---|
| **Veličina dokumenta** | 24.3 MB | | | | |
| **Vrijeme (ms)** | 0 | 121.981 | 204.065 | 310.462 | 520.431 |
| **Memorija (MB)** | 0 | 14.2 | 17.4 | 24.4 | 34 |

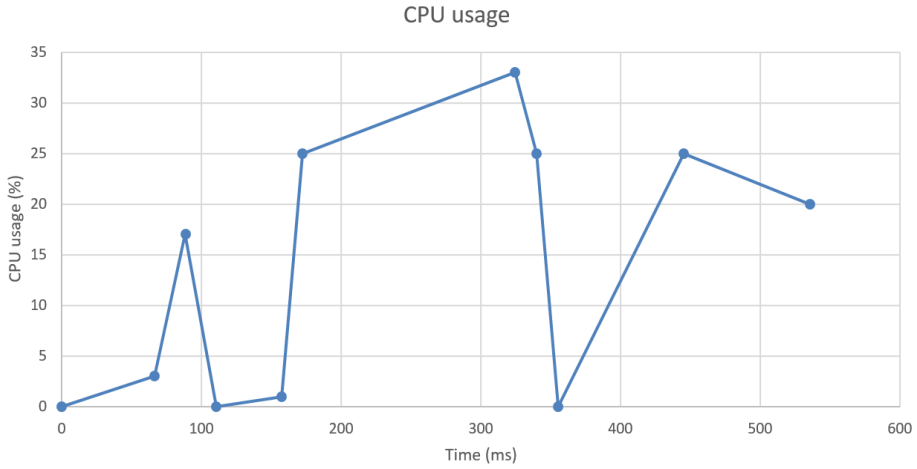



**Slika 5.13.** Graf korištenja memorije prilikom učitavanja XML dokumenta sa mjernim lokacijama

Veći XML dokumenti (24.3 MB) izazivaju veće opterećenje na sustavske resurse. Za XML dokumente veće od 80 MB učitavanje u memoriju može potrajati više od 5-6 sekundi ovisno o memorijskoj brzini.

**Tab. 5.4.** Rezutati testiranja CPU-a u vremenu

| Naziv dokumenta: | NTISModel-MeasurementSites-Workaround.xml |
|---|---|
| **Veličina dokumenta** | 24.3 MB |
| **Vrijeme (ms)** | **Memorija (MB)** |
| 0 | 0 |
| 66.442 | 3 |
| 88.603 | 17 |
| 110.664 | 0 |
| 157.707 | 1 |
| 172.307 | 25 |
| 324.648 | 33 |
| 340.203 | 25 |
| 355.345 | 0 |
| 445.239 | 25 |
| 535.793 | 20 |




**Slika 5.14.** Graf korištenja CPUa prilikom učitavanja XML dokumenta sa mjernim lokacijama

Tijekom mjerenja performansi korišteno računalo nije moglo procesirati velike dokumente. Učitavanje XML dokumenta sa mjernim lokacijama traje 520.431 ms. Najveće opterećenje procesorske jedinice prilikom učitavanja XML dokumenta sa mjernim lokacijama iznosi 33% ukupnog kapaciteta i traje 535.793 ms. Iz grafa se može vidjeti da procesor u 2 navrata dolazi u stanje zastoja pri čemu je njegova iskorištenost 0%. Zastoj se događa iz razloga što korišteno računalo nije namjenjeno za ovakav pothvat što se jasno može vidjeti u njegovim performansama.

Za uspješan rad programskog dodatka u GIS sustavu ResQmap koriste se specijalizirana računala sa velikim performansama. Računala moraju biti sposobna obraditi velike količine podataka koji pristižu u stvarnom vremenu od pružatelja podataka (u ovom slučaju NTIS). Na osnovu prikupljenih podataka računala vrše obrade i pokreću odgovarajuće akcije zadane od strane operatera.

# 6. ZAKLJUČAK

Integracija prometnih senzora će stvoriti ekosustav u kojem će se moći izvršiti odgovarajuća akcija ako se izazove određeni događaj iz senzora. Korištenjem CoordCom sustava sa ResQmap sustavom možemo stvoriti platformu za odlučivanje u sustavima za upravljanje prometom. Podaci prikupljeni sa različitih senzora koji uključuju brojila prometa i slične senzore mogu biti prikupljeni, analizirani i vizualizirani kroz programski dodatak koji se nalazi unutar ResQmap sustava. Podaci se mogu prikazati na GIS bazi podataka i uočiti bilo kakve anomalije koje odstupaju od normalnih vrijednosti te podignuti odgovarajuće akcije unutar CoordCom sustava. Datex 2 je razvio standardizirani način za razmjenu podataka na sistemskoj razini kako bi omogućio bolje upravljanje i nadzor nad Europskim prometnicama te udovoljio sve većom potražnjom za podacima u stvarnom vremenu. Datex 2 format je standardiziran od strane CENa te se koristi u nekoliko zemalja diljem Europe. Ciljevi projekta su kontinuirano prikupljanje i procesiranje prometnih podataka od strane NTIS sustava. Prikupljeni podaci su vizualizirani pomoću ResQmap GIS sustava koji je integriran sa sustavom za upravljanje i nadzor. Podaci se trenutno izmjenjuju što omogućava brže i responzivnije upravljanje prometnom mrežom. CoordCom i ResQmap su skalabilni i fleksibilni sustavi što znači da se jednostavno integriraju sa postojećim sustavima što omogućuje žurnim službama jednostavno i efikasno dijeljenje podataka. Integracija standardiziranih prometnih podataka sa sustavom za upravljanje i nadzor dovodi do boljeg upravljanja prometnim mrežama uključujući brže odzivno vrijeme za različite akcije. Sustav je jednostavno primjenjiv na sve države unutar Europske Unije koje koriste Datex 2 standard.

# LITERATURA


[1] L. Wei-feng, C.Wei and H. Jian, "Research on a DATEX II Based Dynamic Traffic Information Publish Platform," Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on, Shanghai, 2008, pp. 412-416

[2] A. Raines and P. Rowley, "Coordinated traffic management through data exchange," Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members' Conference, IET, Manchester, 2008, pp. 1-4.

[3] Datex II - The standard for ITS on European Roads, link http://www.datex2.eu/, May 2016

[4] T. Andersson, S. Petersson and P. Värbrand, Decision support for efficient ambulance logistics, Operational Research for Health Policy: Making Better Decisions: Proceedings of the 31st Annual Conference of the European Working Group on Operational Research Applied to Health Services, 2007.

[5] COORDCOM, Emergency Response Communication Center, Ericsson, link http://www.ericsson.com/res/region_RLAM/press-release/2011-09-19-01-coordcom-futurecom.pdf, April 2016.

[6] J. Barcelo, and M. Kuwahara, "Traffic Data Collection and its Standardization", Springer Science & Business Media, Barcelona, July 2010.

[7] Datex II v2.0, User Guide, http://www.datex2.eu/sites/www.datex2.eu/files/DATEXIIv2.0-UserGuide.pdf, 30.6.2016

[8] Datex II v2.0, Software Developers Guide, http://www.datex2.eu/sites/www.datex2.eu/files/DATEXIIv2.0-DevGuide.pdf, 30.6.2016

[9] D.J. Morris "Communication for Command and Control Systems: Int. Series on System and Control vol. 5", Elsevier, Haifa, May 2014

[10] NTIS Subscriber Service example, https://github.com/ntisservices/ntis-dotnet-web-services-Release2.5, 30.6.2016

## SAŽETAK

Klučna uloga inteligentnih transportnih sustava je prepoznavanje performansi, stanja i uvjeta transportnih sustava. Prometni podaci se prikupljaju, spajaju i obrađuju od strane različitih upravitelja ICT sustava transportnog sustava. Prometni podaci se razmjenjuje između sustava koristeći Datex II standard. Diplomski rad opisuje razvoj programskog dodatka za prikupljanje i obradu prometnog podatkovnog toka, vizualiziranje podataka kroz ResQmap GIS sustav i integraciju sa sustavom za upravljanje i nadzor CoordCom. Konačni rezultat je sustav u kojemu događaji dobiveni iz prometnih podataka mogu pokrenuti odgovarajuću akciju.



**Ključne riječti:** Sustavi za upravljanje i nadzor, standardi za razmjenu podataka, Datex II, ITS, promet



## ABSTRACT

Situation awareness regarding transport system performances, status and condition is key purpose of ITSs (Intelligent Transport Systems). Transport related data is being collected, aggregated and processed by ICT systems of various stakeholders within the transport system. Traffic related data is being exchanged between systems using Datex II standard. Paper describes development of a plugin for gathering and processing traffic related data stream, data visualization through ResQMap GIS system and integrated with C&C (Command and Control) system – CoordCom. Final result is ecosystem where appropriate actions could be launched if triggered by event from traffic related data stream.



**Keywords:** Command and Control Systems, Data Exchange Standards, Datex II, ITS, Traffic

# ŽIVOTOPIS

Dinko Jakovljević rođen je 15. listopada 1992. godine u Požegi. Pohađao je osnovnu školu „Julija Kempfa“ u Požegi nakon čega upisuje srednju Tehničku školu, zanimanje tehničara a računarstvo. Tijekom srednjoškolskog školovanja sudjeluje na međunarodnim natjecanju ACSL (American Classrom Science League) gdje ostvaruje dva puta prvo mjesto. U 2011. godini dobiva zahvalnicu Tehničke škole Požega radi uspješnog predstavljanja škole na državnim i međunarodnim natjecanjima. Nakon završene srednje škole upisuje se na Elektrotehnički fakultet u Osijeku na preddiplomski studij smjer računarstvo. Iste godine postaje članom međunarodnog udruženja inženjera elektrotehnike i elektronike pod nazivom IEEE kroz lolakni studentski ogranak. Pridružuje se Microsoft Student Partner programu u Hrvatskoj. Unutar IEEE studentskog ogranka u Osijeku vršio je funkcije tajnika i predsjednika ogranka. Kroz rad u ogranku s kolegama je osvojio mnoge nagrade i priznanja unutar IEEE organizacije te su proveli veliki broj projekata od kojih je najznačajniji natjecanje u izradi mobilnih aplikacija pod nazivom IEEEmadC. Nakon završetka preddiplomskog studija 2013. godine, upisuje diplomski studij procesnog računarstva na Elektrotehničkom fakultetu Osijek. Godinu dana kasnije dolazi na mjesto koordinatora za nagrade i natjecanja u regionalnom odboru za studentske aktivnosti IEEEa koji obuhvaća Europu, Afriku i Bliski Istok. Iste godine postaje predstavnik studenata Hrvatske sekcije IEEEa. Tijekom studiranja dobio je tri priznanja od dekana Elektrotehničkog fakulteta i odradio dvije stručne prakse u tvrtkama Farmeron iz Osijeka i Ericsson Nikola Tesla iz Zagreba.

Dinko Jakovljević

_____

# PRILOZI

**Prilog 1.** Programski kod datoteke “Plugin.cs”

```
using Carmenta.ResQMapAPI.Plugins.CustomEntities.CustomEntityDefinitions;
using Carmenta.ResQMapAPI.Plugins.Registrars;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SensorPlugin
{
    public class Plugin : Carmenta.ResQMapAPI.Plugins.Plugin
    {
        protected override void RegisterCustomPanels(CustomPanelRegistrar registrar)
        {
            registrar.Register(
                "Traffic data",
                typeof(SensorPanel),
                CustomPanelOrientation.Left,
                Resource1.traffic_icon);

            base.RegisterCustomPanels(registrar);
        }
    }
}
```

**Prilog 2.** Programski kod datoteke "SensorPanel.cs"

```
/*
 * This is project: Integration of stationary and mobile M2M sensors with Command &
Control Platform
 * Following people have worked on this project:
 * Andrea Marasovic, Frane Antunovic, Dinko Jakovljevic, MICO DUJAK, Kresimir Vidovic,
Darko Sobar
 * Special tribute: Damir Tomic
 */

using Carmenta.Engine;
using Carmenta.ResQMapAPI.Models;
using Carmenta.ResQMapAPI.Models.CarmentaEngineModels;
using Carmenta.ResQMapAPI.Plugins.CustomEntities;
using System;
using System.Xml;
using System.Xml.Linq;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SensorPlugin
{
    /// <summary>
    /// The Sensors panel implementation that handles all business logic regarding the
data gathered from sensors,
    /// </summary>
    public class SensorPanel : CustomPanel<SensorControl>
    {
        /// <summary>
        /// The root model from the ResQMap API, received in the <see cref="OnAttached"/>
method.
        /// </summary>
        public ResQMapModel resQMapModel;

        /// <summary>
        /// The memory dataset containing all Events data.This dataset is filled in
the<see cref="FillDataSet"/> method
        /// which reads custom data(lat, long, speed, vechile flow) related data from
Sensors located on map which are stored in .csv file
        /// Events data set contains description of event occured on particular road
section
        /// </summary>
        public MemoryDataSet dataSetFlowPoint;
        public MemoryDataSet dataSetFlowPointHistorical;
        public MemoryDataSet dataSetFlowLine;
        public MemoryDataSet dataSetSpeedPoint;
        public MemoryDataSet dataSetSpeedPointHistorical;
        public MemoryDataSet dataSetSpeedLine;
        public MemoryDataSet dataSetEventRoadworks;
        public MemoryDataSet dataSetEventUnplanned;
        public MemoryDataSet dataSetEventPolygon;

        public List<DisplayPointObject> PointObjectList = new List<DisplayPointObject>();
        public List<DisplayPointObject> HistoricalPointObjectList = new
List<DisplayPointObject>();
        public List<DisplayLineObject> LineObjectList = new List<DisplayLineObject>();
```

```
        public List<DisplayEventPointData> EventRoadworksObjectList = new
List<DisplayEventPointData>();
        public List<DisplayEventUnplannedObject> EventUnplannedObjectList = new
List<DisplayEventUnplannedObject>();
        public List<DisplayEventPolygonObject> EventPolygonObjectList = new
List<DisplayEventPolygonObject>();

        public override void OnAttached(ResQMapModel resQMapModel)
        {
            // Remember the ResQMapModel for use later on
            this.resQMapModel = resQMapModel;

            this.dataSetFlowPoint = AddFlowPointDataLayer();
            this.dataSetFlowPointHistorical = AddFlowPointHistoricalDataLayer();
            this.dataSetFlowLine = AddFlowLineDataLayer();
            this.dataSetSpeedPoint = AddSpeedPointDataLayer();
            this.dataSetSpeedPointHistorical = AddSpeedPointHistoricalDataLayer();
            this.dataSetSpeedLine = AddSpeedLineDataLayer();
            this.dataSetEventRoadworks = AddEventRoadworksDataLayer();
            this.dataSetEventUnplanned = AddEventUnplannedDataLayer();
            this.dataSetEventPolygon = AddEventPolygonDataLayer();

            FillPointObjectList();
            FillHistoricalPointObjectList();
            FillLineObjectList();
            FillEventRoadWorksObjectList();
            FillEventUnplannedObjectList();
            FillEventWeatherObjectList();
        }


        /// <summary>
        /// Adds the CarmentaEngine layer that will be used to visualize POINT data from
All sensor sources in the map.
        /// </summary>
        /// <returns>The MemoryDataSet that is created at the end of the operator chain
for this new layer.</returns>
        private MemoryDataSet AddFlowPointDataLayer()
        {
            //get the layer set in which we should put the new layer
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            //Create the layer and put it in the layer set
            var layer = new OrdinaryLayer { Selectable = false };
            layerSet.Layers.Add(layer);

            //Create the read operator and the data set using WGS84LongLat as reference
system
            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            //Create the visualization operator which will visualize lines
            var visualizationOperator = new VisualizationOperator(readOperator);
            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Dots2 });
            visualizationOperator.Visualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = -25,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
```

```
                BackgroundOutlineColor = System.Drawing.Color.Blue,
                BackgroundPadding = 5

            });

            //Set the visualization operator as the input to the layer. Now the operator
chain is complete
            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddFlowPointHistoricalDataLayer()
        {
            //get the layer set in which we should put the new layer
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            //Create the layer and put it in the layer set
            var layer = new OrdinaryLayer { Selectable = false };
            layerSet.Layers.Add(layer);

            //Create the read operator and the data set using WGS84LongLat as reference
system
            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            //Create the visualization operator which will visualize lines
            var visualizationOperator = new VisualizationOperator(readOperator);
            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Dots2 });
            visualizationOperator.Visualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 20,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.LightBlue,
                BackgroundPadding = 5

            });

            //Set the visualization operator as the input to the layer. Now the operator
chain is complete
            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddFlowLineDataLayer()
        {
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            var layer = new OrdinaryLayer { Selectable = true };
            layerSet.Layers.Add(layer);

            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            var visualizationOperator = new VisualizationOperator(readOperator);
```

```
            visualizationOperator.Visualizers.Add(new LineVisualizer
            {
                Style = LineStyle.SolidLine,
                Color = System.Drawing.Color.Blue,
                Width = new IndirectAttributeVariable<Double>("WIDTH")

            });

            visualizationOperator.SelectionVisualizers.Add(new LineVisualizer
            {
                Style = LineStyle.SolidLine,
                Color = System.Drawing.Color.Blue,
                Width = new IndirectAttributeVariable<Double>("WIDTH")

            });

            visualizationOperator.SelectionVisualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = -25,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.LightBlue,
                BackgroundPadding = 5
            });

            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddSpeedPointDataLayer()
        {
            //get the layer set in which we should put the new layer
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            //Create the layer and put it in the layer set
            var layer = new OrdinaryLayer { Selectable = false };
            layerSet.Layers.Add(layer);

            //Create the read operator and the data set using WGS84LongLat as reference
system
            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            //Create the visualization operator which will visualize lines
            var visualizationOperator = new VisualizationOperator(readOperator);
            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Dots2 });
            visualizationOperator.Visualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = -25,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.Green,
                BackgroundPadding = 5

            });
```

```
            //Set the visualization operator as the input to the layer. Now the operator
chain is complete
            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddSpeedPointHistoricalDataLayer()
        {
            //get the layer set in which we should put the new layer
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            //Create the layer and put it in the layer set
            var layer = new OrdinaryLayer { Selectable = false };
            layerSet.Layers.Add(layer);

            //Create the read operator and the data set using WGS84LongLat as reference
system
            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            //Create the visualization operator which will visualize lines
            var visualizationOperator = new VisualizationOperator(readOperator);
            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Dots2 });
            visualizationOperator.Visualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 20,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.LightGreen,
                BackgroundPadding = 5

            });

            //Set the visualization operator as the input to the layer. Now the operator
chain is complete
            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddSpeedLineDataLayer()
        {
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            var layer = new OrdinaryLayer { Selectable = true };
            layerSet.Layers.Add(layer);

            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            var visualizationOperator = new VisualizationOperator(readOperator);

            visualizationOperator.Visualizers.Add(new LineVisualizer
            {
                Style = LineStyle.SolidLine,
                Color = new IndirectAttributeVariable<System.Drawing.Color>("COLOR"),
```

```
                Width = 4

            });

            visualizationOperator.SelectionVisualizers.Add(new LineVisualizer
            {
                Style = LineStyle.SolidLine,
                Color = new IndirectAttributeVariable<System.Drawing.Color>("COLOR"),
                Width = 4

            });

            visualizationOperator.SelectionVisualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 0,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.LightBlue,
                BackgroundPadding = 5
            });

            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddEventRoadworksDataLayer()
        {
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            var layer = new OrdinaryLayer { Selectable = true };
            layerSet.Layers.Add(layer);

            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            var visualizationOperator = new VisualizationOperator(readOperator);

            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Bullseye });

            visualizationOperator.SelectionVisualizers.Add(new SymbolVisualizer { Symbol
= Symbol.Bullseye, Color = System.Drawing.Color.Red });
            visualizationOperator.SelectionVisualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 0,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.Red,
                BackgroundPadding = 5,
            });

            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddEventUnplannedDataLayer()
        {
```

```
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            var layer = new OrdinaryLayer { Selectable = true };
            layerSet.Layers.Add(layer);

            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            var visualizationOperator = new VisualizationOperator(readOperator);

            visualizationOperator.Visualizers.Add(new SymbolVisualizer { Symbol =
Symbol.Bullseye });

            visualizationOperator.SelectionVisualizers.Add(new SymbolVisualizer { Symbol
= Symbol.Bullseye, Color = System.Drawing.Color.Red });
            visualizationOperator.SelectionVisualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 0,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.Red,
                BackgroundPadding = 5,
            });

            layer.Input = visualizationOperator;

            return dataSet;
        }

        private MemoryDataSet AddEventPolygonDataLayer()
        {
            LayerSet layerSet =
resQMapModel.CarmentaEngine.GetLayerSet(CarmentaEngineModel.LayerSetDefinitions.AboveOver
lays);

            var layer = new OrdinaryLayer { Selectable = false };
            layerSet.Layers.Add(layer);

            var dataSet = new MemoryDataSet(Crs.Wgs84LongLat);
            var readOperator = new ReadOperator(dataSet);

            var visualizationOperator = new VisualizationOperator(readOperator);

            visualizationOperator.Visualizers.Add(new PolygonVisualizer
            {
                // 43 jer je Mićo rekao
                Color = System.Drawing.Color.FromArgb(43, 255, 0, 0)
            });

            visualizationOperator.Visualizers.Add(new TextVisualizer
            {
                Text = new IndirectAttributeVariable<string>("INFO"),
                OffsetY = 0,
                OffsetX = 15,
                BackgroundColor = System.Drawing.Color.White,
                BackgroundOutlineColor = System.Drawing.Color.Yellow,
                BackgroundPadding = 5
            });

            layer.Input = visualizationOperator;
```

```
            return dataSet;
        }


        public void FillPointObjectList()
        {
            #region Loading and parsing MIDAS XML (Point data)

            /// Loading XML document from local folder through StreamReader
            var xDocMidas = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/MIDAS_Loop_Traffic_Data_3793828281116175.xml
"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublication = xDocMidas.Descendants().SingleOrDefault(p =>
p.Name.LocalName == "payloadPublication");
            /// Extracting id's for measurementSite location located within NTISModel-
MeasurementSites.xml
            var measurementSiteReference = payloadPublication.Descendants().Where(d =>
d.Name.LocalName == "measurementSiteReference");
            /// Contains speed value of every mesauredValue object.
            IEnumerable<XElement> speed = payloadPublication.Descendants().Where(d =>
d.Name.LocalName == "speed");
            /// Contains speed value of every mesauredValue object.
            IEnumerable<XElement> vechileFlow = payloadPublication.Descendants().Where(d
=> d.Name.LocalName == "vehicleFlowRate");

            var MidasDataPart = speed.Zip(vechileFlow, (s, v) => new { speed = s,
vechileFlow = v });
            var MidasData = MidasDataPart.Zip(measurementSiteReference, (m, e) => new {
MidasDataPart = m, measurementSiteReference = e });

            #endregion

            #region Loading and Parsing Measurement sites

            /// Loading XML document from local folder through StreamReader
            var xDocMeasurementSite = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/NTISModel-MeasurementSites.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationMeasurement =
xDocMeasurementSite.Descendants().SingleOrDefault(p => p.Name.LocalName ==
"payloadPublication");
            /// Extracting data about location (long, lat)
            IEnumerable<XElement> latitude = xDocMeasurementSite.Descendants().Where(d =>
d.Name.LocalName == "latitude");
            /// Extracting id
            IEnumerable<XElement> longitude = xDocMeasurementSite.Descendants().Where(d
=> d.Name.LocalName == "longitude");

            var MeasurementData = latitude.Zip(longitude, (ld, l) => new { latitude = ld,
longitude = l });

            #endregion

            //list of objects loaded with MIDAS data and fused with sensor locations
            var midasData = MidasData.Zip(MeasurementData, (mi, m) => new { MidasData =
mi, MeasurementData = m });

            //loading data from sensor into custom made object for manipulation in
ResQMap. This is MIDAS PointData
```

```
            foreach (var item in midasData)
            {
                DisplayPointObject MidasPointObject = new DisplayPointObject();
                MidasPointObject.Latidute =
double.Parse(item.MeasurementData.latitude.Value, CultureInfo.InvariantCulture);
                MidasPointObject.Longitude =
double.Parse(item.MeasurementData.longitude.Value, CultureInfo.InvariantCulture);
                MidasPointObject.Speed = item.MidasData.MidasDataPart.speed.Value;
                MidasPointObject.VehicleFlow =
item.MidasData.MidasDataPart.vechileFlow.Value;

                PointObjectList.Add(MidasPointObject);
            }
        }

        public void FillHistoricalPointObjectList()
        {
            #region Loading and parsing MIDAS XML (Point data)

            /// Loading XML document from local folder through StreamReader
            var xDocMidas = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/MIDAS_Loop_Traffic_Data_3793830663507513.xml
"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublication = xDocMidas.Descendants().SingleOrDefault(p =>
p.Name.LocalName == "payloadPublication");
            /// Extracting id's for measurementSite location located within NTISModel-
MeasurementSites.xml
            var measurementSiteReference = payloadPublication.Descendants().Where(d =>
d.Name.LocalName == "measurementSiteReference");
            /// Contains speed value of every mesauredValue object.
            IEnumerable<XElement> speed = payloadPublication.Descendants().Where(d =>
d.Name.LocalName == "speed");
            /// Contains speed value of every mesauredValue object.
            IEnumerable<XElement> vechileFlow = payloadPublication.Descendants().Where(d
=> d.Name.LocalName == "vehicleFlowRate");

            var MidasDataPart = speed.Zip(vechileFlow, (s, v) => new { speed = s,
vechileFlow = v });
            var MidasData = MidasDataPart.Zip(measurementSiteReference, (m, e) => new {
MidasDataPart = m, measurementSiteReference = e });

            #endregion

            #region Loading and Parsing Measurement sites

            /// Loading XML document from local folder through StreamReader
            var xDocMeasurementSite = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/NTISModel-MeasurementSites.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationMeasurement =
xDocMeasurementSite.Descendants().SingleOrDefault(p => p.Name.LocalName ==
"payloadPublication");
            /// Extracting data about location (long, lat)
            IEnumerable<XElement> latitude = xDocMeasurementSite.Descendants().Where(d =>
d.Name.LocalName == "latitude");
            /// Extracting id
```

```
            IEnumerable<XElement> longitude = xDocMeasurementSite.Descendants().Where(d
=> d.Name.LocalName == "longitude");

            var MeasurementData = latitude.Zip(longitude, (ld, l) => new { latitude = ld,
longitude = l });

            #endregion

            //list of objects loaded with MIDAS data and fused with sensor locations
            var midasData = MidasData.Zip(MeasurementData, (mi, m) => new { MidasData =
mi, MeasurementData = m });

            //loading data from sensor into custom made object for manipulation in
ResQMap. This is MIDAS PointData



            foreach (var item in midasData)
            {
                DisplayPointObject MidasPointObject = new DisplayPointObject();
                MidasPointObject.Latidute =
double.Parse(item.MeasurementData.latitude.Value, CultureInfo.InvariantCulture);
                MidasPointObject.Longitude =
double.Parse(item.MeasurementData.longitude.Value, CultureInfo.InvariantCulture);
                MidasPointObject.Speed = item.MidasData.MidasDataPart.speed.Value;
                MidasPointObject.VehicleFlow =
item.MidasData.MidasDataPart.vechileFlow.Value;

                HistoricalPointObjectList.Add(MidasPointObject);
            }

        }

        public void FillLineObjectList()
        {
            #region Loading and parsing FUSED FVD and Sensor data (Line data)

            /// Loading XML document from local folder through StreamReader
            var xDocFused = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/Fused_Sensor-
only_PTD_3793817774626104.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationFused = xDocFused.Descendants().SingleOrDefault(p
=> p.Name.LocalName == "payloadPublication");
            /// Extracting id's to predefined location in NTISModel-
PredefinedLocations.xml file
            //IEnumerable<XElement> predefinedLocationReference =
payloadPublicationFused.Descendants().Where(d => d.Name.LocalName ==
"predefinedLocationReference");
            /// Contains list of objects that contains data about speed
            IEnumerable<XElement> speedL = payloadPublicationFused.Descendants().Where(d
=> d.Name.LocalName == "speed");
            /// Contains list of objects that contains data about vehicle flow rate
            IEnumerable<XElement> vehicleFlowRate =
payloadPublicationFused.Descendants().Where(d => d.Name.LocalName == "vehicleFlowRate");

            var FusedData = speedL.Zip(vehicleFlowRate, (s, v) => new { speedL = s,
vehicleFlowRate = v });
            //var FusedData = FusedDataPartTwo.Zip(predefinedLocationReference, (f, p) =>
new { FusedDataPartTwo = f, predefinedLocationReference = p });

            #endregion
```

```
            #region Loading and Parsing PredefinedLocation

            /// Loading XML document from local folder through StreamReader
            var xDocPreDefinedLocation = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/NTISModel-PredefinedLocations.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationPreDefined =
xDocPreDefinedLocation.Descendants().SingleOrDefault(p => p.Name.LocalName ==
"payloadPublication");
            /// Detecting all location for particular event
            IEnumerable<XElement> predefinedLocationContainer =
payloadPublicationPreDefined.Descendants().Where(d => d.Name.LocalName ==
"predefinedLocationContainer");
            /// Extracting coordinates (long, lat)
            IEnumerable<XElement> pointCoordinates =
payloadPublicationPreDefined.Descendants().Where(d => d.Name.LocalName ==
"pointCoordinates");
            /// Extracting id
            //var id = xDocPreDefinedLocation.Descendants().Attributes("id").Select(e =>
e.Value);

            #endregion

            //list of objects loaded with FUSED data and fused with sensor locations
            var fusedDataContainer = FusedData.Zip(predefinedLocationContainer, (f, p) =>
new { FusedData = f, predefinedLocationContainer = p });
            var fusedData = fusedDataContainer.Zip(pointCoordinates, (f, p) => new {
fusedDataContainer = f, pointCoordinates = p });



            XNode tempNode;

            IEnumerable<XElement> pointsNode;

            foreach (var item in fusedData)
            {
                DisplayLineObject LineObject = new DisplayLineObject();

                if (item.fusedDataContainer.predefinedLocationContainer.Name.LocalName ==
"predefinedLocationContainer")
                {
                    //extracting coordinates for every location within LocationContainer
in PredefinedLocation.xml
                    foreach (var node in
item.fusedDataContainer.predefinedLocationContainer.Descendants())
                    {
                        if (node.Name.LocalName == "predefinedLocation" &&
node.LastAttribute.Name == "id")
                        {
                            foreach (var loop in node.Descendants())
                            {
                                if (loop.Name.LocalName == "pointCoordinates")
                                {

                                    pointsNode = node.Descendants().Where(d =>
d.Name.LocalName == "pointCoordinates");
                                    tempNode = pointsNode.First();
                                    var Pointlatitude =
((System.Xml.Linq.XElement)((System.Xml.Linq.XContainer)tempNode).FirstNode).Value;
```

```
                                    var Pointlongitude =
((System.Xml.Linq.XElement)((System.Xml.Linq.XContainer)tempNode).LastNode).Value;
                                    Point point = new
Point(Convert.ToDouble(Pointlongitude, CultureInfo.InvariantCulture),
double.Parse(Pointlatitude, CultureInfo.InvariantCulture));
                                    LineObject.listPoint.Add(point);
                                }
                            }
                        }
                    }
                }
                LineObject.Speed = item.fusedDataContainer.FusedData.speedL.Value;
                LineObject.VehicleFlow =
item.fusedDataContainer.FusedData.vehicleFlowRate.Value;
                LineObjectList.Add(LineObject);
            }
        }

        public void FillEventRoadWorksObjectList()
        {
            #region Loading Event data

            /// Loading XML document from local folder through StreamReader
            var xDocEvents = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/Event_Data_-_Full_Refresh.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationEvent = xDocEvents.Descendants().SingleOrDefault(p
=> p.Name.LocalName == "payloadPublication");
            /// Detecting overallStartTime for Event that occured
            IEnumerable<XElement> startOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "startOfPeriod");
            /// Detectign overallEndTime for Event that occured
            IEnumerable<XElement> endOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "endOfPeriod");
            /// Extracting comment for event
            IEnumerable<XElement> comment = payloadPublicationEvent.Descendants().Where(d
=> d.Name.LocalName == "comment");
            /// Detecting infor about event that occured (Weather, Roadworks, etc)
            IEnumerable<XElement> condition =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "situationRecord");
            /// Detecting object which contains points (locations)
            IEnumerable<XElement> locationContainedInGroup =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationContainedInGroup");

            IEnumerable<XElement> locationForDisplay =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationForDisplay");
            /// extracting accidents events from XML
            IEnumerable<XElement> accidents = condition.Where(x => x.FirstAttribute.Value
== "Accident");

            IEnumerable<XElement> comments = accidents.Descendants().Where(d =>
d.Name.LocalName == "comment");

            /// extracting weather conditions from XML
            IEnumerable<XElement> weather = condition.Where(x => x.FirstAttribute.Value
== "PoorEnvironmentConditions");

            IEnumerable<XElement> whecomm = weather.Descendants().Where(d =>
d.Name.LocalName == "comment");
```

```
            IEnumerable<XElement> pointCoordinatess =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "pointCoordinates");

            var EventDataTime = startOfPeriod.Zip(endOfPeriod, (s, e) => new {
startOfPeriod = s, endOfPeriod = e });
            var EventDataCom = EventDataTime.Zip(comment, (e, c) => new { EventDataTime =
e, comment = c });
            var EventDataLoc = EventDataCom.Zip(locationContainedInGroup, (e, l) => new {
EventDataCom = e, locationContainedInGroup = l });
            var eventData = EventDataLoc.Zip(condition, (e, c) => new { EventDataLoc = e,
condition = c });
            var ev = eventData.Zip(locationForDisplay, (e, l) => new { eventData = e,
locationForDisplay = l });
            var eve = ev.Zip(pointCoordinatess, (e, p) => new { ev = e, pointCoordinatess
= p });

            /// Accident object list
            var acc = accidents.Zip(locationForDisplay, (a, l) => new { accidents = a,
locationForDisplay = l });
            var acci = acc.Zip(comments, (a, c) => new { acc = a, comments = c });

            /// Weather object list
            var wea = weather.Zip(whecomm, (w, wh) => new { weather = w, whecomm = wh });


            #endregion

            // temp variables that helps in detecting certain node in XML
            XNode tempNode;

            /// parsing Maintenance event into objects
            foreach (var item in eve)
            {
                DisplayEventPointData EventPointObject = new DisplayEventPointData();

                /// All locations are nested in locationContainedInGroup XML tag
                if (item.ev.eventData.condition.FirstAttribute.Value ==
"MaintenanceWorks")
                {
                    if
(item.ev.eventData.EventDataLoc.locationContainedInGroup.Name.LocalName ==
"locationContainedInGroup")
                    {
                        foreach (var node in
item.ev.eventData.EventDataLoc.locationContainedInGroup.Descendants())
                        {
                            /// detecting sensor points that represent roadwork events
                            if (node.Name.LocalName == "locationForDisplay")
                            {
                                var PointLatitude = node.FirstNode;
                                tempNode = node.LastNode;
                                var PointLongitude =
((System.Xml.Linq.XElement)tempNode).Value;
                                EventPointObject.Latitude =
double.Parse(((System.Xml.Linq.XElement)PointLatitude).Value,
CultureInfo.InvariantCulture);
                                EventPointObject.Longitude = double.Parse(PointLongitude,
CultureInfo.InvariantCulture);

                                EventPointObject.StartOfPeriod =
item.ev.eventData.EventDataLoc.EventDataCom.EventDataTime.startOfPeriod.Value;
                                EventPointObject.EndOfPeriod =
item.ev.eventData.EventDataLoc.EventDataCom.EventDataTime.endOfPeriod.Value;
```

```
                                EventPointObject.Condition =
item.ev.eventData.condition.Value;
                                EventPointObject.Comment =
item.ev.eventData.EventDataLoc.EventDataCom.comment.Value;
                                EventRoadworksObjectList.Add(EventPointObject);
                            }
                        }
                    }
                }
            }
        }

        public void FillEventUnplannedObjectList()
        {
            #region Loading Event data

            /// Loading XML document from local folder through StreamReader
            var xDocEvents = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/Event_Data_-_Full_Refresh.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
            XElement payloadPublicationEvent = xDocEvents.Descendants().SingleOrDefault(p
=> p.Name.LocalName == "payloadPublication");
            /// Detecting overallStartTime for Event that occured
            IEnumerable<XElement> startOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "startOfPeriod");
            /// Detectign overallEndTime for Event that occured
            IEnumerable<XElement> endOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "endOfPeriod");
            /// Extracting comment for event
            IEnumerable<XElement> comment = payloadPublicationEvent.Descendants().Where(d
=> d.Name.LocalName == "comment");
            /// Detecting infor about event that occured (Weather, Roadworks, etc)
            IEnumerable<XElement> condition =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "situationRecord");
            /// Detecting object which contains points (locations)
            IEnumerable<XElement> locationContainedInGroup =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationContainedInGroup");

            IEnumerable<XElement> locationForDisplay =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationForDisplay");
            /// extracting accidents events from XML
            IEnumerable<XElement> accidents = condition.Where(x => x.FirstAttribute.Value
== "Accident");

            IEnumerable<XElement> comments = accidents.Descendants().Where(d =>
d.Name.LocalName == "comment");

            /// extracting weather conditions from XML
            IEnumerable<XElement> weather = condition.Where(x => x.FirstAttribute.Value
== "PoorEnvironmentConditions");

            IEnumerable<XElement> whecomm = weather.Descendants().Where(d =>
d.Name.LocalName == "comment");

            IEnumerable<XElement> pointCoordinatess =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "pointCoordinates");

            var EventDataTime = startOfPeriod.Zip(endOfPeriod, (s, e) => new {
startOfPeriod = s, endOfPeriod = e });
```

```
            var EventDataCom = EventDataTime.Zip(comment, (e, c) => new { EventDataTime =
e, comment = c });
            var EventDataLoc = EventDataCom.Zip(locationContainedInGroup, (e, l) => new {
EventDataCom = e, locationContainedInGroup = l });
            var eventData = EventDataLoc.Zip(condition, (e, c) => new { EventDataLoc = e,
condition = c });
            var ev = eventData.Zip(locationForDisplay, (e, l) => new { eventData = e,
locationForDisplay = l });
            var eve = ev.Zip(pointCoordinatess, (e, p) => new { ev = e, pointCoordinatess
= p });

            /// Accident object list
            var acc = accidents.Zip(locationForDisplay, (a, l) => new { accidents = a,
locationForDisplay = l });
            var acci = acc.Zip(comments, (a, c) => new { acc = a, comments = c });

            /// Weather object list
            var wea = weather.Zip(whecomm, (w, wh) => new { weather = w, whecomm = wh });


            #endregion

            XNode tempNode;

            /// parsing accident events into objects
            foreach (var item in acci)
            {
                DisplayEventUnplannedObject EventUnlpannedObject = new
DisplayEventUnplannedObject();

                if (item.acc.locationForDisplay.Name.LocalName == "locationForDisplay")
                {
                    foreach (var node in item.acc.accidents.Descendants())
                    {
                        if (node.Name.LocalName == "values")
                        {
                            EventUnlpannedObject.Comment = node.Value;
                        }

                        if (node.Name.LocalName == "latitude")
                        {
                            var PointLatitude = node.FirstNode;
                            tempNode = node.NextNode;
                            EventUnlpannedObject.Latitude =
double.Parse(((System.Xml.Linq.XText)PointLatitude).Value, CultureInfo.InvariantCulture);
                            EventUnlpannedObject.Longitude =
double.Parse(((System.Xml.Linq.XElement)tempNode).Value, CultureInfo.InvariantCulture);
                            EventUnplannedObjectList.Add(EventUnlpannedObject);
                        }
                    }
                }
            }
        }

        public void FillEventWeatherObjectList()
        {
            #region Loading Event data

            /// Loading XML document from local folder through StreamReader
            var xDocEvents = XDocument.Load(new
StreamReader("C:/Users/sc2015/Desktop/podaci/Event_Data_-_Full_Refresh.xml"));

            /// Detecting payloadPublication descendant of d2LogicalModel via Linq
```

```
            XElement payloadPublicationEvent = xDocEvents.Descendants().SingleOrDefault(p
=> p.Name.LocalName == "payloadPublication");
            /// Detecting overallStartTime for Event that occured
            IEnumerable<XElement> startOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "startOfPeriod");
            /// Detectign overallEndTime for Event that occured
            IEnumerable<XElement> endOfPeriod =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "endOfPeriod");
            /// Extracting comment for event
            IEnumerable<XElement> comment = payloadPublicationEvent.Descendants().Where(d
=> d.Name.LocalName == "comment");
            /// Detecting infor about event that occured (Weather, Roadworks, etc)
            IEnumerable<XElement> condition =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "situationRecord");
            /// Detecting object which contains points (locations)
            IEnumerable<XElement> locationContainedInGroup =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationContainedInGroup");

            IEnumerable<XElement> locationForDisplay =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName ==
"locationForDisplay");
            /// extracting accidents events from XML
            IEnumerable<XElement> accidents = condition.Where(x => x.FirstAttribute.Value
== "Accident");

            IEnumerable<XElement> comments = accidents.Descendants().Where(d =>
d.Name.LocalName == "comment");

            /// extracting weather conditions from XML
            IEnumerable<XElement> weather = condition.Where(x => x.FirstAttribute.Value
== "PoorEnvironmentConditions");

            IEnumerable<XElement> whecomm = weather.Descendants().Where(d =>
d.Name.LocalName == "comment");

            IEnumerable<XElement> pointCoordinatess =
payloadPublicationEvent.Descendants().Where(d => d.Name.LocalName == "pointCoordinates");

            var EventDataTime = startOfPeriod.Zip(endOfPeriod, (s, e) => new {
startOfPeriod = s, endOfPeriod = e });
            var EventDataCom = EventDataTime.Zip(comment, (e, c) => new { EventDataTime =
e, comment = c });
            var EventDataLoc = EventDataCom.Zip(locationContainedInGroup, (e, l) => new {
EventDataCom = e, locationContainedInGroup = l });
            var eventData = EventDataLoc.Zip(condition, (e, c) => new { EventDataLoc = e,
condition = c });
            var ev = eventData.Zip(locationForDisplay, (e, l) => new { eventData = e,
locationForDisplay = l });
            var eve = ev.Zip(pointCoordinatess, (e, p) => new { ev = e, pointCoordinatess
= p });

            /// Accident object list
            var acc = accidents.Zip(locationForDisplay, (a, l) => new { accidents = a,
locationForDisplay = l });
            var acci = acc.Zip(comments, (a, c) => new { acc = a, comments = c });

            /// Weather object list
            var wea = weather.Zip(whecomm, (w, wh) => new { weather = w, whecomm = wh });


            #endregion
            XNode tempNode;
```

```
            /// parsing weather conditions events into objects
            foreach (var item in wea)
            {
                DisplayEventPolygonObject EventPolygonObject = new
DisplayEventPolygonObject();

                foreach (var node in item.whecomm.Descendants())
                {
                    if (node.Name.LocalName == "values")
                    {
                        EventPolygonObject.Comment = node.Value;
                    }

                }

                foreach (var node2 in item.weather.Descendants())
                {
                    if (node2.Name.LocalName == "latitude")
                    {
                        var PointLatitude = node2.FirstNode;
                        tempNode = node2.NextNode;
                        Point point = new
Point(double.Parse(((System.Xml.Linq.XElement)tempNode).Value,
CultureInfo.InvariantCulture), double.Parse(((System.Xml.Linq.XText)PointLatitude).Value,
CultureInfo.InvariantCulture));

                        EventPolygonObject.listPoint.Add(point);
                    }
                }

                EventPolygonObjectList.Add(EventPolygonObject);
            }

        }

        /// <summary>
        /// Fills the provided dataset with map features from the Event_Data_-
_Full_Refresh.xml file.
        /// </summary>
        /// <param name="FillDataSet">The dataset to fill with map features.</param>
        public void FillFlowPointDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < PointObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
PointGeometry(PointObjectList[i].Longitude, PointObjectList[i].Latidute),
Crs.Wgs84LongLat);

                //adding info about speed and vechileFlow on map
                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue("Current" + "\n" + "Traffic flow: " + PointObjectList[i].VehicleFlow)));

                dataSet.Insert(feature);

                if (i % 50 == 0)
                    resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillFlowPointHistoricalDataSet(MemoryDataSet dataSet)
```

```
        {
            for (int i = 0; i < HistoricalPointObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
PointGeometry(HistoricalPointObjectList[i].Longitude,
HistoricalPointObjectList[i].Latidute), Crs.Wgs84LongLat);

                //adding info about speed and vechileFlow on map
                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue("07.06.2015." + "\n" + "Traffic flow: " +
HistoricalPointObjectList[i].VehicleFlow)));

                dataSet.Insert(feature);

                if (i % 50 == 0)
                    resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillFlowLineDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < LineObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
LineGeometry(LineObjectList[i].listPoint), Crs.Wgs84LongLat);

                double koef = 0.002 * double.Parse(LineObjectList[i].VehicleFlow,
CultureInfo.InvariantCulture);

                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("WIDTH",
new AttributeValue(koef)));

                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(LineObjectList[i].VehicleFlow)));

                dataSet.Insert(feature);

                if (i % 5000 == 0)
                    resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillSpeedPointDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < PointObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
PointGeometry(PointObjectList[i].Longitude, PointObjectList[i].Latidute),
Crs.Wgs84LongLat);

                //adding info about speed and vechileFlow on map
                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue("Current" + "\n" + "Traffic speed: " + PointObjectList[i].Speed)));

                dataSet.Insert(feature);

                if (i % 50 == 0)
```

```
                resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillSpeedPointHistoricalDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < HistoricalPointObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
PointGeometry(HistoricalPointObjectList[i].Longitude,
HistoricalPointObjectList[i].Latidute), Crs.Wgs84LongLat);

                //adding info about speed and vechileFlow on map
                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue("07.06.2015." + "\n" + "Traffic speed: " +
HistoricalPointObjectList[i].Speed)));

                dataSet.Insert(feature);

                if (i % 50 == 0)
                    resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillSpeedLineDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < LineObjectList.Count; ++i)
            {
                // Create a feature with a point geometry at the position given by the
long/lat values.
                Feature feature = new Feature(new
LineGeometry(LineObjectList[i].listPoint), Crs.Wgs84LongLat);

                double value = double.Parse(LineObjectList[i].Speed,
CultureInfo.InvariantCulture);


                if (value < 50)
                    feature.Attributes.Add(new KeyValuePair<Atom,
AttributeValue>("COLOR", new AttributeValue("Red")));
                else if (value >= 50 && value < 80)
                    feature.Attributes.Add(new KeyValuePair<Atom,
AttributeValue>("COLOR", new AttributeValue("Yellow")));
                else if (value >= 80)
                    feature.Attributes.Add(new KeyValuePair<Atom,
AttributeValue>("COLOR", new AttributeValue("Green")));


                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(LineObjectList[i].Speed)));

                dataSet.Insert(feature);

                if (i % 5000 == 0)
                    resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }

        public void FillEventRoadworksDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < EventRoadworksObjectList.Count; ++i)
```

```
            {
                Feature feature = new Feature(new
PointGeometry(EventRoadworksObjectList[i].Longitude,
EventRoadworksObjectList[i].Latitude), Crs.Wgs84LongLat);

                int length = EventRoadworksObjectList[i].StartOfPeriod.Length - 9;

                string lines = string.Join(Environment.NewLine,
EventRoadworksObjectList[i].Comment.Split()
                    .Select((word, index) => new { word, index })
                    .GroupBy(x => x.index / 9)
                    .Select(grp => string.Join(" ", grp.Select(x => x.word))));

                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(EventRoadworksObjectList[i].Condition + "\n\n" + lines)));

                dataSet.Insert(feature);
            }
            resQMapModel.CarmentaEngine.MapControl.View.Update();
        }
        public void FillEventUnplannedDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < EventUnplannedObjectList.Count; ++i)
            {
                Feature feature = new Feature(new
PointGeometry(EventUnplannedObjectList[i].Longitude,
EventUnplannedObjectList[i].Latitude), Crs.Wgs84LongLat);

                string lines = string.Join(Environment.NewLine,
EventUnplannedObjectList[i].Comment.Split()
                    .Select((word, index) => new { word, index })
                    .GroupBy(x => x.index / 9)
                    .Select(grp => string.Join(" ", grp.Select(x => x.word))));

                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(lines)));

                dataSet.Insert(feature);
            }
            resQMapModel.CarmentaEngine.MapControl.View.Update();
        }
        public void FillEventPolygonDataSet(MemoryDataSet dataSet)
        {
            for (int i = 0; i < EventPolygonObjectList.Count; ++i)
            {
                Feature feature = new Feature(new
PolygonGeometry(EventPolygonObjectList[i].listPoint), Crs.Wgs84LongLat);

                string lines = string.Join(Environment.NewLine,
EventPolygonObjectList[i].Comment.Split()
                    .Select((word, index) => new { word, index })
                    .GroupBy(x => x.index / 9)
                    .Select(grp => string.Join(" ", grp.Select(x => x.word))));

                feature.Attributes.Add(new KeyValuePair<Atom, AttributeValue>("INFO", new
AttributeValue(lines)));

                dataSet.Insert(feature);
            }
            resQMapModel.CarmentaEngine.MapControl.View.Update();
        }
    }
}
```

**Prilog 3.** Programski kod datoteke "DisplayEventPointData.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Carmenta.Engine;

namespace SensorPlugin
{
    public class DisplayEventPointData
    {
        //defining properties
        double latitude;
        double longitude;
        string startOfPeriod;
        string endOfPeriod;
        string condition;
        string comment;

        public List<Point> listPoint = new List<Point>();

        public double Latitude
        {
            get { return latitude; }
            set { latitude = value; }
        }

        public double Longitude
        {
            get { return longitude; }
            set { longitude = value; }
        }

        public string StartOfPeriod
        {
            get { return startOfPeriod; }
            set { startOfPeriod = value; }
        }

        public string EndOfPeriod
        {
            get { return endOfPeriod; }
            set { endOfPeriod = value; }
        }

        public string Condition
        {
            get { return condition;  }
            set { condition = value; }
        }

        public string Comment
        {
            get { return comment; }
            set { comment = value; }
        }
    }
}
```

**Prilog 4.** Programski kod datoteke "DisplayEventPolygonObject.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Carmenta.Engine;

namespace SensorPlugin
{
    public class DisplayEventPolygonObject
    {
        private double latitude;
        private double longitude;
        private string startOfPeriod;
        private string endOfPeriod;
        private string comment;
        string condition;
        public List<Point> listPoint = new List<Point>();

        public double Latitude
        {
            get { return latitude; }
            set { latitude = value; }
        }

        public double Longitude
        {
            get { return longitude; }
            set { longitude = value; }
        }

        public string StartOfPeriod
        {
            get { return startOfPeriod; }
            set { startOfPeriod = value; }
        }

        public string EndOfPeriod
        {
            get { return endOfPeriod; }
            set { endOfPeriod = value; }
        }

        public string Comment
        {
            get { return comment; }
            set { comment = value; }
        }

        public string Condition
        {
            get { return condition; }
            set { condition = value; }
        }
    }
}
```

**Prilog 5.** Programski kod datoteke "DisplayEventPolygonObject.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Carmenta.Engine;

namespace SensorPlugin
{
    public class DisplayEventPolygonObject
    {
        private double latitude;
        private double longitude;
        private string startOfPeriod;
        private string endOfPeriod;
        private string comment;
        string condition;
        public List<Point> listPoint = new List<Point>();

        public double Latitude
        {
            get { return latitude; }
            set { latitude = value; }
        }

        public double Longitude
        {
            get { return longitude; }
            set { longitude = value; }
        }

        public string StartOfPeriod
        {
            get { return startOfPeriod; }
            set { startOfPeriod = value; }
        }

        public string EndOfPeriod
        {
            get { return endOfPeriod; }
            set { endOfPeriod = value; }
        }

        public string Comment
        {
            get { return comment; }
            set { comment = value; }
        }

        public string Condition
        {
            get { return condition; }
            set { condition = value; }
        }
    }
}
```

**Prilog 6.** Programski kod datoteke "DisplayEventUnplannedObject.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SensorPlugin
{
    public class DisplayEventUnplannedObject
    {
        private double latitude;
        private double longitude;
        private string comment;

        public double Latitude
        {
            get {return latitude;}
            set { latitude = value; }
        }

        public double Longitude
        {
            get { return longitude; }
            set { longitude = value; }
        }

        public string Comment
        {
            get { return comment; }
            set { comment = value; }
        }
    }
}
```

**Prilog 7.** Programski kod datoteke “DisplayLineObject.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Carmenta.Engine;

namespace SensorPlugin
{
    public class DisplayLineObject
    {

        private string speed;
        private string vehicleFlow;
        public List<Point> listPoint = new List<Point>();


        public string Speed
        {
            get { return speed; }
            set { speed = value; }
        }

        public string VehicleFlow
        {
            get { return vehicleFlow; }
            set { vehicleFlow = value; }
        }
    }
}
```

**Prilog 8.** Programski kod datoteke "DisplayPointObject.cs"

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace SensorPlugin
{
    public class DisplayPointObject
    {
        //defining properties
        private string speed;
        private string vehicleFlow;
        private double longitude;
        private double latitude;

        public string Speed
        {
            get { return speed; }
            set { speed = value; }
        }

        public string VehicleFlow
        {
            get { return vehicleFlow; }
            set { vehicleFlow = value; }
        }

        public double Longitude
        {
            get { return longitude; }
            set { longitude = value; }
        }

        public double Latidute
        {
            get { return latitude; }
            set { latitude = value; }
        }
    }
}
```

**Prilog 9.** Programski kod datoteke “Sensor.Control.cs”

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Carmenta.Engine;

namespace SensorPlugin
{
    public partial class SensorControl : UserControl
    {
        private readonly SensorPanel panel;

        public SensorControl(SensorPanel panel)
        {
            InitializeComponent();
            this.panel = panel;

            comboBoxType.DisplayMember = "Text";
            comboBoxType.ValueMember = "Value";

            var items = new[] {
                new { Text = "Traffic flow", Value = "flow" },
                new { Text = "Traffic speed", Value = "speed" }
            };

            comboBoxType.DataSource = items;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            panel1.Visible = (
                panel1.Visible == true ? false : true
                 );
        }

        private void button2_Click(object sender, EventArgs e)
        {
            panel2.Visible = (
               panel2.Visible == true ? false : true
                );
        }

        private void checkBoxFlowPoint_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxFlowPoint.Checked == true)
                panel.FillFlowPointDataSet(panel.dataSetFlowPoint);

            else if (checkBoxFlowPoint.Checked == false)
            {
                panel.dataSetFlowPoint.Clear();
                panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
            }

        }
```

```
private void checkBoxFlowLine_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxFlowLine.Checked == true)
        panel.FillFlowLineDataSet(panel.dataSetFlowLine);
    else if (checkBoxFlowLine.Checked == false)
    {
        panel.dataSetFlowLine.Clear();
        panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
    }
}

private void checkBoxSpeedPoint_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxSpeedPoint.Checked == true)
        panel.FillSpeedPointDataSet(panel.dataSetSpeedPoint);

    else if (checkBoxSpeedPoint.Checked == false)
    {
        panel.dataSetSpeedPoint.Clear();
        panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
    }

}

private void checkBoxSpeedLine_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxSpeedLine.Checked == true)
        panel.FillSpeedLineDataSet(panel.dataSetSpeedLine);
    else if (checkBoxSpeedLine.Checked == false)
    {
        panel.dataSetSpeedLine.Clear();
        panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
    }
}

private void checkBoxEventsRoadWorks_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxEventsRoadWorks.Checked == true)
        panel.FillEventRoadworksDataSet(panel.dataSetEventRoadworks);

    else if (checkBoxSpeedLine.Checked == false)
    {
        panel.dataSetEventRoadworks.Clear();
        panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
    }
}

private void checkBoxEventsUnplanned_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxEventsUnplanned.Checked == true)
            panel.FillEventUnplannedDataSet(panel.dataSetEventUnplanned);

    else if (checkBoxEventsUnplanned.Checked == false)
    {
        panel.dataSetEventUnplanned.Clear();
        panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
    }
}

private void button3_Click(object sender, EventArgs e)
{
    string type = comboBoxType.SelectedValue.ToString();
```

```
            if(type.Equals("flow"))
                panel.FillFlowPointHistoricalDataSet(panel.dataSetFlowPointHistorical);
            else if (type.Equals("speed"))
                panel.FillSpeedPointHistoricalDataSet(panel.dataSetSpeedPointHistorical);

        }

        private void buttonClear_Click(object sender, EventArgs e)
        {
            panel.dataSetFlowPointHistorical.Clear();
            panel.dataSetSpeedPointHistorical.Clear();
            panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
        }

        private void checkBoxMeteo_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxMeteo.Checked == true)
                panel.FillEventPolygonDataSet(panel.dataSetEventPolygon);
            else if (checkBoxMeteo.Checked == false)
            {
                panel.dataSetEventPolygon.Clear();
                panel.resQMapModel.CarmentaEngine.MapControl.View.Update();
            }
        }



    }
}
```