Czech Technical University in Prague
Faculty of Transportation Sciences

MASTER THESIS

Complex Analysis of Traffic Information

Amsterdam, 2008 **Filip Křikava**

Insert original assessment

Declaration of Authorship

I hereby confirm that I have authored this thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other sources, are marked as such.

In Amsterdam ______________ ____________________

signature

Abstract

Analysis of traffic information is an essential part of creating and operating any traffic information system that provides its customers with accurate and timely data about the traffic status of a road network. As the amount of data generated by these systems is vast, there is an urgent need to provide system developers, traffic engineers and analysts with a tool that would allow them to easily visualize and observe collected traffic data.

This thesis presents such an information system which is used for traffic data analysis and visualization at TomTom International B.V. Common techniques for traffic data analysis are summarized and an outline of the complete architecture and implementation of the system is provided. The proposed architecture is based on the service oriented architecture paradigm, which in combination with stateless services resulted in a highly extensible and scalable system. The implemented system has been deployed into the production environment and is intensively used for daily operations to analyze the TomTom High Definition traffic data chain. It has proven itself to be an effective tool and has helped to improve the overall quality of the data.

Abstrakt

Analýza dopravních infromací je důležitou součástí vývoje a provozu každého dopravního informačního systému, který má poskytovat svým uživatelům přesné a aktuální informace. Jelikož množství generovaných dat je velké, je potřeba poskytnout systémovým vývojářům, dopravním inženýrům a analytikům nástroj, který by jim umožnil jednoduše sledovat a vizualizovat shromážděné dopravní informace.

V této práci je prezentován informační systém, který je používán pro analýzu a vizualizaci dopravních informací ve firmě TomTom International B.V. Jsou zde shrnuty základní metody používané pro analýzu dopravních dat spolu s kompletní architekturou a implementací tohoto systému. Navržená architektura je postavená na paradigmatu SOA - service oriented architecture, což v kombinaci s bezstavovými službami dalo vzniknout vysoce rozšiřitelnému a škálovatelnému systému, který byl úspěšně nasazen a je denně používán pro analýzu dopravních dat. Aplikace se ukázala být efektivním nástrojem pro analýzu dopravních informací a pomohla zlepšit celkovou kvalitu dat.

Acknowledgement

During my work on the project presented in this thesis, I have received help from many. First of all, I thank my supervisor prof. Ing. Petr Moos, CSc. and TomTom International, B.V. for allowing me to combine the graduation project with my work. Many colleagues have provided me with very valuable advices, comments and discussions. I would like to thank especially Felix Faassen, Ida de Groot, Peter Krootjes, Stefan Lorkowski, Andrew Phillips, Rob Schuurbiers and Bob Soeters.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The rapid growth of interest in personal navigation in the last few years has changed the industry. With the technological evolution of navigation products, the main goal is no longer just the static determination of a route from one location to another. Instead, they have changed into products and services that provide dynamic optimal routing to selected destinations with time accurate travel information. This could never be possible without a system providing precise information about the current traffic status in a road network. Real-time traffic information, broadcast directly to a user, is the current state-of-the-art in personal navigation including flexible traffic signs like Variable Message Signs (VMS), internet route-planing services like Google Maps, the media and, recently, a growing number[1] of Personal Navigation Devices.

In general, there is an ongoing move to use more and more technology to provide better, more accurate and more timely information in order for drivers to be able to make informed decisions before and during their trip. This also contributes to other aspects of transport as it makes drivers more flexible when planning, less stressed en route and, in the end, possibly even makes their trips safer.[2]

One of the example of such a solution today is TomTom HD Traffic (HDT); a Traffic Information Provider (TIP) that is operating in five countries,[3] where it is generating traffic information for all primary and most of the secondary roads and is fully integrated with TomTom personal navigation device (PND).

Generally, TIPs are systems gathering information about current traffic flow, processing them and publishing them to consumers. They operate in road networks which are highly

[1]About 40 million units in Europe and US (source Navteq report July-August 2008).

[2]There is as yet no particular research on beneficial effects of real-time traffic information on user behavior, stress level and trip safety, but based on [29], we believe it is safe to predict that such an effect exists.

[3]At the time of writing this text.

dynamic, with millions of people using them every day, country wide. In such networks, traffic flow conditions change all the time. In order to handle this load, traffic providers are usually complex distributed systems with large scale deployment. They work with huge amounts of data and constantly generate large datasets of traffic information. This information is usually used in real-time in order to provide smart routing. The main goal is to inform users about congestion before they get stuck in it. This means providing information instantly as soon as flow changes.[4] Alongside the requirement for speed, it is critically important that TIPs are accurate, especially in conjunction with the aforementioned smart routing. With inaccurate information about traffic congestion, the user can be guided to a destination in a very inefficient way, creating a significant delay to his or her travel. TIPs, like other mission critical applications, require constant monitoring and analysis for quality assurance.

In large systems producing vast amount of complex data, it is difficult to get a good overview of the overall quality and performance of the system. In such a case, we normally do not look at the raw output that is being produced; instead, we need to use a tool which allows us to look at the data in some sort of well arranged way to provide more meaningful insight, for instance through visual projection. Since traffic information is connected to certain locations, the need for visual representation (in a geographical context) is all the greater. Such a tool is easily capable of producing an overview of traffic situations in a certain area of interest, and can also be used as a debugging tool, allowing in-depth inspection of the entire process, in order to analyze possibly incorrect behavior. It is also valuable during development of the traffic information provider itself, and can help to quickly verify results of experiments with different system tuning parameters, etc.

Although there are a few applications addressing data analysis and visualization in general that could be considered for analysis purposes, their cost and steep learning curves are rather significant. Traffic information analysis, like any other data analysis task, has its own specific needs, and there exists no system that could be used in TomTom context directly without serious adjustments and customizations.[5] These observations led to the decision to create a new application tailored to support all specific use cases needed for complex traffic information analysis. The following sections will present review of these requirements.

[4]There is some delay, because the system needs to wait a bit before deciding that an event is indeed congestion, for example.

[5]We will present an overview of systems considered in section 4.

1.2 Goal and Vision

The goal of this work is to design and create an information system for traffic data analysis and visualization that fulfils all requirements for performing analysis of the TomTom HD Traffic data chain, yet with extensibility to other potential content in mind. We will investigate how an application can be designed to be able to analyze large datasets in a visual form. Even though most of the remainder of this text is based on experience gained while working with TomTom HD Traffic, we will try to generalize to the problem of traffic information analysis in a larger domain. The system presented is meant to be used primarily for internal purposes. The main stakeholders are traffic analysts and developers. The system should assist these users in traffic data analysis primarily by data visualization; the actual analysis is done by traffic specialists. Their specific requirements will be covered in later sections.

There are three main points to be taken into consideration while outlining initial requirements for a system that operates with HDT in general:

- HDT is based on very recent research and uses some of the latest cutting-edge technology. Deployment on such a scale as HDT is somewhat unique and there is not yet much expertise in monitoring and analysis. For this reason, there might be some issues that have not (or could not) be foreseen and therefore requirements may change as we gain more experience.
- The amount of data generated by HDT is vast and as the coverage is increasing, this will continue to grow.
- Generated traffic data are being store in remote data-centers, thus bandwidth constraints need to be taken into consideration.

With this in mind, we decided on the following general design principles for the application:

- **Flexibility and Extensibility**. With regards to changeable requirements, the tool should provide an easy and quick way to add new functionality or extend the existing feature set. There might be a need for some special type of analysis that is not yet supported, for instance. Also, the system will be used for experimenting with data and for prototyping some new ideas that have not been formally specified. Therefore it should define an easy to use, yet powerful set of APIs that can be used directly from within the application and it could be used by more experienced user to quickly add new or extend current functionality.
- **Data Accessibility**. Traffic data are constantly being generated by numerous servers deployed in data-centers partitioned by geographical area. The system must

be able to provide access to all traffic data without having direct file access to the data-center.

- **Usability**. There will be different types of users interacting with the tool. They will have different skills and interests. The application should provide a flexible and easy to use user interface (UI), and have a look and feel consistent with other systems at TomTom, in order to provide a comparable user experience.
- **Testability**. Based on the output of this application, the quality and performance of HDT will be judged. It will affect the tuning of the traffic system, thus good test coverage verifying application correctness is a necessity. There are not many testing resources available for the development, consequently the design should make unit testing especially easy.[6]

In general, traffic data analysis operates with two primary data artifacts: traffic data and geographical data. These are, in general, immutable - traffic data, once generated, do not change,[7] and neither does the map.[8] This simplifies the system quite substantially, because there is no need for complicated data locking and synchronization.

As this tool is not considered a mission critical system,[9] we therefore do not impose any constrains regarding RAS-D (Reliability, Availability, Serviceability, and Durability).

The above mentioned list is just a basic sketch of general requirements to give the reader an idea of the scope of the system we want to build. Section 3.2 provides an organized review of functional and non-functional requirements for the application.

1.3 Organization of this Document

Section 2 of this document provides theoretical background on traffic flow theory, traffic data and traffic data analysis. It covers the main principles that will be the base of building the system. The next chapter 3 introduces the actual project for traffic data analysis that is goal of this work. It summarizes the environment which the system will be part of and presents a structured overview of requirements for the system. Chapter 4 presents the system architecture, with detailed description of its modules. In chapter 5, a brief overview of the implementation with an outline of the different technologies used is presented, together with some decisions that have been taken and issues that influenced

[6]Unit testing is a method of verifying that individual units of source code are working properly. A unit is the smallest testable part of an application.

[7]Here it is important to look at traffic data as a traffic flow property that was measured at particular location, at particular point in time and therefore is fixed. In chapter 3 we will show different aspect of traffic data which makes them mutable.

[8]There are map changes as infrastructure of roads changes, however, traffic data are always generated against certain map version and therefore we can consider map as immutable source of data.

[9]We consider mission critical systems to be systems that need to be running 24/7 and where even a short down time is considered to have a big impact on business.

them. Finally, chapter 6 summaries the work done and introduces perspectives for possible further development.

Chapter 2

Theoretical Background

2.1 Fundamentals of Traffic Flow Theory

This section presents an introduction to traffic flow theory with the aim of developing a basic theoretical framework with which to examine the characteristics of traffic flow. We will start by looking at traffic stream from a microscopic level. At this level, each vehicle is examined separately.

2.1.1 Time-Space diagram

Generally, when looking at some traffic operation we track the position of a *unit* along a guideway as a function of time.

Definition 2.1. A unit is an object capable of a standalone movement transport process. For example, a vehicle with a driver [31].

Let's use variable x to denote the distance travelled along the guideway from some arbitrary location and another variable t to denote the time elapsed from an arbitrary instant. We can define a tracking function $x(t)$ that returns location x for every t in the relevant range. Graphical representation of such a function in a (t, x) plane is a curve representing a trajectory of a traced unit. We will call a set of one or more of those trajectories plotted in this a plane *time-space diagram* [6].

Definition 2.2. Let's have a function $x(t)$ returning, for every given time t, a distance that a unit has travelled from some arbitrary location. A (t, x) plot of such functions is a *time-space diagram* ([6]).

An example of such a diagram is given in figure 2.1. It shows trajectories for two vehicles. Time-space diagrams are simple, yet very powerful and are one of the major tools used for traffic data analysis.

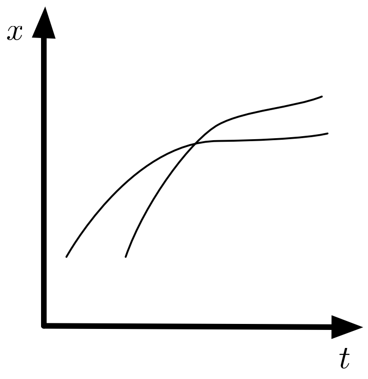


Figure 2.1: A Time-space diagram

In [6], different ways of constructing time-space diagrams are presented. Some applications might find it convenient to record the passage of units past stationary observers by looking at the immediate surroundings of some arbitrary location in a road network during time interval T, as shown in fig. 2.2a - *local measurement*. In this case, data are displayed as marks on horizontal lines at locations of the observers. We can also approximate vehicle trajectories (shown at dotted line) by connecting the dots for specific units. As can be seen in the figure, this is done by interpolation of the $x(t)$ between two consecutive discrete observing locations (in this case linear interpolation). The times between consecutive unit observations (h_i) in 2.2a is called *headways*

Another method (fig. 2.2b), which is actually the opposite of local measurement, is *instant measurement*. This time, we are acquiring data by taking snapshots[1] of a set of roads of length L over time. Each of these snapshots captures the traffic situation at a particular time t. In a diagram, this is represented by a vertical line for each instant, and the marks show either beginnings or ends of units in each sample. The distance between consecutive units at a given instant (s_i) as in 2.2b is referred to as *spacing*.

The last method, shown in fig. 2.2c (*moving measurement*), involves observers which are moving at some speed v while recording the times when units pass them. This is a generalized form of the former two because setting $v = 0$ gives us local and $v \to \infty$ instant measurement.

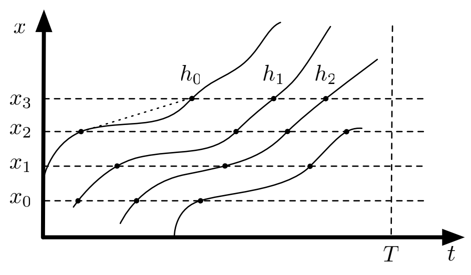
It does not matter in which way (t, x) diagram is constructed. Horizontal lines always identify times at which successive vehicles pass stationary observers and vertical lines identify locations of units at the given times. A time space diagram represents a complete description of the history of units' longitudinal motion.

Definition 2.3. *Headway is the time gap between two units in a traffic stream. It is defined as the difference in time between the moment the front of a unit arrives at a point*

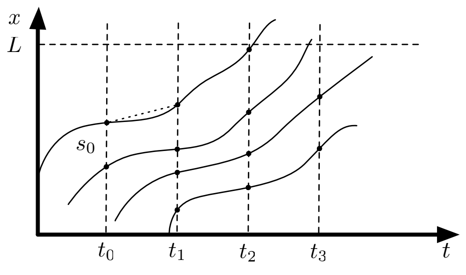
[1]aerial photographs, for example

and the moment the front of the following unit arrives at the same point ([5]).

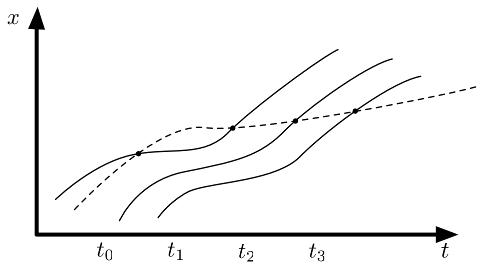
Definition 2.4. *Spacing* is the distance gap between two units in a traffic stream. It is defined as the distance from the front a unit to the front of the following unit ([5]).



(a) local observer at various locations



(b) instant observing at various instants



(c) moving observer (dashed)

Figure 2.2: Different ways of measurement

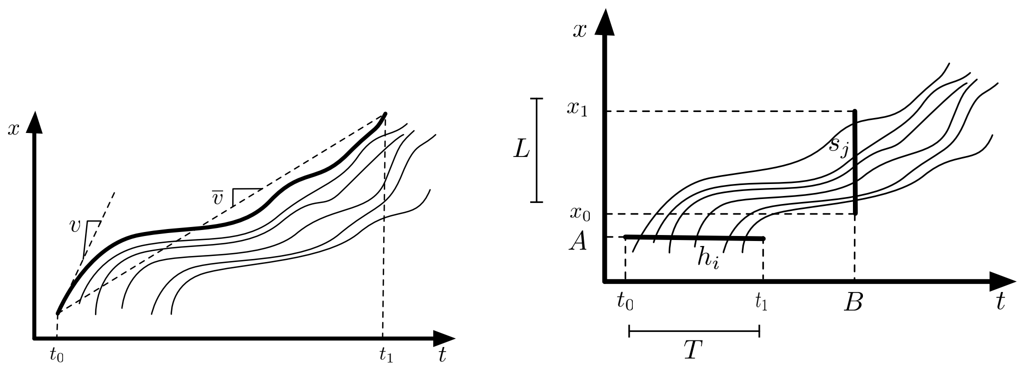
2.1.2 Traffic Flow Characteristics

The state of traffic flow can be represented by its state variables. In this section we will identify the most essential ones - *speed*, *flow* and *density* - using time-space diagrams. The definitions are based on [6].

Definition 2.5. *Speed* v is the rate of motion, or equivalently the rate of change in position, often expressed as distance x traveled per unit of time t. In fig. 2.3a we identify two speeds:

1. an average speed

$$\overline{v}(t_0, t_1) = \frac{x(t_1) - x(t_0)}{t_1 - t_0} \tag{2.1}$$



(a) Speed **(b)** Flow and density

Figure 2.3: Traffic flow characteristics in time-space diagram

2. an instantaneous speed

$$v(t) = \frac{dx(t)}{dt} = \\ = \lim_{\Delta t \to 0} \overline{v}(t, t + \Delta t) = \\ = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t} \tag{2.2}$$

Definition 2.6. *Flow* q is the number of units measured (using local measurement) during a given time, m, divided by the length of this interval, T.

$$q = \frac{m}{t_1 - t_0} = \frac{m}{T} \tag{2.3}$$

In fig. 2.3b we can see that at point A, stationary observer records $m = 6$ units during the interval $t_0 \leq t \leq t_1$, thus $q = 6/(t_1 - t_0) = 6/T$. For long observations, that include many units with comparable headways ($m, T \to \infty$), we can rewrite flow to:

$$q = \frac{m}{T} = \frac{m}{\sum_{i=1}^{m} h_i} = \frac{1}{\overline{h}} \tag{2.4}$$

Definition 2.7. *Density* ρ is the number of units measured (using instantaneous measurement), n, over a stretch of road of given length, L.

$$\rho = \frac{n}{x_1 - x_0} = \frac{n}{L} \tag{2.5}$$

With the same treatment as we used in (2.4) for flow and headways, we obtain an analogous relationship between density and average spacing:

$$\rho = \frac{n}{T} = \frac{n}{\sum_{j=1}^{n} s_j} = \frac{1}{\overline{s}} \tag{2.6}$$

Combining eq. (2.4) and (2.6), and looking at fig. 2.3 we conclude:

$$\begin{aligned} s &= hv \\ \overline{s} &= \overline{h}v \\ \frac{1}{\overline{s}} &= \frac{v}{\overline{h}} \end{aligned} \tag{2.7}$$

and therefore obtain the important relationship between state variables, sometimes called the *fundamental relation of traffic flow theory* [3]:

$$q = \frac{1}{\overline{s}} = \frac{v}{\overline{h}} = v\frac{1}{\overline{h}} \tag{2.8}$$

$$\boxed{q = v\rho} \tag{2.9}$$

2.1.3 Stationary Traffic

The previous section defined the fundamental relation between the state variables of traffic flow (2.9). Because of this equation, there are only two independent variables. In the following text we will focus on the relationship between those two variables (using [6, 16, 3]) and on the different phases of traffic flow. We will assume *stationary traffic* a situation when traffic does not change significantly along given road stretches for an extend period of time. This will simplify the notation, because the dependence on location and time no longer applies in this flow.

Definition 2.8. Stationary traffic is traffic where flow rates neither change along road stretches nor over time. Consequently, in time-space diagrams, all trajectories must be parallel and equidistant ([6, 3]). Fig. 2.4 shows an example of both situations.

The relationship between traffic state variables is often depicted in a two-dimensional plot known as a *fundamental diagram*. This diagram combines all possible homogeneous stationary traffic states in an equilibrium and represents it in a visual way [3]. It shows the relation between two of the three traffic state variables, as the third one can always be calculated using (2.9). There are numerous mathematical models for the fundamental diagrams ([13, 9, 30, 5]). One of the earliest studies of transport and traffic operations, Greenshields et al (1947), proposed a linear relationship between speed and density. Over the years, however, researches have suggested different forms, considering two separate phases for *free-flow* and *congested-flow*. In this thesis, we will present the same non-linear

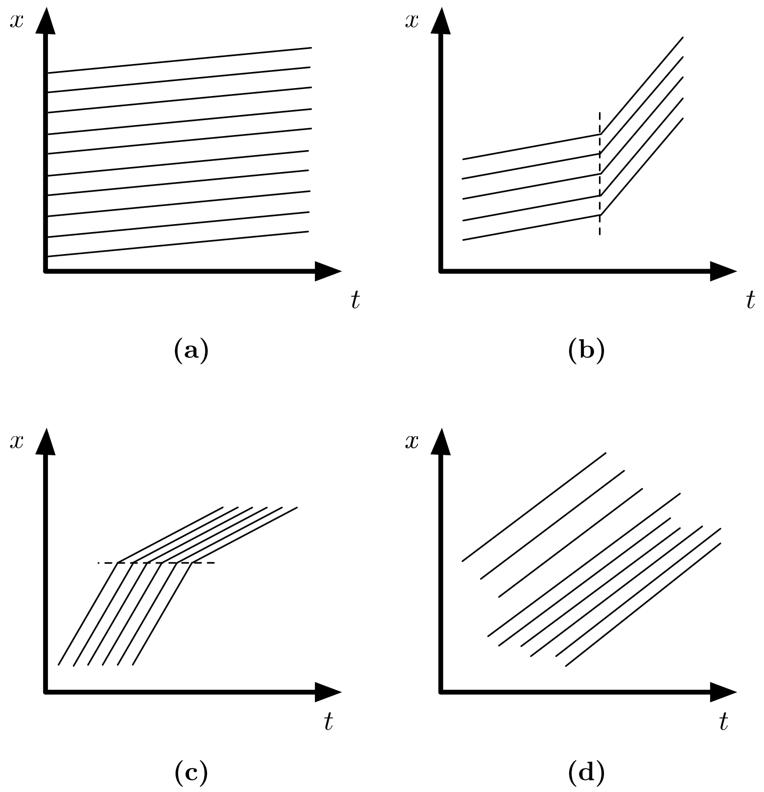


Figure 2.4: Examples of both stationary and non-stationary traffic situations: (a) has all lines parallel and equidistant to each other, thus it shows stationary traffic. By contrast, the rest present non-stationary traffic situations: in (b), at a given time all units increased their speeds; (c) shows decelerating traffic at certain location; and in (d) q and ρ increases after the middle unit ([6]).

model as in [6], since it is much more relevant in real world traffic situation than the original Greenshields.

The figure 2.5 shows a fundamental diagram: (a) displays the relation between speed and density; (b) between speed and flow and the last one (c) shows the relation between flow and density. In (a), the speed distribution is also shown as a color spectrum, from light green, indicating free moving traffic, through yellow to red, all the way to dark blue indicating a fully congested, stationary traffic stream.

From 2.5, we can also see that if the density on a road is increasing, but the distances between units are big enough, the travelling units do not affect each other (they are not in each other's way) and therefore the flow increases till it reaches its maximum q_0, when the derivative $\frac{dq}{d\rho}$ is 0. The associated speed at this point v_0 is called the *capacity speed*.[2] With increasing density, the distance between units is smaller, so they start affecting each other and this leads to a decrease in speed and increase in density. For very high densities

[2]Sometimes referred to as optimum or critical speed.

$\rho \to \rho_j$, it will lead to complete cessation of the stream $(q, v \to 0)$.

Based on this, we can identify three main states[3] of traffic flow: *free-flow*, *capacity flow* and *saturated flow.*

Definition 2.9. Free flowing traffic is a state of traffic at which units travel at *free-flow speed* v_f. Free flow speed is the absolute maximum possible[4] speed that is attained when flow and density approach 0 ([5]).[5]

Definition 2.10. Saturated (congested) traffic is a state of traffic, when speed and flow rate are down to 0. Units are queuing and when density reaches its maximum at ρ_j (*jam density*), the traffic stream stops moving ([5]).

Definition 2.11. Capacity (bounded) traffic is a traffic state around the maximum flow rate $q = q_0$ at which units are moving at speeds around v_0 ([5]).

2.1.4 Kinetics of Traffic Flow

In the previous section we have defined the essential properties of traffic flow and presented the important equation (2.9) as well as shown this relationship in the fundamental diagram (fig. 2.5) for stationary traffic. This section will briefly focus on real traffic that is neither homogeneous, because there are a variety of vehicle types and drivers' behaviors, nor stationary, because units' speed change continuously as drivers accelerate and decelerate to react to current traffic conditions (fig. 2.6).

We will introduce a modified notation for traffic flow state variables as a function in the $t - x$ plane: $v(x,t)$, $q(x,t)$, $\rho(x,t)$ and will discover their dynamic relation based on fluid dynamics, as known from classical physics [31].[6] We will assume that we are dealing with point variables and that they are uniquely defined at any moment and any location in the $t - x$ plane.
With the traffic stream state variables as $t - x$ functions, we will rewrite (2.9) as:

$$q(x,t) = v(x,t)\rho(x,t). \tag{2.10}$$

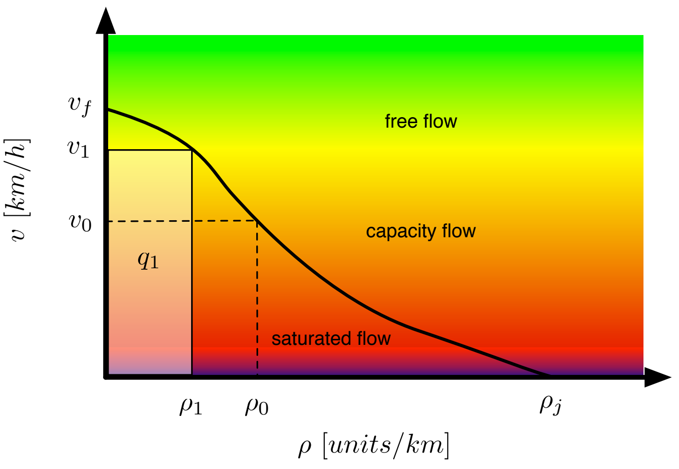
Now, we divide the road into $k \in \mathbf{N}$ cells, each of length Δx and time into $l \in \mathbf{N}$ intervals of duration Δt. Density ρ_i of a cell i, $i \in \{1, 2, \ldots, k\}$ at time t_j, $j \in \{1, 2, \ldots l\}$ is given by the function $\rho(i,j)$, and the number of units in such a cell is $n = \rho(i,j)\Delta x$. Example of two consecutive time intervals t_j and t_{j+1} is shown in fig. 2.7.

[3]The phases of traffic flow is together with the theory of transition between them discussed in detail in *three-phase-model* ([13]).

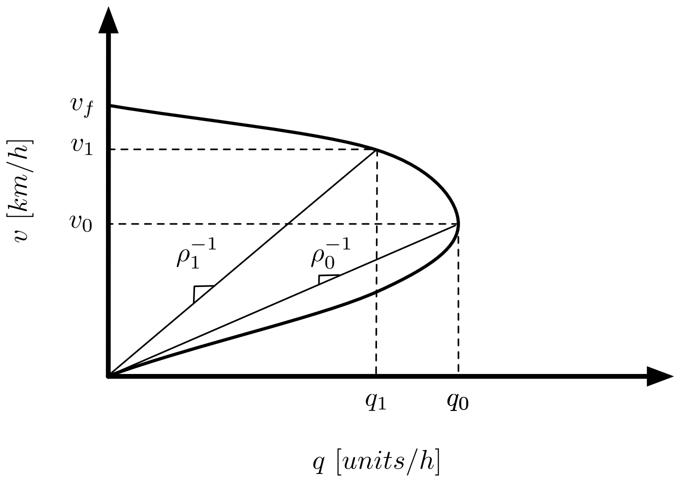
[4]usually legally constrained

[5]For example only one vehicle exists on a highway.

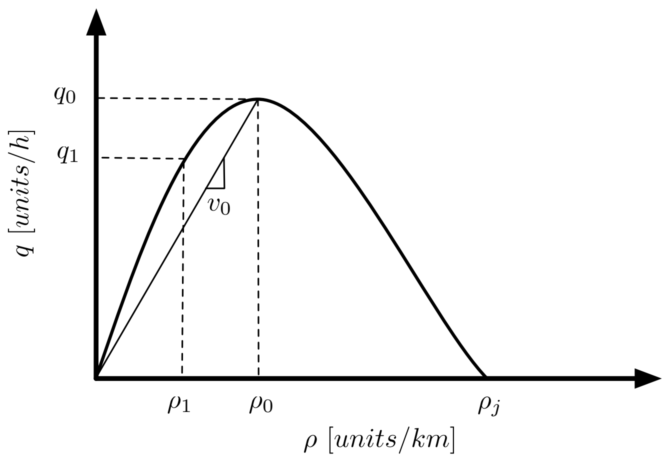
[6]It is important to note that traffic flow has strictly discrete character in contrast to fluid flow, which we understand as a continuous process.



(a) speed vs. density

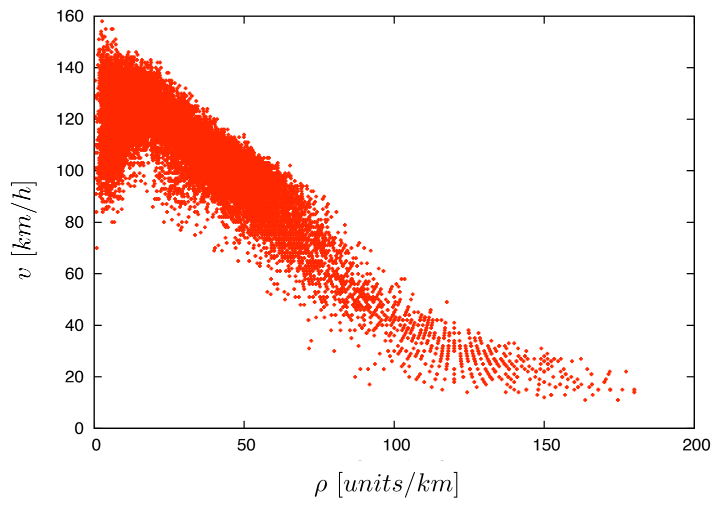


(b) speed vs. flow

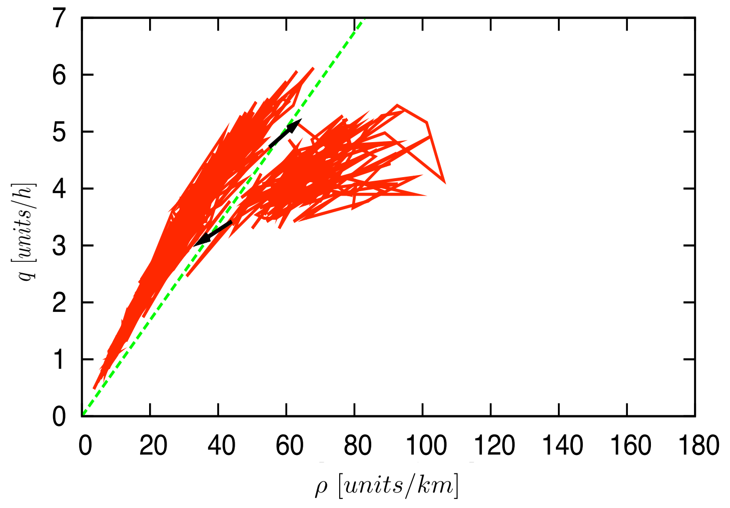


(c) flow vs. density

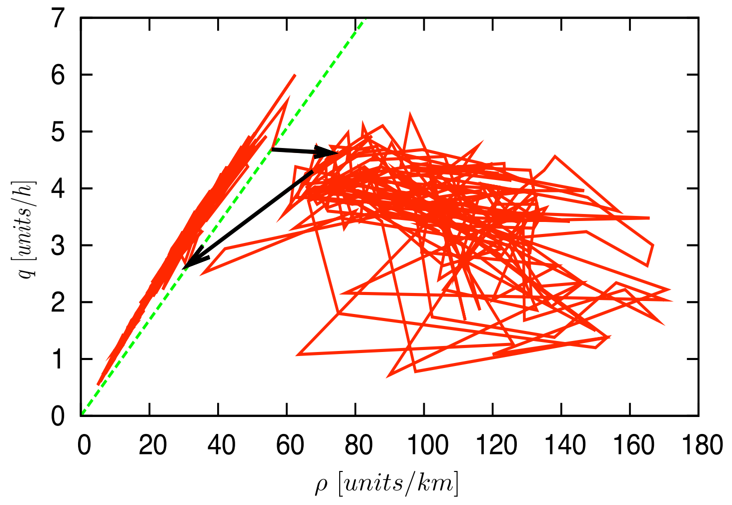
Figure 2.5: Fundamental diagram



(a) speed vs. density (A9)



(b) flow vs. density (A9)



(c) flow vs. density (A13)

Figure 2.6: Empirical fundamental diagram. In (b) and (c) green dashed lines separates capacity flow area on the left from free flow area on the right. [16]

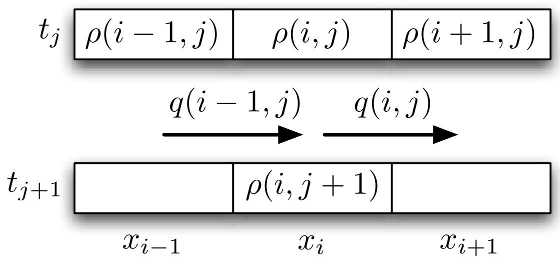


Figure 2.7: Density over time. The number of cars in cell at t_{j+1} is equal to the number of cars in cell at t_j and the number of cars entering minus the number of cars leaving the cell.

For very small Δx and Δt we can approximately write:

$$\begin{gathered}\rho(i,j+1)\Delta x = \rho(i,j)\Delta x + q(i-1,j)\Delta t - q(i,j)\Delta t \\ \frac{\rho(i,j+1)-\rho(i,j)}{\Delta t} + \frac{q(i,j)-q(i-1,j)}{\Delta x} = 0\end{gathered} \tag{2.11}$$

and for very small time intervals ($\Delta t \to 0$) and very small cell sizes ($\Delta x \to 0$) we obtain the partial differential equation representing the *conservation law of traffic flow*:

$$\frac{\partial\rho(x,t)}{\partial t} + \frac{\partial q(x,t)}{\partial x} = 0 \tag{2.12}$$

This equation is the same as the particle observation law of classical physics and it was applied to earliest traffic flow models (including Greenshields) in order to address unit conservation on a road. All traffic flow theories *must* always either satisfy equation (2.12) ([13]).

2.2 Traffic Data Chain

So far, we have given a brief introduction to traffic flow theory. We have identified the main properties of traffic flow - the properties that are the subject of analysis in this thesis. Now, with have a simple framework with which to observe and analyze traffic flow, we need to focus on the next problem, and that is the actual data acquisition from a traffic network.

In the introduction, we have touched on the concept of a Traffic Information Provider. It was introduced as a system that provides us with traffic data. For simplification, we will, throughout this thesis, consider traffic data as time-space measures of the traffic flow properties in a traffic network, as they were discussed in the section 2.1.2.

Generally, any data generation consists in principle of three phases: collection, processing and publishing. In the traffic domain, we will refer to these as the *traffic data chain*

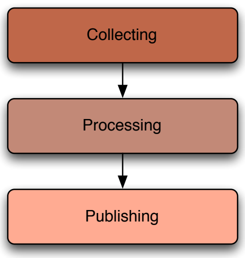


Figure 2.8: Traffic data chain

(fig. 2.8). A TIP can therefore be defined as a system that is implementing this data chain. One of the simplest examples of such a system is manual human traffic observation. In this case all three phases are performed by an individual. He or she will be observing traffic at an arbitrary location (2.2a) for a certain time period, logging all vehicle passages and possibly notifying someone in extreme cases like congestion. Industrial solutions, on the other hand, will more likely be telematics systems ([21]) consisting of interconnected subsystems for data gathering, processing and distribution.

The following subsections will describe the individual phases of the data chain. A concrete example will be presented in 3.1.

2.2.1 Collection

Traffic data collection is the essential part of the chain. More generally, it is an essential part of any traffic engineering, planning or operational activity. Without any data, non of these activities would be able to proceed.[7] This part of the traffic data chain is responsible for gathering time-space measurements of traffic flow properties during some time period in some traffic network. Usually, this is done at the microscopic level by tracking individual vehicles, essentially by looking at position changes of vehicles over time. These changes give us the necessary information about the distance travelled over time, thus giving us speed averages.

There are number of methods for vehicle tracking. They all use different principles and technologies and with many of them we are interacting on a daily basis while driving, but they all are based on methods outlined in 2.2.

In general, they can be divided into two major categories: infrastructure systems and floating car data.

[7]Besides looking at the problem, why there are no data.

Infrastructure Systems

Probably everyone has in some way been in contact with infrastructure systems for vehicle detection. Inductive loop detectors, radar and video cameras are all examples of systems that belong into this category. Even though they use different methods of detection, they all have in common the fact that they are based on *external* object detection. This means that they use an external sensor which is in whatever way detecting *other* objects around it. Generally,[8] we can think about these systems as road infrastructure accessories (2.2a). Since they are in some way part of the infrastructure network, they share its very high initial costs and difficulty of deployment. This results in relatively low coverage of the entire transportation network, usually only of the most important parts such as major motorways and complex junctions. In addition, in an urban environment there are many traffic interruptions, particularly at intersections. These interruptions cause delays that are not easily detected by measuring speeds at any specific point along the road [2].

Reliability varies among these systems. For example, inductive loops are usually very solid and robust systems, but also very expensive to build and difficult to install and maintain. Video detection systems tends to be sensitive to weather influences such as rain or fog.

Often, implementations of these systems have started as projects undertaken by governmental organizations[9] for traffic flow analysis, in order to identify bottlenecks and provide better support to drivers. Today, private parties[10] are also getting involved in the collection of traffic data using infrastructure systems. Further information about different implementations of infrastructure system for vehicle detection can be found in [14] and [21].

Floating Car Data

Floating Car Data (FCD) is an alternative approach that uses vehicles itself as the sensors in order to detect their locations in the network. As good as it sounds, it can only be done when cars are equipped with appropriate positioning devices that keep track of the vehicle's position as it *flows* in the network. The most widely used positioning devices today are based on Global Positioning System (GPS) technology. These devices can keep track of their position, but are not usually connected to the outside world, so the location data are not available in real time. GPS units are by nature only one-way communication devices (satellite-to-unit), thus additional systems are needed if location transmission is required.

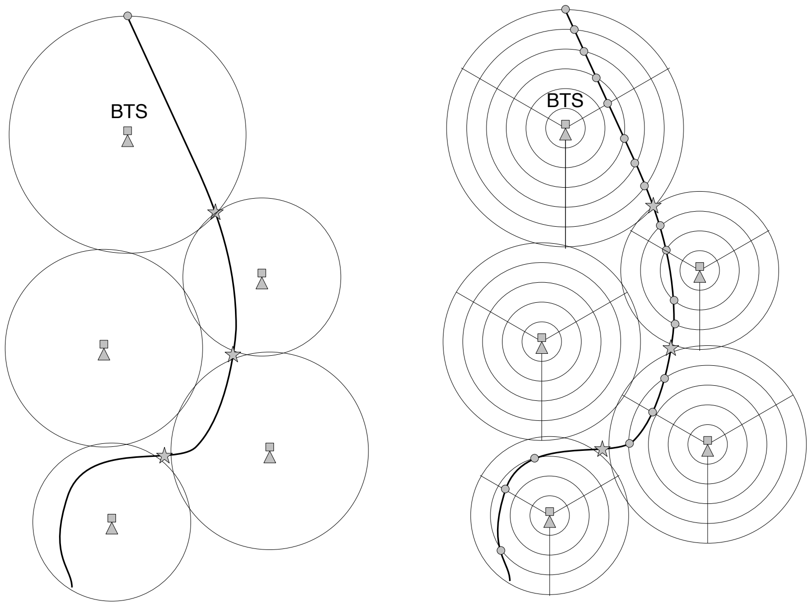
[8]An exception, for example, could be a human observer or a news reporting helicopter (a moving measurement 2.2c), which is not part of the road infrastructure, yet also undertaking external observation and therefore will also be considered as an infrastructure system.

[9]For example Verkeerscentrum Nederland (VCNL) van Rijkswaterstaat in the Netherlands.

[10]Traffic master in United Kingdom, DDG in Germany, INRIX in USA

There have been several projects using dedicated *floating cars* equipped with GPS devices for generating traffic data. For example, up-to-date travel time estimates for Berlin, Germany were obtained using 10,000,000 GPS observations from 300 taxis over five years [24]. However, there have not been any large scale deployments of such systems, primarily because of the need for additional equipment (for positioning, storage and/or transmission of logged data), thus additional cost.

A different option is to use mobile phones as positioning devices. Today, because of the high penetration of mobile phones, one or more are present in virtually every car during a journey ([37, 2]). These phones transmit their signal information to cellular networks; as the car moves (*floats*) in a road network, so does the signal. Using one of the cell phone location methods (i.e. by triangulating the signal), it is possible to track vehicles' locations over time (GSM network example 2.9). This enables us to use cellular phones as positioning devices and allows us to eliminate the cost of vehicle positioning equipment. The cost is still in there, but it is combined with additional functionality, thereby making it cheaper (from TIP perspective).



(a) using Hand-Over (HO) **(b)** using Timing-Advance (TA)

Figure 2.9: In (a), a GSM probe vehicle travels from a certain origin to a certain destination, following a specific route. As the vehicle is driving, its location and passage time is recorded at each hand-over (marked as star) event between two Base Transceiver Stations (BTS). The travel time is then calculated for each individual road segment within such a cell ([17]). Another approach (b) uses the timing-advance feature of GSM networks which compensates for the distance from the handset to the base station. This information can be used to narrow its location to a 550-m wide concentric band radiating from the BTS ([23]). Figure from ([16]).

This method has recently been getting a lot of attention. One of the reasons is that it does not require expensive infrastructure changes to the road network, as it leverages the existing infrastructure of cellular network providers. The main disadvantage of using cell phones is that we generally do not have the phone's location in terms of a known feature of a digital road map and therefore have to perform map matching, which is quite a complicated step [22]. The ideal option would be a combination of a GPS unit together with mobile phone. In this case we would have precise positioning on a digital map and an open data channel to the outside world. There are several such devices available nowadays, like some of the top level of Nokia cell phones, Apple's iPhone or the TomTom GO740 and GO940 PNDs.[11]

The performance of FCD varies depending on the concrete set of technologies used. In one of the field studies using mobile phones, it was shown that on motorways the relative error between the FCD and ground truth GPS travel times is below 15% for over 70%, even 90% of the time ([17]), but on urban and regional roads containing intersections or complex road layouts, the FCD system produces large variations in real experienced travel times. The latter might be improved by further filtering in the processing phase.

Also, much as with infrastructure road cameras, the use of mobile phone probes localization is experiencing social resistance for reasons of privacy. Even though such traces are anonymous, there is valid concern regarding abuse for surveillance purposes.

At the time of writing, one of the largest commercial systems based on FCD is TomTom HD Traffic.[12]

2.2.2 Processing

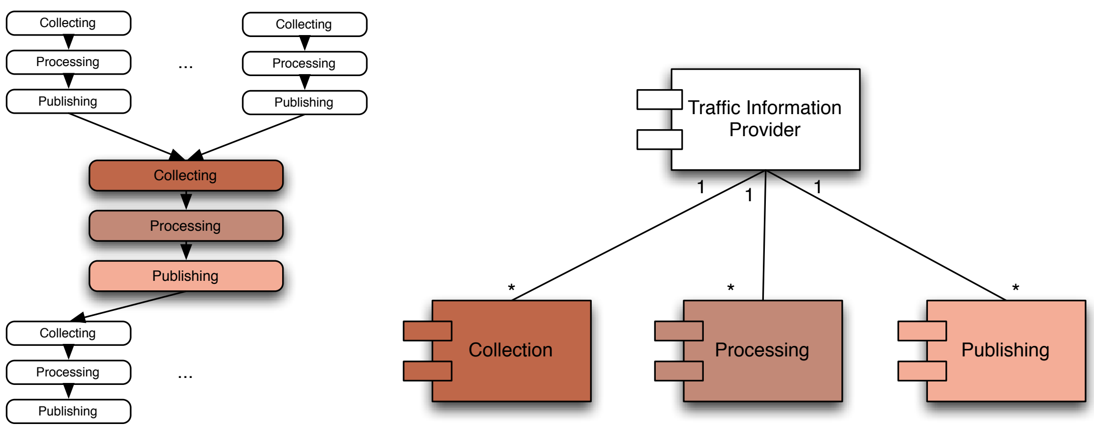
Collected data usually need some processing before they can be published. What kind of processing is necessary depends on the nature of collected data. Often, this phase also incorporates data enhancements, for example softening, gap filling, fusing, etc. For instance, if the purpose of a TIP is generating traffic information about congestion, such a system will, in processing phase, evaluate all incoming traffic data and, based on certain algorithms, will announce traffic incidents. Detailed information can be found in [16].

2.2.3 Publishing

The last phase of the traffic chain is the distribution of traffic data to the consumer. There are many different ways and protocols that are used for publishing traffic data, which depend on the kind of data prepared by the processing phase. In the case of ready-to-use traffic information, we can see published data on one side as traffic information shown

[11]These are not conventional mobile phones, but rather GPS navigation devices with a GSM modem for data communication.

[12]General information available at `http://www.tomtom.com/services/service.php?id=2`. An overview of the system is presented in 3.1.



(a) composition

(b) components

Figure 2.10: TIP composition and components

at roadside or information presented to the end-user in the media (traffic broadcasts on radio or television). On the other hand, when the data consists of more low-level metrics, they will very likely be dispatched to some intelligent system, which does some further processing, or storage or both, using some data protocol.

It is important to note that we can regard a TIP as a system composed of other TIPs. In other words, the data output of one TIP can be used as one of the inputs to another TIP (fig. 2.10a). TIPs can have multiple components responsible for different parts of the traffic chain (fig. 2.10b), for example multiple inputs for collection or different systems for publishing. Therefore, it can be defined as a system providing traffic information by orchestrating its components for data collection, processing and publishing.

It is very important to have access to data before and after each phase of the chain. This will allow us to easily identify which part is malfunctioning, as well as allowing us to inject different data for simulation or testing. Without the ability to feed the system with static or perhaps prefabricated data, it is very difficult to get the chain configuration "just right".

Traffic data analysis concerns all phases of the traffic data chain, including the data output at the very end. But even if the analysis covers the entire chain, it still does not guarantee correctness from the end user's perspective as it cannot verify any misinterpretation of the data by the final presentation system. This, however, is not in the scope of the TIP analysis.

2.3 Traffic Data Analysis

Traffic data analysis is a very new field and has not been developed as far as some other fields. Therefore, analysts still have to use some of the basic modeling skills and think on their own [6]. This is one of the reasons that makes analyzing traffic information very challenging. In the following text we will introduce a very basic set of instruments for traffic analysis. These tools are necessary for any meaningful discussion about traffic data. They will also build up the initial requirements for the data analysis application, which is the subject of this thesis.

To start with, it is important to define the scope of traffic data analysis. For the purpose of this text, we will refer to traffic data analysis as an instrument of transforming and visualizing traffic data with the goal of helping us to understand the underlying processes, suggest conclusions and support decision making. But first, we need to be able to visualize the data, in order to understand them, before we can draw any conclusions and make further decisions. Even though we are able to draw conclusions about data quality based on data analysis, we should not see it *only* as a tool for measuring data quality; rather, it should also help us to explain the behavior of the observed system.

We assume that traffic data are being analyzed for the following reasons:

- Problem investigation. In this case we know about some discrepancy between our traffic data and the real situation,[13] or we suspect that. In such a situation we often undertake an analysis in order to understand the system's behavior and to examine what went wrong. This also applies in the case when things went as expected, according to the system's settings, but are undesirable nonetheless for different reasons. These are some of the hardest problems to solve (a model that functions perfectly, yet gives undesirable results).
- Simulation. When performing system tuning, one is interested in how the change affected the resulting data in order to make suggestions about the tuning.
- Research. An interest in data visualization, usually with the goal of understanding the patterns and the behavior of the underlying traffic flow.

2.3.1 Traffic Data

Traffic data are the result of observation of traffic characteristics of a road network. Usually, when we are observing a function in an infrastructural network, we use some mathematical abstraction of such a network. In the case of a road network, we call this abstraction a *graph* ([25]). A simple *graph* G is a pair (V, E), where V is a finite set of nodes and

[13]Either reported directly (for example, a customer complaint) or as a result of comparison with another set of traffic data, taken from a different system.

$E \subseteq V \times V$ are the edges of the graph G. Edges can be seen as road stretches (segments) with different lengths. We will assume an oriented graph, where every segment has a unique identifier, a direction and an attribute that denotes its full geometry so, when drawn over a map image, it aligns with a particular road. Further, edges are weighted by a function $v_f : E \rightarrow \mathbf{R}$ which assigns to every edge a free flow speed (2.9) as defined by the road network. Often, we will refer to this road network abstraction simply as a map.

Most of the time we will use as the input for the analysis traffic data samples defined by two time-stamps T_0, T_1 with a sampling interval of Δt dividing the total time period into m similar cells $t_r,\ r \in \{1, 2, \ldots, m\}$ and route R as a set of n road stretches, $x_s \in E,\ s \in \{1, 2, \ldots, n\}$ of a total length $\sum_{s=1}^{n} |x_s| = |R|$, where $|x_s|, |R|$ denotes the length of the stretch x_s and route R respectively, followed by a function $v(t, x)$ that for each pair $(t_r$,$x_s)$ returns the speed that is the average of all measured speeds during the t_r interval on the x_s stretch or an empty value (a gap) representing no data recorded.[14]

This time-space division is a similar to the one shown in 2.7. Because the length of each road segment is known, we can easily substitute speed with transit time.

In real traffic data, and especially in case of data obtained from an FCD-based system, it is often the case that, at some time and for some road stretch, there are no data. There thus has to be a special procedure provided to handle gaps in the data. This procedure is called gap filling, for which there are many statistical techniques. One of the easiest methods is to use the free-flow value v_f for segments containing no data. This is very easy to implement, but introduces a certain inaccuracy, so often different interpolations are used instead.

Different maps might be used by different TIPs or even different components within a TIP. Essentially, every map should provide some location reference system which is used to identify road segments within the map and to link traffic data to them. Currently three location referencing methods exist ([12]):

- Traffic Message Channel (TMC) location referencing[15],
- on the fly referencing (e.g. AGORA-C) without a precise known geo-resolution ([26]),
- a simple, accessible WGS84 co-ordinate together with a road description and a heading.

Traffic Data Arithmetics

There are three main arithmetic operations that can be performed on traffic data returned by function v:

[14]In general this can be any traffic flow characteristic, however, in this thesis we will work mostly with speed or transit time.
[15]http://www.tmcforum.com/

Definition 2.12. *Arithmetic mean* of k traffic functions $v_i,\ i \in \{1, 2, \ldots, k\}$ is:

$$v = \frac{1}{k}\sum_{i=1}^{k} v_i \tag{2.13}$$

Most of the time, the arithmetic mean is performed in order to get an overview of a traffic sample in time (e.g. an average of all Mondays in a month for a particular road).

Definition 2.13. *Subtraction* of two traffic functions $v_1,\ v_2$ is:

$$v = v_1 - v_2 \tag{2.14}$$

Subtraction is primarily used for comparison of the differences between two traffic data sets. In this case, it is important to see if a particular interval and road segment contains no data. Because of this, we define an additional operation - subtraction with availability, that keeps track of the gaps and adds additional value to the gap filled data. Fig. 2.11 shows an example of both.

Definition 2.14. *Subtraction with availability* of two traffic functions $v_1,\ v_2$ is the same as in (2.14) with an additional matrix S that stores a reference to the function(s) that has/have no data for that particular interval and segment.

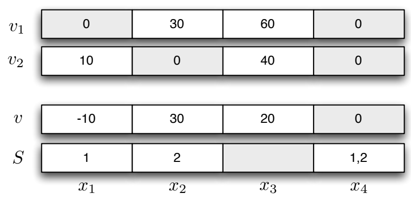


Figure 2.11: Traffic data subtraction. The shaded cells indicate no data. No gap filling method was used, empty values were replaced by 0

2.3.2 Time-Space Plots

In 2.1.1, we provided a detailed description of a time-space diagram. In this subsection, we will show the very practical usage of such a diagram for traffic data analysis, because it is an excellent tool for diagnosing problems in traffic data generated by a TIP. It includes all relevant information regarding the progress of the vehicles on the system during the period of interest and displays this information in a way that can be easily interpreted by traffic analysts. For this, we will present a variant of this diagram: a plot which looks at traffic flow on a more macroscopic scale.

A graphical representation of a function v from the previous section, with values translated into a color spectrum, is a scattered plot as shown in fig. 2.12. This plot is a speed visualization of a traffic situation. It allows us to easily identify traffic congestions and presents us with a clear overview of different states of traffic flow: free flow (2.9), saturated flow (2.10) and capacity flow (2.11). No gap filling is necessary for visualization using a time-space chart. Empty values are usually highlighted by a special color to easy identification of possible problems.

Subsequently, we will be mostly referring to this variant of the time-space diagram. This scattered plot is the essential and most commonly used tool for traffic data analysis as it presents a detailed overview of the traffic situation in observed time-space. Often, for practical reasons, we will swap the axes. It's easier to plot the space axis horizontally because the observed road usually takes more space than the time period (most of the time we work with a maximum range of one day). Also, it is somewhat easier to match the road stretches with the underlying map because it is also positioned in the x-axis.

2.3.3 Time Series Plots

A time series is a sequence of data points, typically measured at the end of successive time intervals (usually of a minute). In traffic analysis, time series are mostly used to visualize transit time over certain period of time. This series provides an information of how long would it take to travel a selected road at a particular time.

There are two strategies for computing the total time needed to travel along road R starting at time t_0:

Definition 2.15. *Instantaneous transit time* τ is simply calculated as a sum of road stretches' transit times at departure time t_0.

$$\tau = \sum_{s=1}^{n} \frac{|x_s|}{v(t_0, x_s)} \tag{2.15}$$

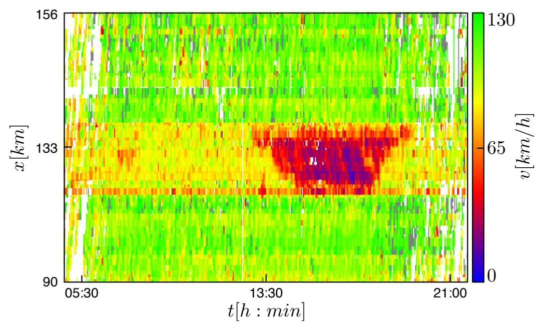
In a $t - x$ plot, the trajectory of this sum is parallel with the space axis.

Definition 2.16. *Traveler transit time* τ' takes into account the actual time Δt_s that is needed to traverse all road segments prior to x_s, so for the next segment x_{s+1} it adds the transit time measured at $t_0 + \Delta t_s$.

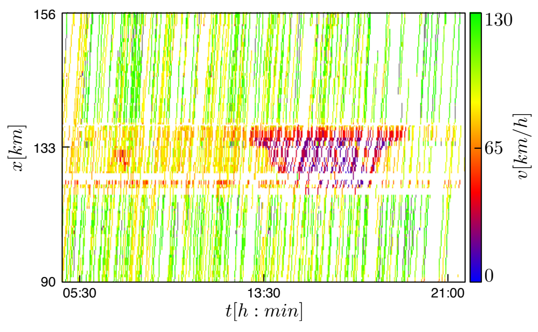
$$\tau' = \Delta t_s \quad \Delta t_s = \begin{cases} \frac{|x_s|}{v(t_0+\Delta t_{s-1}, x_s)} & \text{for } s > 1 \\ \frac{|x_s|}{v(t_0, x_s)} & \text{for } s = 1 \end{cases} \tag{2.16}$$

In a $t - x$ plot, the trajectory is the real trajectory of the traveller. This transit time reflects the traffic situation much more accurately.

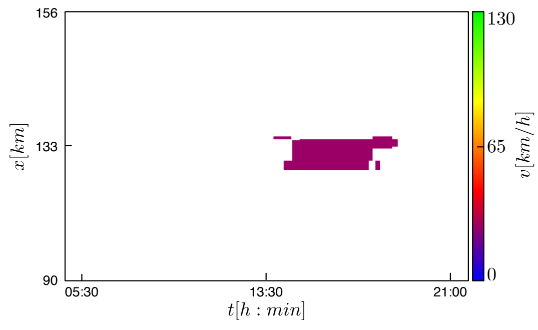
Another time series that is particularly valuable in combination with transit time series is a series which represents the data coverage at every time interval as a ratio of the number of covered segments (where there have been data measured) to the number of



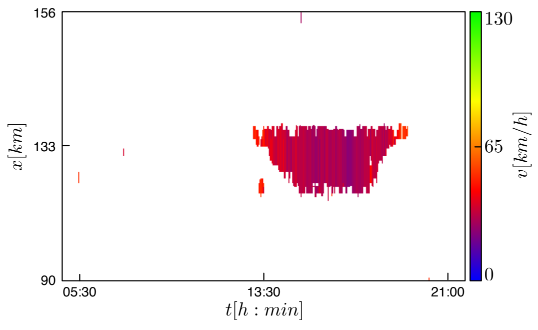
(a) GSM-FCD



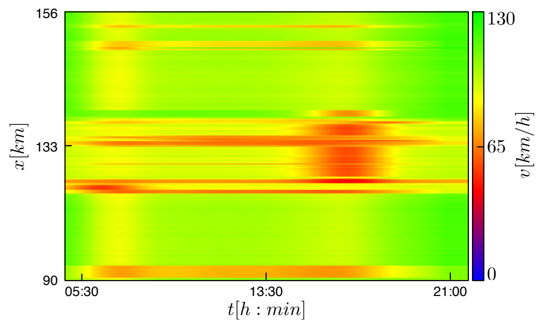
(b) GPS-FCD



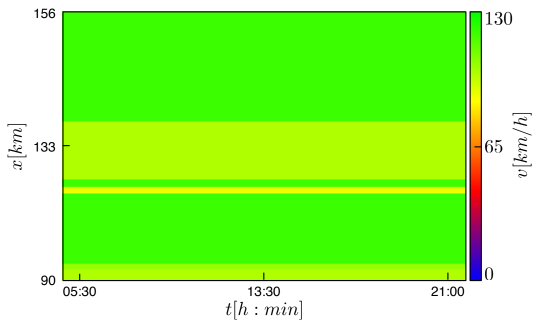
(c) Inductive loops



(d) Fused data



(e) Speed profiles



(f) Free flow

Figure 2.12: Examples of time-space diagram. They show the same road (the A2 motorway from the Netherlands to Belgium) and the same time with data measured in different ways. In the vertical axis, we can identify the different road segments. The sample interval used was one minute. The systems that these data originate from will be described in 3.1.

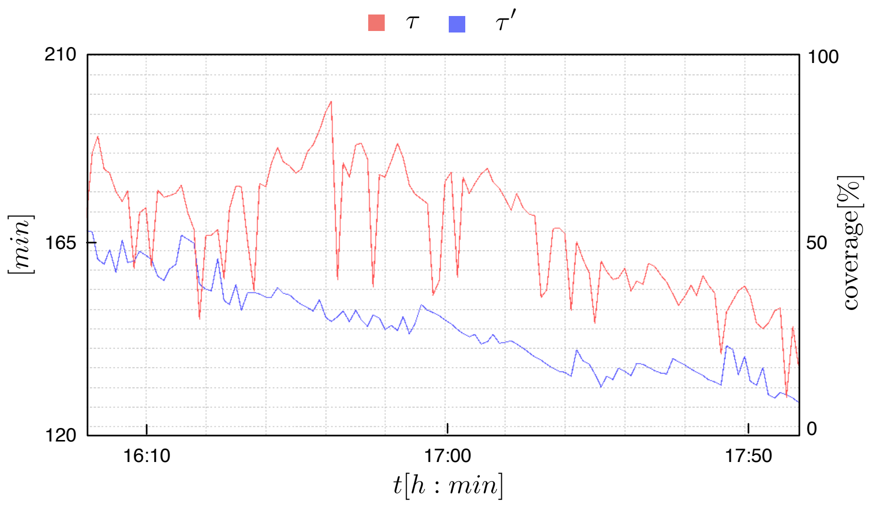


Figure 2.13: Time series showing instantaneous and traveller transit time series for two hours on entire A2 (from The Netherlands to Belgium) during the afternoon rush hour.

all considered segments. This will allow us to interpret results with gap filled parts more accurately (2.14).

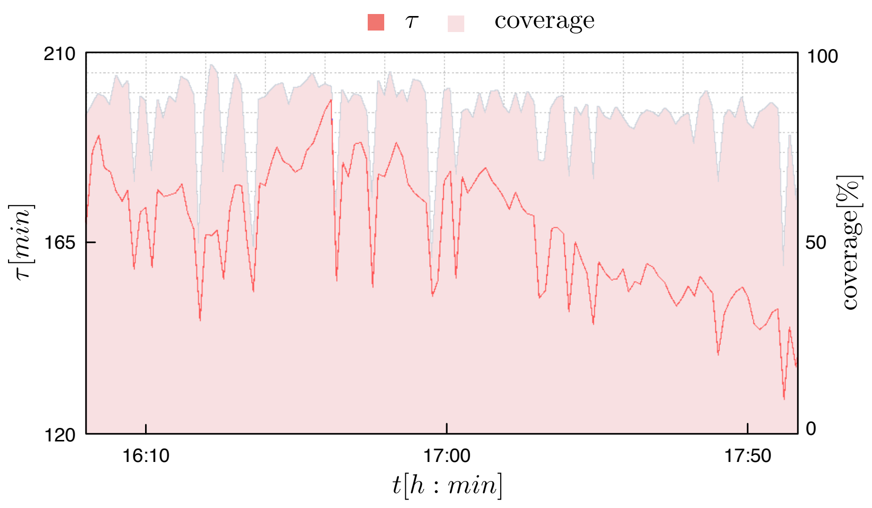


Figure 2.14: Time series showing instantaneous transit time series together with coverage for the same road and time as in 2.13.

Chapter 3

Project Introduction

In this chapter, we will introduce the analysis application project that is the goal of this thesis (sec. 1.2). Firstly, we will present an overview of TomTom HD Traffic (HDT) - the traffic information provider whose data are the subject of our analysis. This section is dedicated to the requirements for this project. The last part will comprise a technology assessment, during which different technologies will be evaluated in order to choose the ones suitable for building the application.

The project of traffic data analysis - *Heavenly*[1] - is part of TomTom's HDT chain. Its main purpose is to assist traffic engineers and developers in traffic data analysis; allowing them to look at the data in a visual way on a microscopic level - observing an arbitrary set of roads during some time period. The aim is to have assistance in providing a detailed picture of a traffic situation at a certain location rather than an overview of the entire system performance.[2]

3.1 Overview of TomTom HD Traffic

TomTom High Definition Traffic is a system that provides real-time traffic information to TomTom's customers via their PNDs or web-based route planner.[3] By the end of 2008, it is scheduled to be operating in five European countries - the Netherlands, United Kingdom, Switzerland, France, Germany – and the intention is to expand the service further, even to other continents, with the goal of eventually covering all primary and a major part of all secondary roads and providing frequent traffic updates (approximately every three minutes).

Figure 3.1 shows a simplified overview of the system's component and the data flows between them. The colors of components match[4] the different parts of traffic data chain

[1]The code name has been chosen after the name of a skiing resort in Northern California, in the Lake Tahoe region called Heavenly.

[2]This is also very important, but beyond the scope of this thesis.

[3]http://routes.tomtom.com

[4]A gradient was used when a component is responsible for more than one phase.

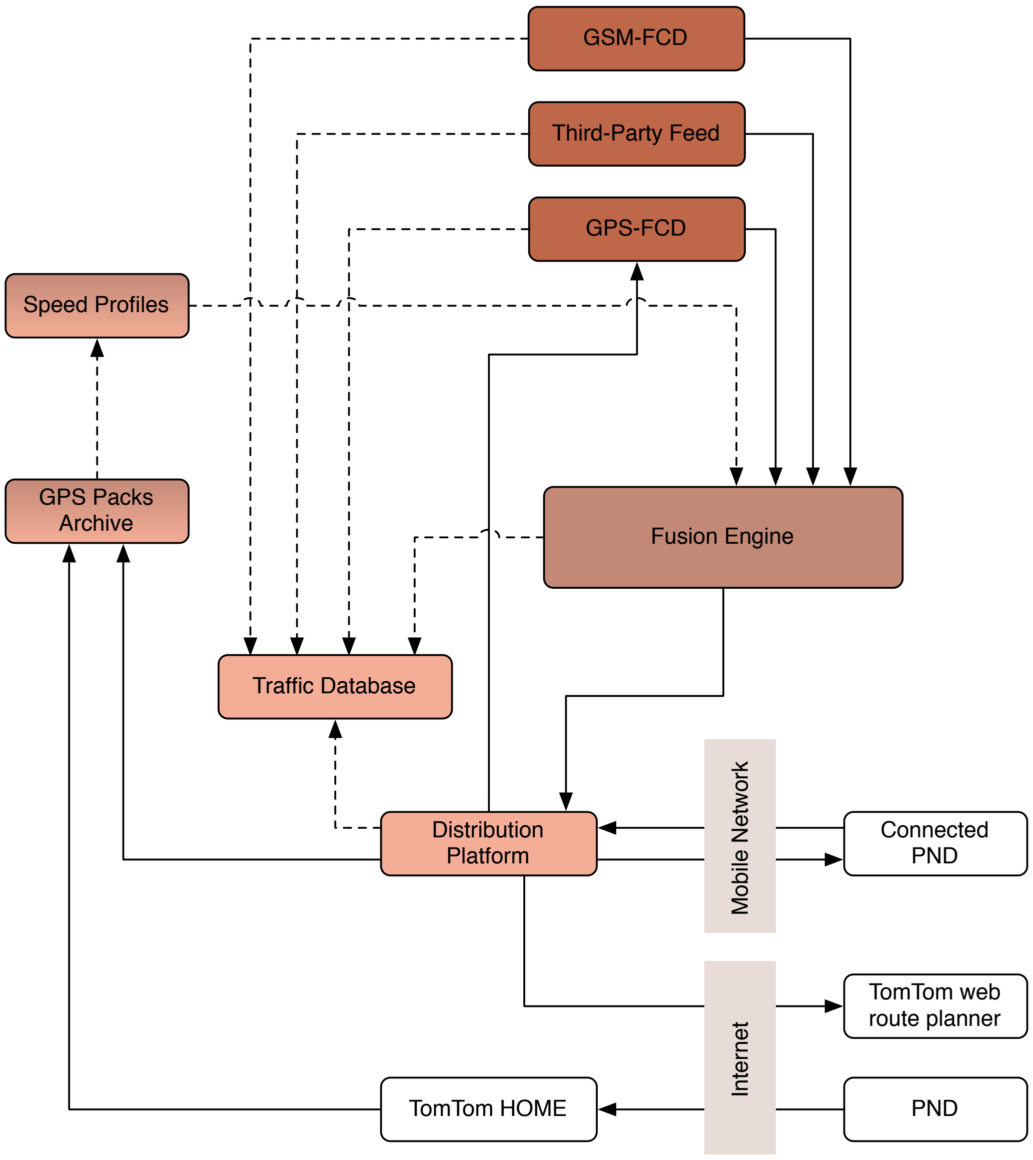


Figure 3.1: TomTom HD Traffic Overview diagram with traffic data chain components highlighted and dashed line indicating the data flow critical for data analysis.

as they were shown in fig. 2.8 and the dashed lines mark the data flow that is critical to traffic data analysis, including the components that produce it. The diagram has been simplified to show only the components relevant to this thesis.

As an example of the differing nature of the data in the HDT chain, we will use figures from 2.12. These figures present all the different traffic data types within the chain in time space plots. In the next section, we will briefly describe the components that are part of HDT, as the analysis tool will be part of the same environment and will be working with data generated within this chain.

3.1.1 Collection

As can be seen from the overview diagram (3.1), there are four different components that are responsible for traffic data collection. They use different sources as input, but all provide the same output which goes into the processing traffic data fusion engine component and is copied to the traffic database, which is (as it will be described later) the historic traffic database that stores data from the HDT traffic data chain.

GSM-FCD

The HDT FCD system, based on anonymous mobile phones probes, was developed by Applied Generics, Ltd., which was acquired by TomTom in 2006. The input to this system are raw network probes for all active subscribers in the cellular network of the particular provider.[5] The system is deployed in the provider's data network, running on the management servers, collection units and gateways. It determines road status based on the raw data feed of GSM events, a cell plan and geographical map of the country. A single collection service provides a road status feed for a part of the country. The entire country or region is monitored by a cooperating network of collection services. It is responsible for matching collected data to the digital map and all necessary filtering, like removing non-road initiated data (e.g. data originating in trains) in order to provide accurate positioning information relevant to the road network. The output generated and encoded using the Gateway Road Status Traffic Protocol (GRSTP, TomTom proprietary) is constantly pushed over a secure communication channel to the TomTom data-center, where the rest of HDT is running. An example of collected data is given in fig. 2.12a.

GSM-FCD has the largest coverage in the traffic network and generated data covers all traffic flow states (2.1.3).

[5]Currently, Vodafone in the Netherlands and United Kingdom, Swisscom in Switzerland, D2 / Vodafone in Germany and SFR / Vodafone in France.

GPS-FCD

GPS-FCD is a real-time GPS data processor with source data coming from TomTom connected devices,[6] which upload current trip data (GPS traces) to the server over a mobile network (via a distribution platform). Since these data are coming from PNDs and have already been properly positioned in the PND's digital map, the quality of this data is usually much higher than the ones generated by floating car data using GSM. Uploaded traces are, as in case of network probes, anonymous and every user can decide whether he or she is willing to contribute or not. An example of collected data is in fig. 2.12b. As with GSM-FCD, GPS-FCD covers all traffic flow states.

Third-Party Feeds

In order to provide as complete a picture of the traffic situation as possible, including for example road closure and construction announcements, HDT is trying to leverage existing TIPs in certain country by becoming a consumer of their data. What sort of technology is behind the third-party feed differs from vendor to vendor. In most of the cases, these data originate from a road infrastructure system, usually inductive loops 2.2.1. An example of a 3rd party feed that is used in the Netherlands[7] is in fig. 2.12c. Depending on the vendor's implementation, it usually only receives data for traffic incidents corresponding to saturated traffic flow (2.10).

3.1.2 Processing

GPS Packs Archive

This system is responsible for the archiving of anonymous GPS trace data from PND devices. The sources of traces are both connected and non-connected devices. In the case of connected PNDs, the same data as for GPS-FCD are used. From a non-connected device, a computer with TomTom Home[8] software is used to upload logged trip data (again only if a user is willing to contribute). The archive itself does not do any collecting, it works with already collected data, which are being processed and stored. It also provides a great opportunity for further data mining.

Archived data are continuously processed as traces come in. This means that the aggregated data in the system might change as new measurements are processed. This is different from the other systems whose data are purely static.

An example of these data would be similar to GPS-FCD data (fig. 2.12b).

[6]TomTom GO series 715+, 915+ and TomTom Work devices combine a GPS receiver and GSM modem.

[7]Verkeerscentrum Nederland (VCNL) van Rijkswaterstaat

[8]The client software responsible for device management.

Speed Profiles

In the previous paragraph we mentioned the archive of GPS traces. These historical measurements are collated and used to generate profiles of the average speed on a road during a day. These speed profiles[9] are used alongside the regular (free-flow) speeds v_f from the digital map. This significantly improves the available speed information and therefore gives a much more accurate ETA (Estimated Time of Arrival), since we all know from real life that the average travelling speed varies a lot with the time of day and day of the week. Also, the confidence level in the system's estimate of the traffic situation can go up if a detected situation matches what is to be expected historically, even if the actual delay time is still calculated using the original free flow speeds.

In the latest 2008.10 Tele Atlas map, these profiles cover 8.3M km of roads in Europe (100% of all traversable roads) and 9.3M km of roads in USA (89%).[10] Figure 3.2 shows an example of a speed profile as a time series chart, a time distance plot example is in fig. 2.12e.

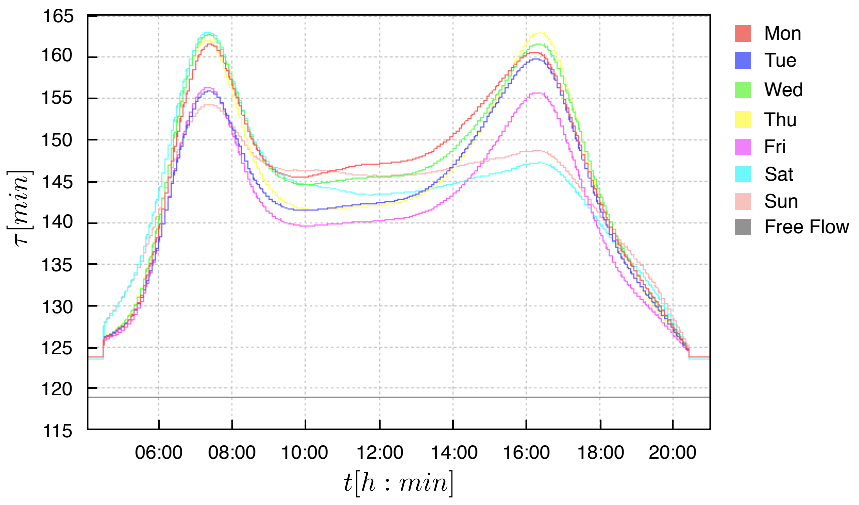


Figure 3.2: An example of the speed profile on motorway A2 from the Netherlands to Belgium for the full week including default free flow value from the digital map.

Fusion Engine

The fusion engine is the server component responsible for estimating the current traffic state in a road network based on information from different sources. It is the main processing engine in the HDT chain, responsible for fusing all input traffic data sources available. Currently it uses all collection components, as previously described, as well as a digital map enhanced with speed profiles. By fusing this information it continuously

[9]The commercial name of the technology used by TomTom is IQ ROUTES^{TM}.
[10]from Tele Atlas Speed Profiles 2008.10 product brochure

generates an estimation of the current traffic state in the observed road network. The output of the engine are real-time traffic incidents. This information is dispatched to a traffic database for storage and via a distribution platform to end consumers. It also keeps long-term statistics about the most important incidents in terms of length, delay, duration, etc.

Fig. 2.12d shows an example of processed data output by the fusion engine. This particular output has been generated using all the other traffic data from fig. 2.12 as the input. It shows that, as in third-party feed, it also reports the only traffic incident (e.g. when the speed drops below certain threshold).

3.1.3 Publishing

Traffic Database

In the HDT chain, this is the database of *historical* traffic data. The input is the data gathered by collection components, processed by the fusion engine as well as information published by the service distribution platform. Therefore, its main concern is data analysis. It creates a traffic model from the data received and periodically saves them. Each input is stored separately.

Distribution Platform

The service distribution platform is responsible for communication between services and PNDs over mobile networks. For PNDs, it is the provider of all dynamic content, including traffic incidents generated by the fusion engine. The very same traffic incidents are also stored in the traffic database for validation purposes.

3.1.4 Traffic Sources

The above components are the most critical for traffic data analysis. From fig. 3.1 we can identify four main sources of traffic data in the HDT chain:

- historical traffic data from the traffic database,
- the GPS packs archive,
- speed profiles, and
- free flow information from the map.

All these systems are running in a remote data-center and are storing data in some proprietary file format to the file-system; no third-party database system is used. From the data accessibility point of view, we can consider these systems as Java libraries providing certain APIs (Application Programming Interface) that allow us to access the data through a function that, for given time and road segment, returns measured data.

HDT's deployment is partitioned geographically; for each country, there is a specific configuration of the components, but the stored data are accessible from a single point in a data-center.

3.2 Requirements

In the previous section, we presented the environment in which our system will be operating and defined the main components that are crucial for traffic data analysis. This section will discuss the main requirements that the system should fulfil in order to be a useful tool for traffic analysis as discussed in 1.2.

Requirements gathering is the initial phase of any project life cycle. Sometimes it might be preceded by a feasibility study or other research that should answer the question of whether or not to actually proceed with the project, but once the project has been approved, the requirements are the starting point. A requirement is a documented statement that identifies a certain functionality or quality of a system necessary for it to be valuable to the end-user. Having a good set of requirements is crucial to the success of any product.

In software development requirements may be divided into two groups: *functional* and *non-functional*. The functional requirements describe what the system must do. It is a specification of something that is going to be developed and delivered in the target system. The non-functional type of requirements specify something about system itself. They describe how well it must be done. Put another way, functional requirements can be seen as an answer to question "what will happen if user presses the button" while non-functional requirements would be the answer to "what will happen if 1000 users press the button at the same time". The non-functional requirements are sometimes referred to as *quality of service* requirements and can be further classified as performance requirements, maintainability requirements, safety requirements, reliability requirements, or one of many other such types ([32]).

In this project, we define functional requirements in the form of use cases, which is a widespread practice for capturing such requirements. A *use case* defines a goal-oriented set of interactions between actors and the system. Actors are parties outside the system that interact with it ([4]), which may be a user (stakeholder) or another system.

In general, when we look at traffic data analysis and quality assurance, we can identify three main actors:

- power users (developers, traffic engineers, operators),
- business users (product management, marketing, sales),
- management users (management team, board).

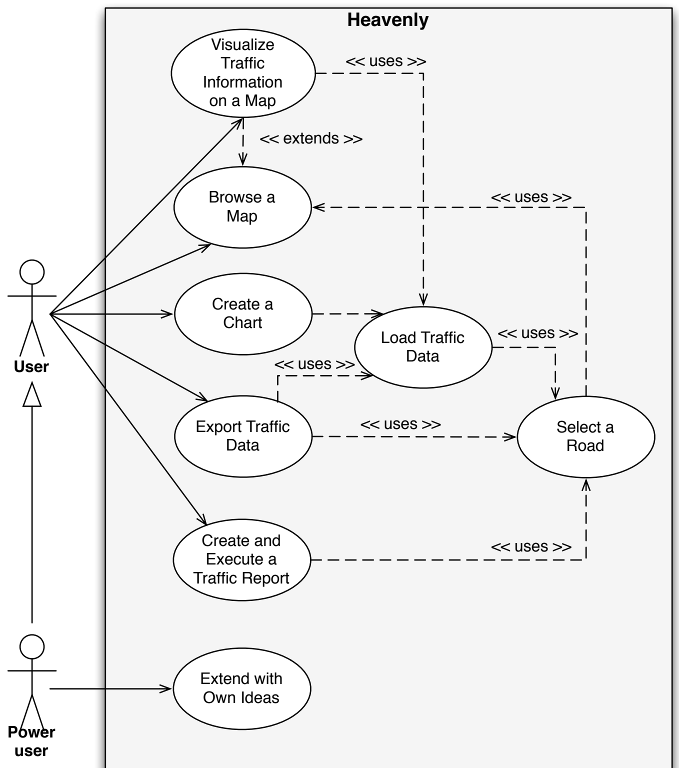


Figure 3.3: Use cases

These actors are interested in one or more of these high-level functional areas:

- problem investigation related to a system misbehavior,
- development, testing and simulation,
- third-party feed assessment (an evaluation of the performance of a third-party feed)
- monitoring of Key Performance Indicators (KPI[11]),
- marketing, generation of various reporting for marketing purposes.

The scope of Heavenly is primarily to support power users in problem investigation and development. Occasionally, the tool might be used for marketing purposes.

Use Cases

In this section, we will present an overview of the high level functional requirements (HFR) capturing all major aspects of the construction of the system.

Use cases are usually kept in two forms: in an overview diagram[12] and as a written description. Fig. 3.3 shows the main HFRs for the system:

[11] A metric defining and measuring progress toward certain goal.
[12] A use case diagram as defined in UML (Unified Modeling Language [4]).

Browse a Map A user explores map a by panning and zooming in or out. He or she can search for location by typing an address or part of it and then moving the map to that location a zoom level based on the type of location (e.g. zoom out on a city, zoom in on a street). The map images and geo-coding service for translating addresses to geographic locations should be identical to the one used in TomTom's web route planner.

Select a Road A user selects a road that will be the subject of analysis by one of these methods: 1. loading a file containing a list of segments identifiers; 2. entering two TMC locations; 3. entering the road name and selecting country; or 4. visually by selecting start and end point and using a route algorithm to connect it or, more refined, using guided routing (clicking and connecting individual road segments). This use case is not directly triggered by the user because road selection is only done for some additional purpose, like creating a chart.

Load Traffic Data A user loads traffic data from a selected traffic source(s) (3.1.4) for a selected road(s) and time period. The user can load raw or processed data - averaged or subtracted (2.3.1). Should the loading of the data take long time, the user is notified about progress. It is important that user is able to work with any traffic data from the HDT chain. Again, this use case is not directly triggered by user, because loading of traffic data is only done for some addition purpose.

Visualize Traffic Information on a Map A user can see traffic information from a selected traffic source displayed on a map. Traffic data are displayed as poly-lines (using the road segment geometry) above the roads with colors matching the indicated speed. The amount of displayed information is dependent on the zoom level (e.g. at country level, only motorways are displayed).

Create a Chart A user creates a time-space (2.3.2) or transit-time (2.3.3) plot for previously loaded traffic data. There are system-wide preferences for these charts (like colors, titles, line types, etc.). A user can explore different parts of a chart by zooming in or out. The range of both axes can be adjusted once the chart is displayed. The chart can be saved as an image to user local file-system.

Export Traffic Data A user exports loaded traffic data as a CSV (Comma Separated Values) or TSV (Tab Separated Values) file for further analysis in an additional software package like SPSS.[13]

Create and Execute a Traffic Report A user can create and execute a traffic report. A traffic report consists of various charts (time-space and transit-time) for an

[13]Originally: Statistical Package for the Social Sciences, a leading statistical analysis software http://www.spss.com/.

extended time period, multiple roads and multiple traffic sources. The output of a report execution is a set of images containing selected charts for all combination of roads, time and traffic sources. It provides an easy way to get an overview report about traffic situation in the time/area of interest.

Extend with Own Ideas A user can extend existing functionality with his or her own ideas. This covers the possibility of prototyping new functionality and the ability to present the traffic data in different forms from the ones already mentioned. The user can also to automate certain tasks, so that they can run non-interactively. This is important as there will always be some desired feature that the UI does not yet support, but this should not prevent a user from carrying out his or her tasks.

Non-functional Requirements

Installability Measure of cost of installing the system, initially or after an upgrade. Installation and configuration of the complete system should be done by any user in the target group within a few moments. For a special configuration that might arise in the context of a specific architecture and implementation, technical support might be required, but the complete procedure should not exceed a few hours of work. The software should be capable of self-updating; a user should be notified that a new version is available (including a list of changes).

Operating system The system must be able to run on Windows (32-bit) and on Linux (32-bit, 64-bit) - CentOS and RedHat Enterprise Linux.

Language HDT itself is primarily written in Java. In order to be able to reuse the API for data access and other functionality, Java has to be used as the main programming platform and the development should comply with HDT standards. Another reason for Java is to standardize the development languages (and tools) so further resource allocations is easier.

Up-to-dateness Measure of how long it takes for the latest information stored on the server to be available on a client after issuing a request to a service. For historical data, this metric is limited by the up-to-dateness of the traffic database. For real-time data, it depends on the connection, but should update as fast as the data in the fusion engine's management console.[14]

Testability This is a constraint on the way the software should be developed (or existing software modified). It should be done in such a way that automated testing can be done easily. In the Java world, this means being able to write unit and integration tests easily using stubs and mocks (will be discussed later in the next chapter).

[14]The management interface for the fusion engine system that is used internally.

Usability Measure of the ease with which people can employ a particular tool in order to achieve a particular goal. This is very difficult to measure as it tends to be very subjective, however all major concepts of user interaction with the system should be agreed with the target user base.

Data accessibility This analysis application is a data oriented system and deals with large amount of traffic data. This should, however, not limit the user in terms of how much data he or she can access, as the system should be designed in such a way that all historic and real-time data can be available to user at any time. The data are stored in an external data center with limited access due to security restrictions. Analysis might also be done over a Virtual Private Network (VPN) and therefore bandwidth restrictions might apply and should be taken into account while designing any communication protocols used. All traffic data are processed and stored in a dedicated environment with limited remote access, thus some part of the application must be part of the data-center in order to be able to access the data.

3.3 Technology Assessment

Currently, there are couple of applications on the market that could be used for data analysis with a suitable spacial extension (as traffic data are aligned to geographical locations), but because of the way traffic data are distributed within the HDT chain (all based on proprietary TomTom formats and protocols), none of the existing solutions could be used without modifications. One of the options would be to modify the storage components, like the traffic database, to use a standardized format instead of custom file-based structure. This option was considered but rejected for cost reasons, because without a proper enterprise solution (including very expensive hardware, software and support) dealing with such a big datasets would become a major performance bottleneck.

One of the applications considered was Matlab,[15] but as much as it is a great system for data modeling and could be connected to the traffic sources, its performance when handling a lot of data is rather slow. Furthermore, the price of a licence with map extension is quite high.

Other applications that exist today in the marketplace were not much of an option either. Besides high prices, there was no in-house experience in using them, which is especially crucial in order to be able to obtain reasonable performance. This would delay the project as one would first have to understand the internals of a third-party application and would introduce serious vendor lock-in, because any data adaptation layer would have to be developed for the needs of the chosen commercial system, as would all further

[15]A numerical computing environment and programming language created by The MathWorks (http://www.mathworks.com/).

modifications of tools for analysis. Also, from the very beginning, there was a consideration that if the project were successful, it could be become a commercial product sold together with traffic data.

This resulted in the decision to build a custom system. In the next chapters, we will discuss its architecture and implementation, as well as the different technologies that were considered and used in building it.

Chapter 4

Architecture Design

In the previous chapter, we described the main requirements that the traffic analysis application should fulfil in order to be a valuable tool to traffic engineers. In this chapter, we will propose an architecture for such an application together with a description of individual components.

There are number of software architectures that can be used to build an application, but with the requirements (3.2) given, we decided to use classical a client-server architecture. In this model, client and server systems are separate components which communicate via a well-defined protocol over a computer network. This model is de-facto the standard for building applications that have a central remote data repository, because it allows us to combine flexible remote data access with a user interface.

In a client-server architecture, the responsibility for a task execution can be either part of the client or server side, or may be combined. Usually, the decision is based upon the requirements of the user interface, but also on the basis of security, flexibility, etc. considerations. This is a major decision that fundamentally influences the design of the application as a whole. There are two general options under this architecture model: 1. using a *thin client* or 2. using a *thick client*.[1]

A thin client is primarily dependent on a server for any kind of data processing, mainly focusing on mediating input and output between the user and the remote server. In contrast to that, a thick client is an application that obtains data from a server, but allows the user to manipulate the received data independently from a server, usually via a rich user interface (UI). The business logic in a thick client is mainly present on a client side, whereas in thin client, it is on the server side.

Our system will consists of a rich client application (front-end) and distributed, service-oriented server (back-end). This model allows us to create a powerful user interface at the front-end while maintaining the data accessibility at the back-end and therefore satisfy the requirements of data interaction while being able to have the back-end running with the

[1] Also referred as a fat or rich client.

rest of the HDT systems in a remote data-center, so maintaining access to the traffic data. Figure 4.1 shows the concept of a client-server model. Specifically in this model, any client can talk to any number of servers and a server can talk to any number of other servers, it becomes yet another client to a server.

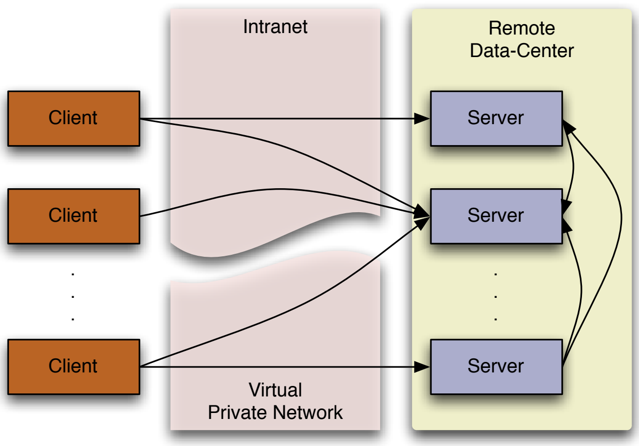


Figure 4.1: Architecture overview

4.1 Server

The main responsibility of the back-end is to provide clients with traffic and map data. The server groups functionality around business processes and exposes them as loosely coupled,[2] high cohesion,[3] interoperable services. A service in this context is a defined operation with the following properties:

- has an unique identifier,
- has a defined number of serializable input arguments,
- can return a serializable result of its execution,
- throws a service exception in case of failure,
- is *stateless*; does not maintain any internal state based on the input arguments.

The distinct functionality units - services - presented are accessible to the client via some communication framework so it can combine and reuse them. Executing services with appropriate arguments in a certain sequence in order to satisfy a business need is called *service orchestration* and is the main responsibility of the client. This paradigm is known as Service Oriented Architecture (SOA).

[2]Loose coupling is the degree to which each program module relies on each one of the other modules ([?]).

[3]Cohesion is a measure of how strongly-related or focused the responsibilities of a single class are. ([35].

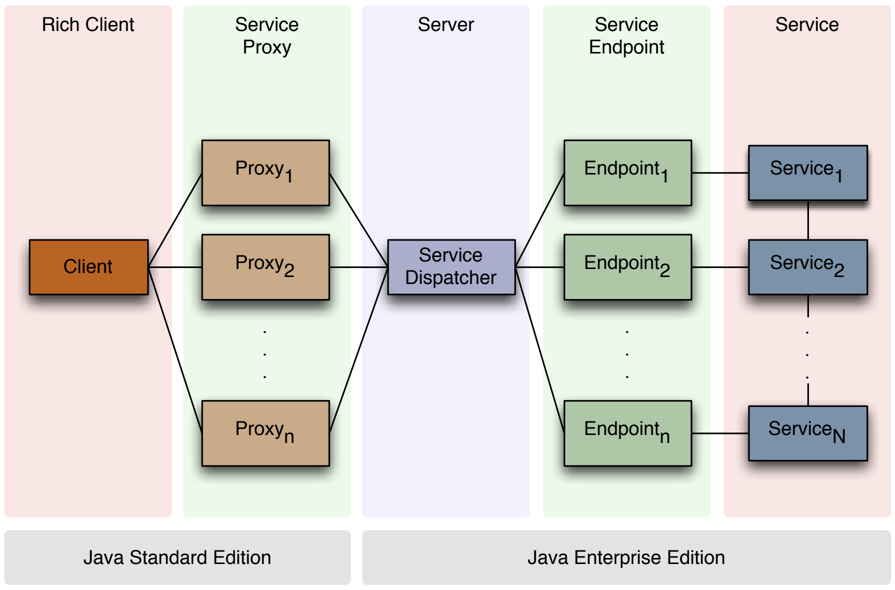


Figure 4.2: Client-server communication

Making the services stateless is very powerful, as it allows to easily scale as needed by just adding new server entities without introducing any bottlenecks (i.e. horizontal scaling [36]). This can be done because neither services nor servers keep any state information about connected clients and it therefore does not matter which back-end server serves which client.

The communication is done over a computer network. In (3.2) we mentioned the access restrictions to the traffic data, which mean that the server effectively has to be running in the same data-centre. There are strong security precautions and the only available protocol for communication is HTTP (HyperText Transfer Protocol) [11]. HTTP is a stateless, lightweight, application-level protocol which is suitable for communication between distributed collaborative systems. It defines a standard request-response sequence as well as a set of error codes which can represent various failure states. We will reuse these HTTP requests as service calls (with properly encoded identifier and arguments) and HTTP responses to send back the results of an execution (again with a properly encoded return value or an exception in case of execution failure). Figure 4.2 shows the schema of communication between client and server.

The server comprises of two core components:

Service Registry is a component responsible for managing services. It allows registration and de-registration of a service, as well as providing references to already registered ones. Services are registered by identifier (which must be unique for each service instance within a server), type and optional properties that describe additional aspects of a given service (i.e. map version, country identifier, etc.). Once these properties are registered, they may not change.

Service Dispatcher is a component responsible for transforming request to a corresponding method call and handling all exception states. For failures HTTP codes are reused.[4]

The sequence of processing a service call is shown in detail in the UML sequence diagram in fig. 4.3.

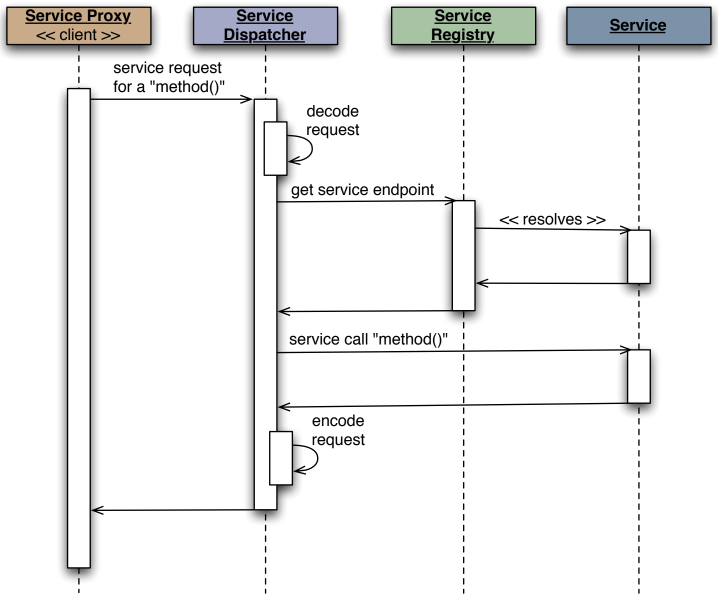


Figure 4.3: Service call sequence diagram

The services have been designed to provide a rather low level API. One of the reason for this is to maintain the ability of having stateless services, the other is to provide flexibility for the clients and allow a high degree of optimization of each call, because the passing objects and return values are represented as simple objects. Design is, however, flexible, and should the need for more complex services arises, it will be just a matter of creating yet another service to satisfy the new user requirement(s).

Following is the list of available services required to fulfil the given requirements. Fig. 4.4 is a corresponding overview class diagram of the server part.

Discovery Service is a service that allows clients to discover which services are available on a server. It returns all necessary information about these services so clients can make appropriate use of them. Clients should be aware of the fact that a new service may be registered at any time, and also that already registered services might disappear at any time. In this architecture, the client is the one who initiates the

[4]For instance HTTP 404 Not Found signals no service with such identifier has been register or HTTP 500 Internal Server Error signals runtime exceptions and Java virtual machine errors.

connection to a server (pulls the information), thus the server does not provide any notification when its state changes.

Map Service is a service providing all necessary operations upon a map, primarily to support road selection.

Traffic Data Services is a service providing access to traffic data. For each traffic data source (3.1.4) instance there is a concrete service registered. The main function is to allow the client to query for traffic data for given road segments and periods of time, returning measured transit time (which, by knowing the length of a segment, gives average speed) similarly to the function discussed in 2.3.1.

Report Execution Services since traffic reports might cover many roads and long periods of time, it is reasonable to have it execute on a server rather than on a client as servers are much more powerful and have more resources available than client machines. This service is responsible for report job management. Clients are able to submit a new report, cancel an existing one, get the progress of currently running reports and receive the result of successfully finished reports.

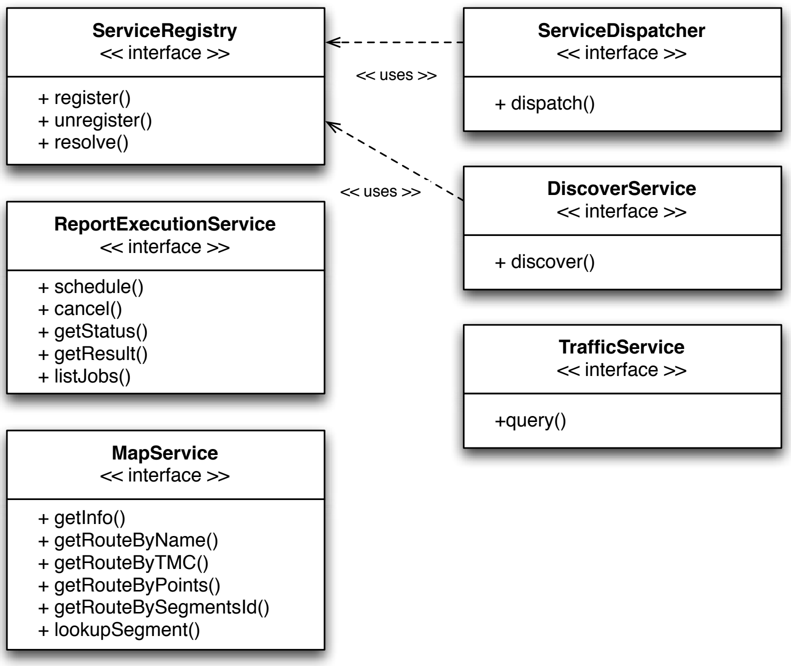


Figure 4.4: Overview class diagram of the server part. Not all available methods are listed; arguments and return types were omitted.

4.2 Client

Rich client applications were traditionally represented by desktop applications with some sort of graphical user interface, but recently also started to be more and more exposed on the web.[5] These clients combine a high-quality end-user experience for a particular domain by providing rich native interfaces (with supports for native desktop metaphors such as drag-and-drop, clipboard, navigation and customization) with high-speed local processing ([18]). In order to be able to fully utilize available resources at a client machine and the richness of today's desktop application framework to provide a powerful, yet easy to use user interface, we have decided to build the client as a desktop application. The other alternative would be building a web application, but it would take a considerably more effort to make a scalable analysis front-end using web technologies.

Building a serious desktop application with a rich user interface is not a simple task and today no one starts from scratch, but rather reuses one of the existing frameworks. [33] contains a summary of the most popular frameworks designed for building desktop application in Java. Application frameworks can be divided into two groups: those that add some extra functionality above the raw windowing toolkit and those that go beyond this and provide the complete solution known as a Rich Client Platform (RCP). RCP is a software base which developers can use to benefit from proven and tested features like dependency management, update management, robust UI widgets, etc.

Nowadays, in the Java world, there are two main such platforms: Netbeans RCP[6] and Eclipse RCP[7]. They are both mature, robust products and offer similar functionality at no cost, but use different approaches for module management and are build upon different windowing toolkits.[8]

We have evaluated both platforms and in the end decided to use Eclipse. The main reason was in-house familiarity with Eclipse as an integrated development environment, better documentation and the fact that it is surrounded by a very active community. Further, Eclipse itself is based on the very powerful OSGi (Open Service Gateway Initiative) Service Platform which we will be extensively using in Heavenly.

Figure 4.5 shows the basic tiers in the Eclipse RCP architecture:

Java Virtual Machine Eclipse RCP is a regular Java application that runs on the standard Java SE platform. It also contains some native code, mostly for the native windowing toolkit (SWT).

[5] Like Adobe Photoshop Express (https://www.photoshop.com/express/) or Google Documents (http://docs.google.com/).

[6] http://platform.netbeans.org/

[7] http://wiki.eclipse.org/index.php/Rich_Client_Platform

[8] an in-depth comparison appears in [28]

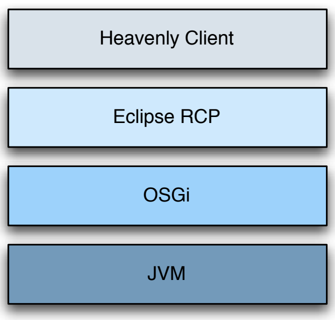


Figure 4.5: Eclipse RCP application

Open Service Gateway Initiative is the framework specification[9] that Eclipse is based on. This layer is responsible for bundle management including identity (to other bundles and to the system), dependency, versioning and life cycle management. The other very important aspect is a service registry. Any bundle can at any time register (or unregister) a service which is a plain Java object that implements agreed interface. Unlike the service layer in our server component, these services do not need to have serializable arguments or return values, because they are all part of the very same JVM and share the same address space. The other main difference is that services can be observed, with listeners being notified every time a new service is available, disposed of or when its properties have been modified. OSGi provides all necessary APIs for managing service life cycle, discovery, etc.

In general, OSGi can be though of as a dynamic module system in Java and it is being used in many different domains, from mobile automotive to enterprise applications. The specification is defined and maintained by the OSGi Alliance. The complete specification is available in [20]. The Eclipse Foundation[10] provides the reference implementation, called *Equinox*[11].

Eclipse Rich Client Platform is the base platform for building rich client applications. The basic unit of function in the platform is called *plug-in*, and applications based on this platform are built by composing such plug-ins. The notion of a plug-in is synonymous with the notion of bundles. Everything within the platform is a plug-in (the platform itself is a set of interdependent plug-ins). This approach of strong modularization is very attractive as it allows the application to easily evolve over time by adding and replacing components.

In addition to the component model, there is a set of extensive frameworks and other facilities that simplify the job of writing applications like flexible and scalable UI

[9]Sometimes also referred as a Service Platform.
[10]http://www.eclipse.org/org/
[11]http://www.eclipse.org/equinox/

widgets, error handling, network updates, help system (with context support), etc. Eclipse is using a custom windowing toolkit called SWT (Standard Widget Toolkit[12]. It is an efficient portable GUI toolkit to access to the native UI facilities of the operating system ([18]) and runs on most of today's architectures and operating systems.

Heavenly Client is a RCP application composed of bundles that are plugged into the platform reusing or extending provided functionality. Heavenly is using a lot of Eclipse specific mechanisms, especially with regards to UI elements like editors, wizards, views, etc. Each module provides a set of services that other modules can use, or which can be bound to a particular element of the user interface.

In the following section, the main plug-ins of Heavenly will be briefly described. The design approach taken assumes clear separation between modules with well-defined interfaces. This allows us to adopt a test-driven software development approach (and therefore help to satisfy the testability requirement of 3.2), which is especially significant in case of a system dealing with a large amount of data, due to the complexity of verifying application correctness. It will also ease the process of understanding the system, as each component can be described separately.

Core

This plug-in defines the core functionality that is shared across other plug-ins. It is primarily responsible for providing:

- access to all services from the server,
- a high-level traffic and map API,
- a uniform logging service,
- a uniform error handling service,
- a general purpose cache service,
- the application container and the main GUI.

Configuration Service

The configuration service provides a mechanism to discover and access all known services available on the servers. Any plug-in that needs to use a server service can ask the configuration service for a proxy to the service. A proxy is an object that implements the same interface as the target service and knows all the necessary information about how to access the remote service. Once it is called, it delegates the call to the remote target

[12]This name is somewhat confusing, as the standard in Java for widgets is Swing, however here the name reflects the fact that this toolkit reuses the *standard* widgets from the underlaying operating system.

and takes care of all serialization of arguments and return values, if needed. This makes the location of the server transparent to the consuming plug-in, because the requestor does not need to know what underlying communication protocol is used. Since plug-in can request a service by its type, it does not even need to know that the service is remote and located on this or that server. Furthermore, these proxies can be cached and reused as the server services do not keep any state.

In contrast to server-side services, which use low level APIs with serializeable arguments, this kind of simplicity is not necessary on the client side. Instead, client services should provide a stateful, rich API to simplify development. This is done by wrapping the low level server services in more logic, using delegation where required. This helps to centralize access to the server component, in order to minimize bandwidth usage.

Traffic and Map Service

The core plug-in defines two services for working with map and traffic data. These provide a high level API to simplify access to the server side traffic and map services.

In the traffic data package, a `TrafficData` interface represents traffic data for a certain route and period of time, as in 2.3.1. An instance of this interface is returned by a query call to the traffic source service defined by this plug-in. This service uses the server side traffic service to retrieve the actual data, which is cached on the client. Traffic data (4.6) consists of m traffic snapshots TS_r $\{1, 2, \ldots, m\}$ one for each sampling interval Δt within the total time span (T_0, T_1). A traffic snapshot TS_r is a set of n traffic reports TR_s $\{1, 2, \ldots, n\}$, one for every segment x_s of a certain road R for given time stamp $t = T_0 + (r - 1) \cdot \Delta t$. A traffic report contains a measured speed v_{t,x_s} for the given road segment x_s and given time stamp t.

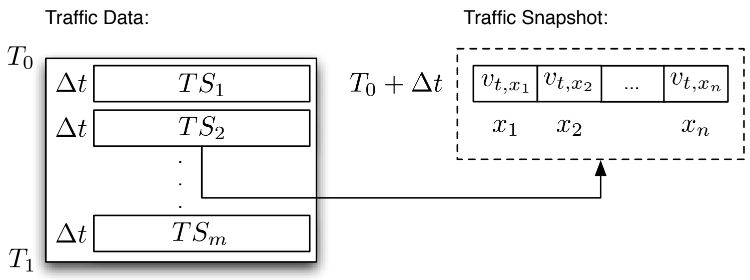


Figure 4.6: Traffic Data

Traffic data is evaluated *lazily*, which means that values are being requested from the server as needed, rather than loaded in advance. Since all the data are stored in the cache, traffic data can seen as a sort of view on the cache. Fig. 4.7 contains an overview of the classes and interfaces defined in the traffic data package. A visitor pattern ([8]) is used to allow us to easily traverse the data structure.

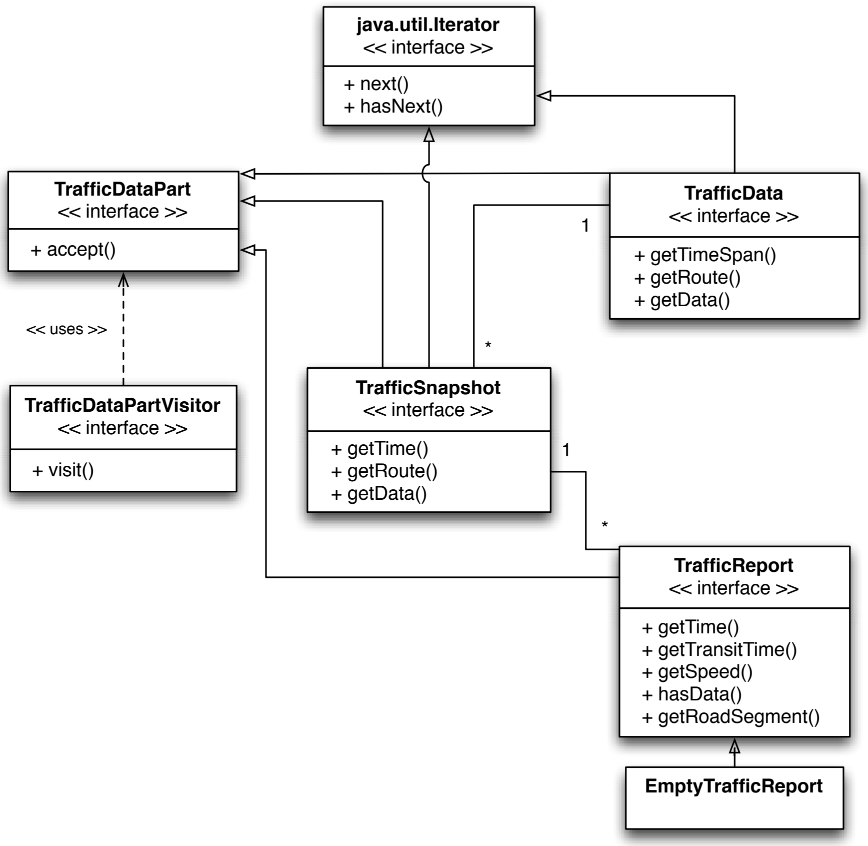


Figure 4.7: Overview class diagram of traffic data package. Not all available methods are listed; arguments and return types were omitted.

Cache

In order to improve performance and save bandwidth, data received from the server are being cached locally. The cache is a very simple object repository with expiration. By default, a most recently used strategy is used to maintain a reasonable memory footprint. A user is provided with a preference setting to adjust the amount of data that can be stored locally. The cache is used primarily for traffic and map data, but can be used by any other plug-in via the cache service. Each module can create its own private context and store objects in there.

User Interface

The core plug-in defines the main user interface of the application and other plug-ins contribute by adding new menu items, windows, editors, etc. as necessary. The very flexible architecture of the RCP platform allows us to create a highly scalable user interface with rich widgets.[13] The design of the user interface follows the Eclipse User Interface Guidelines

[13]The complete explanation of the richness of Eclipse RCP UI is in [18].

([7]), with the architecture of each particular window adhering to the Presenter-First GUI design pattern from [1].

Engine

Engine is an interpreter of a scripting language. A script is essentially a set of commands that is used to automate or extend certain functionality. All scripts are evaluated in a shared context to which other plug-ins can contribute, thus making their functionality accessible. Script evaluation and contribution to the shared context is accomplished via the workbench engine service.

This module also contains all necessary UI elements for editing such a script, as well as a context explorer.

Map View

This component encapsulates all UI elements needed to provide the user with a visual map and geo-coding service. It provides a map view service that can be used by other plug-ins to change the current visible viewport and add or remove layers. A layer can be anything that is drawn on the top of the map using either geographical coordinates or fixed pixel locations. Examples of layers are digital map images, traffic visualizations, scale bars etc.

The UI also provides user with the possibility of explore a map and, in conjunction with the map service, the ability to visually define a route that can be used as an input to traffic services.

Charts

Charts are the major output of the application. In this plug-in two types of charts are defined: a time-space chart and a time-series chart, as defined in 2.3.2 and 2.3.3. This module does not register any services, all its functionality is provided as a public API. The plug-in defines adapters to adapt ([8]) traffic data, as defined in the core plug-in, for charts, using the traffic data part visitor.

Charts can be output using two different drivers: 1. an interactive chart in a dialog window, in which a user can explore different parts of the chart, and 2. an image buffer that can be saved to a file.

Report

The report plug-in contains a visual editor for creating and editing reports that can be scheduled for execution by the server-side report execution service. Currently there is only one report, which provides an overview of the traffic situation during a given time period along one or more routes, using one or more traffic sources. It can also be persisted to a disk in XML (Extensible Markup Language) format. A report item can be a chart or plain data exported to one of the supported formats. The report class encapsulates the report definition and is used as an input to the execution service. The output from an execution is a URL from where the completed report may be downloaded.

Chapter 5

Implementation Overview

In this chapter we will present an overview of concrete approaches taken and technologies used while implementing the system. We will also provide explanations for certain decisions that have been taken and issues that influenced them.

5.1 Lightweight Service Platform

The server side of Heavenly has been built as a lightweight, service oriented platform with a main focus on performance. It also provides a management interface which can be used to adjust the functionality exposed at runtime without the need for a server restart. All client requests are recorded in logs and service calls are transparently monitored so that any performance bottlenecks can be easily identified. An overview of server platform is shown in fig. 5.1.

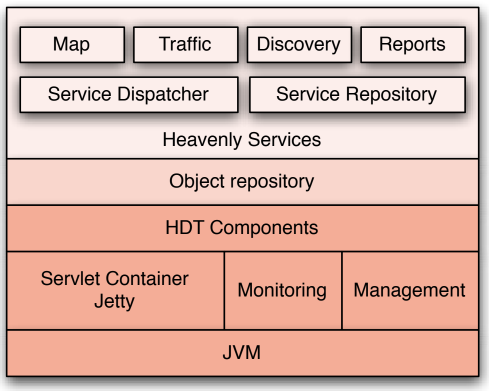


Figure 5.1: Server platform

5.1.1 Server

The server part of Heavenly is a web application and follows today's standards in Java enterprise development. The service dispatcher is implemented as a Java *servlet*,[1] by extending the `javax.servlet.HTTPServlet` class, which is the one of the mechanisms in the Java Enterprise Edition to provide dynamic functionality over the HTTP protocol. Like any other servlet application, it runs in a specific environment called a *servlet container*: a special kind of a web server that, amongst other things, supports execution of servlets and their life cycle and context management. There are many implementations of servlet containers available on the market at zero cost. We have chosen Jetty,[2] because it is very lightweight and simple to configure, yet it offers very good scalability using asynchronous Input/Output.[3]

5.1.2 Remoting

There are various remoting frameworks in Java that satisfy our requirements for inter-process communication between the client and server (sometimes referred as RCP - Remote Procedure Call) and that are able to use HTTP as the underlying courier. Our main constraint is on performance and data size, because the amount of data transited between client and server is vast. [27, 19, 10] provides a comparison of and benchmarks for various available solutions. Based on this evaluation, we decided to use HTTP Invoker,[4] because it has very good performance when dealing with large datasets and is the most transparent solution, as it does not impose any restrictions on the types involved besides serializability.

Because a lot of data is transported to the client a compression filter is used.

5.1.3 Management and Monitoring

The server is fully manageable via JMX[5] (Java Management Extension). Different aspects of services can be configured via the management extension and a good overview of system resource utilization is available. Also, alarms or other notification can be broadcast to any listening clients.

All service methods are transparently monitored and logged. This is achieved using the aspect oriented extension AspectJ.[6] Any service method annotated with `@MonitoredMethod` will automatically be monitored and all calls logged to the server log. Because aspects

[1]http://java.sun.com/products/servlet/
[2]http://www.mortbay.com/jetty/
[3]Asynchronous I/O, or non-blocking I/O, is a form of input/output processing that permits other processing to continue before the transmission has finished. ([34])
[4]http://static.springframework.org/spring/docs/2.5.x/reference/remoting.html
[5]http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/
[6]http://www.eclipse.org/aspectj/

can be inserted (weaved) into existing libraries as well, we are also able to monitor the performance of the HTTP Invoker.

5.1.4 Services

Services are mostly implemented using the APIs provided by different HDT components. The main challenge was the loading of digital maps, which is an extremely expensive operation as it requires a lot of resources and is very time consuming.[7] This is a big constraint especially during development and testing, because as the implementation changes often and the application is thus frequently restarted, this would release the map from memory and cause it to be loaded again. One of the solutions would be to work with a smaller map that loads faster. However, in our specific case this was not an option, because at least one full country is always needed to be loaded otherwise certain for all the functionality required for proper testing to be available.

Maps by their nature do not need to be reloaded every time the application changes because the map does not change (in our case, maps only change once every quarter). In order to avoid this map loading overhead, we built a very thin layer called *object repository* which is a simple object cache that allows us to persist objects over and beyond the application life-cycle. When the service application starts, it checks this repository for all available maps and registers them as map services. This approach also minimizes production downtime after upgrading to a newer versions of the application as there are multiple releases of the application for every map release. This solution is suitable for storing any object that is expensive to load.

The limitation of this approach is that only instances of classes on the server class path can be stored in the repository, because the classloader that has loaded the object repository classes has to have an access to all objects' classes to be should be stored.

5.2 Extensible Analysis Workbench

The client side of the application was implemented following the Eclipse RCP style of development, mostly inspired by the Eclipse code itself. In general, the implementation followed the rule that UI and non-UI code should be kept separate and an adapter pattern used to link UI functionality to non-UI objects, which allows for better separation and testing. We have also used the Spring framework for dependency injection (as in the server part) which enables us to externalize the component configuration and further simplified testing. Figure 5.2 shows implemented workbench plug-ins as a tree-map reflecting the size of individual module. In the following text we will briefly discuss some

[7]Depending on the number of countries loaded, a combined map of FR, DE, UK, CH and NL on average takes about 8-10GB of memory and around 30 minutes to load on eight-core CPU machine.

of the implementation details on the client side.

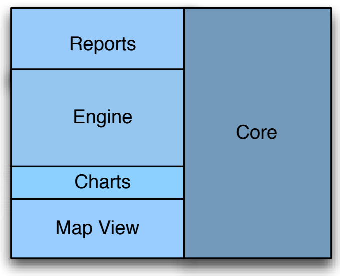


Figure 5.2: Workbench plug-ins represented in a tree-map, with the size denoting the size of the source code.

Service View

Besides the main application window itself, created following the Eclipse RCP practices, the major UI element is a service view. This view displays all known services in a tree and acts as the starting point for most user-initiated actions. Different plug-ins can contribute with special adapters that can affect the way a particular service is presented in this view as well as what actions should be available in the context menu shown after a user right-click.

Groovy Scripting

There are many different scripting languages that are built on top of the Java virtual machine. We considered JavaScript (Rhino)[8], Jython[9] and Groovy[10]. The reasons why decided to use groovy were that it is: 1. easy to learn, 2. easy to embedded into the application, 3. uses the native Java data types.

In summary, Groovy is a dynamically typed scripting language for the Java platform. Groovy scripts compile directly to Java byte code and can be loaded by the Java runtime. We have reused the Groovy Eclipse plug-in to include highlighting support into the scripting editor. The scripts are running in the same Java virtual machine as the application itself and there have to be a security manager installed in order to protect malicious code.

Visual Map

For a visual map component we chose World Wind[11], which is an open source 3D virtual globe developed by NASA (National Aeronautics and Space Administration) that provides

[8]http://www.mozilla.org/rhino/
[9]http://www.jython.org/
[10]http://groovy.codehaus.org/
[11]http://worldwind.arc.nasa.gov/java/index.html

a pluggable Java SDK. Currently, even though the SDK only exists as a preview version, it is the only visual map component we found that would provide advanced layering and reasonable performance when dealing with larger amounts of data.

The most interesting challenge here was creating a layer that could display the same map images as used in TomTom's web route planner. The route planner, like Google Map and other web map application, uses the Mercator projection to display the surface of a sphere (Earth) in two dimensions. This differs from World Wind, which is using the equirectangular projection ([15]). To solve this mismatch, we had to re-transform the images from one projection into the other on the fly using the NVIDIA texture tool[12] to convert PNG (Portable Network Graphics) images into DDS (DirectDraw Surface) textures.

Charting framework

Whilst there are a few charting frameworks available in Java, only JFreeChart[13] provides high-quality charts and decent performance. Implementing time-space series diagrams in JFreeChart proved very easy as this diagram is supported out of the box, so only few a modification were required. There were, however, no scattered plots, so the time-space diagram had to be implemented from the scratch. It uses an underlying image buffer, where the data are initially drawn in high resolution and then scaled and panned to approximately fit the target window area given the supplied ranges.

[12]http://developer.nvidia.com/object/nv_texture_tools.html
[13]http://www.jfree.org/jfreechart/

Chapter 6

Conclusions and Recommendations

6.1 Results

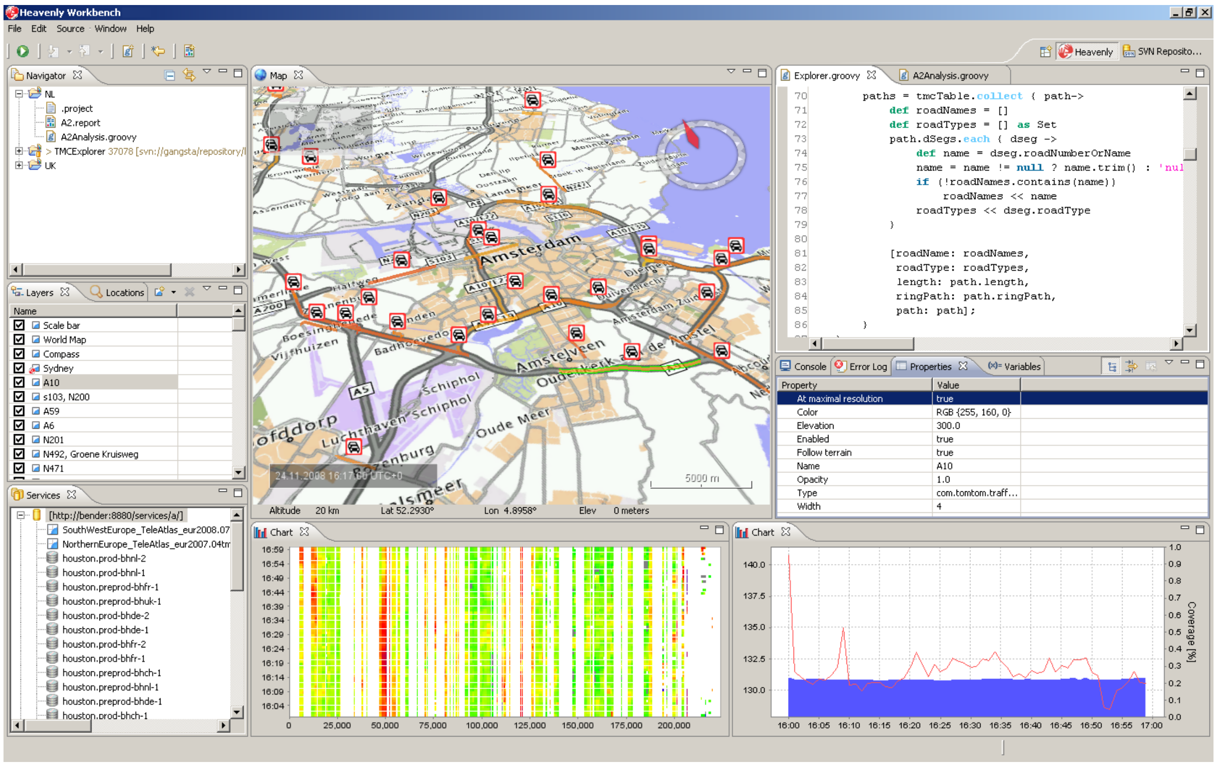
This thesis summarizes the essential theory and tools for traffic data analysis, that, combined, provide a solid environment for complex analysis of traffic information. It shows how a computer system based on this theory of traffic data analysis and capable of dealing with large datasets can be designed and implemented using today's cutting-edge technologies.

A complete computer system fulfilling the requirements for TomTom High Definition Traffic data chain analysis has been created. The software has been deployed and in use in a production environment from completion of the early prototypes onwards. The design is based on a service-oriented architecture combined with a modular approach, and has been implemented using a test driven development methodology. The characteristics of the Java language, together with the Eclipse integrated development environment, allowed for aggressive refactoring that, in combination with extensive usage of unit testing, has resulted in a stable and reliable implementation.

The system has successfully scaled to the current set of interacting users and, since it is designed so that a client can connect to multiple servers, data can be easily partitioned in support of future growth.

The proposed architecture allowed us to create an extensible geographic information system that is initially focused on providing a basic toolkit for traffic analysis, but could be also extended to other types of spacial content. It was developed in very tight integration with the HDT, but having flexible interfaces allows it to be adapted for use with other TIPs with only very minor changes.

Heavenly is in continuous use and the many analyses performed using it have greatly helped to improve the quality of the traffic information provided by TomTom HDT.

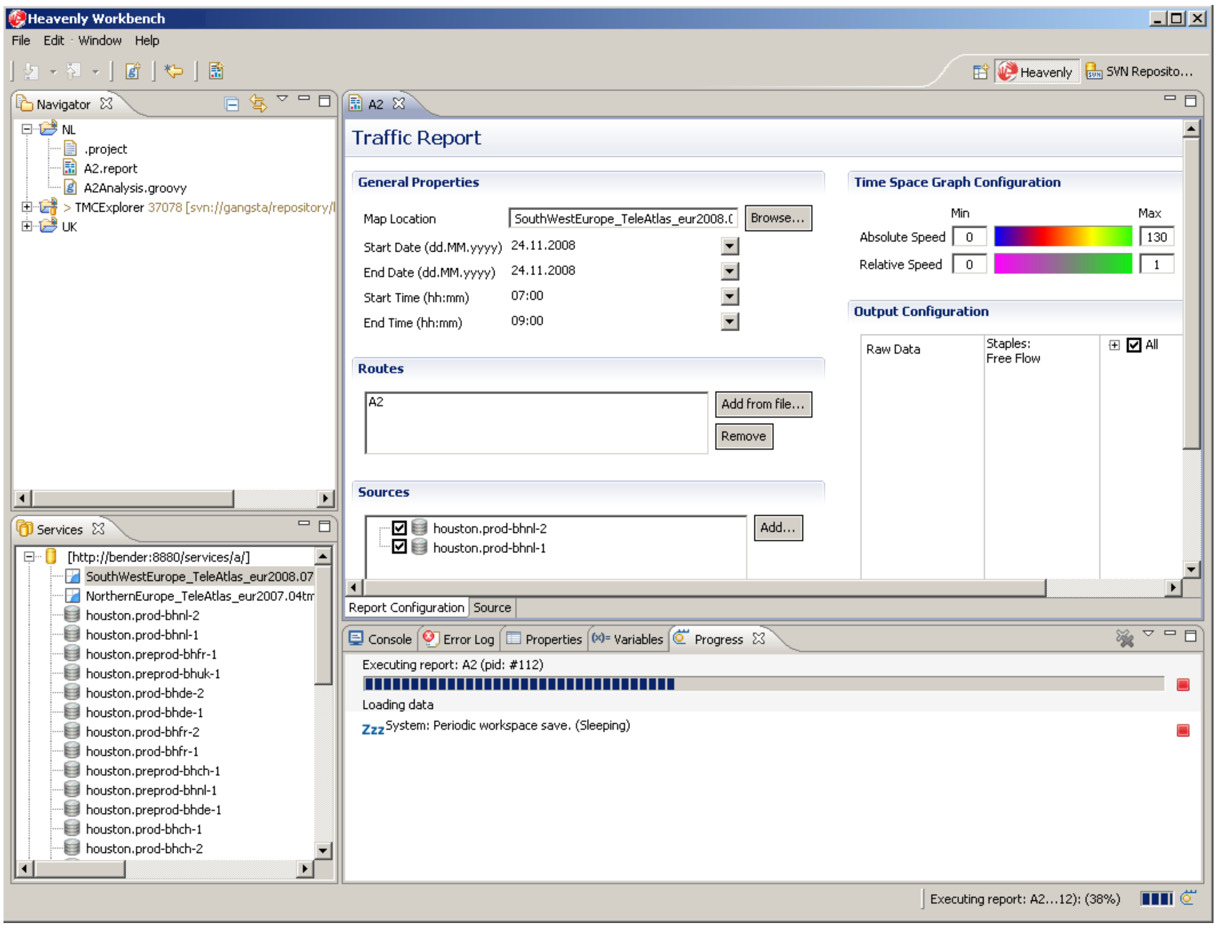


```
        paths = tmcTable.collect { path->
            def roadNames = []
            def roadTypes = [] as Set
            path.dSegs.each { dseg ->
                def name = dseg.roadNumberOrName
                name = name != null ? name.trim() : 'nul
                if (!roadNames.contains(name))
                    roadNames << name
                roadTypes << dseg.roadType
            }

            [roadName: roadNames,
             roadType: roadTypes,
             length: path.length,
             ringPath: path.ringPath,
             path: path];
        }
```

Property	Value
At maximal resolution	true
Color	RGB {255, 160, 0}
Elevation	300.0
Enabled	true
Follow terrain	true
Name	A10
Opacity	1.0
Type	com.tomtom.traff...
Width	4

(a) Analysis of afternoon rush hour traffic in the Amsterdam area. It shows a map, a scripting editor, a time-space and transit time chart and a service view.



Traffic Report

General Properties

Map Location: SouthWestEurope_TeleAtlas_eur2008.0
Start Date (dd.MM.yyyy): 24.11.2008
End Date (dd.MM.yyyy): 24.11.2008
Start Time (hh:mm): 07:00
End Time (hh:mm): 09:00

Routes

A2

Sources

☑ houston.prod-bhnl-2
☑ houston.prod-bhnl-1

Time Space Graph Configuration

	Min		Max
Absolute Speed	0		130
Relative Speed	0		1

Output Configuration

Raw Data | Staples: Free Flow | ☑ All

Executing report: A2 (pid: #112)

Loading data

Zzz System: Periodic workspace save. (Sleeping)

Executing report: A2...12): (38%)

(b) Report editor and remote report execution progress.

Figure 6.1: Screen-shot of the final user interface of the client application.

6.2 Further Work

The application is completely implemented, tested and documented and is used daily by traffic engineers and analysts at TomTom. At the same time, there are lots of areas for further research and development. In this section we will give a brief overview of possible directions for future work.

The system is now used for traffic data analysis, but the strong modularization allows us to extract the core functionality and form a well defined platform for building other special data analysis applications within the TomTom domain. As there will be more and more dynamic content provided by TomTom to its customers that will need to be analyzed, this platform should provide basic facilities for data manipulation and analysis in general, with a focus on simplified development in order to help developers in other teams adopt Heavenly and extend it to meet their own requirements. There could be additional components based on this platform not just for analysis of different content, special tools might also be built for other types of user. An example might be a real-time congestion management system that would allow HDT operators to affect the running system by creating or removing congestion events in cases of system misbehaviour.

From an economic point of view, there is a big opportunity to transform this application from being an internal tool for TomTom employees into a fully-featured commercial product that can be provided to other parties and generate revenue. Heavenly might become TomTom HDT's initial bridge into the government market, with the Dutch Government as a potential first customer. This application can become a valuable tool for analysis and visualization for all consumers of HDT data. One can even think about specific customer releases with custom functionality, creating some sort of product ecosystem around Heavenly.

From a traffic analysis perspective, as the HDT service expands into more and more countries, there should be a major focus on maximizing automation. The Heavenly workbench should allow users to create more advanced reports, modifiable templates and flexible scheduling. Another very interesting problem will be looking into new ways of traffic data visualization and, by using pattern recognition and machine learning, providing automated detection of possible system failures, as well as advanced modeling for better simulation and system overall tuning.

The data generated by TomTom High Definition Traffic, together with this application, provide a great opportunity to learn a lot about traffic flow, its patterns and behaviour.

Bibliography

[1] M. Alles, D. Crosby, C. Erickson, B. Harleton, M. Marsiglia, G. Pattison, and C. Stienstra. Presenter first: Organizing complex gui applications for test-driven development. *Agile International Conference*, 2006.

[2] H. Bar-Gera. Evaluation of a cellular phone-based system for measurements of traffic speeds and travel times: A case study from israel. *86th meeting of the transportation research board*, 2007.

[3] V. Basis. Traffic flow theory. Technical report, Katholieke Universiteit Leuven, 2002.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Professional, 2005.

[5] M. A. Chowdhury and A. W. Sadek. *Fundamentals of Intelligent Transportation Systems Planning*. Artech House Publishers, 2003.

[6] C. F. Daganzo. *Fundamentals of Transportation and Traffic Operations*. Pergamon, 2003.

[7] N. Edgar, K. Haaland, J. Li, and K. Peter. *Eclipse User Interface Guidelines*. Eclipse Foundation, 2004.

[8] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[9] D. C. Gazis. *Traffic Theory*. Springer, 06 2002.

[10] D. Gredler. Java remoting: Protocol benchmarks. http://daniel.gredler.net/2008/01/07/java-remoting-protocol-benchmarks/, 2008.

[11] N. W. Group. Rfc2616 - hypertext transfer protocol – http/1.1. Technical report, The Internet Society, 1999.

[12] C. Hecht. Coopers wireless interface requirements and state of the art in traffic content coding. Technical report, COOPERS, 02 2007.

[13] B. Kerner. *The Physics of Traffic*. Springer, 2002.

[14] L. A. Klein. *Sensor Technologies and Data Requirements for ITS.* Artech House Publishers, 2001.

[15] P. Longley, M. F. Goodchild, D. Maguire, and D. Rhind. *Geographic Information Systems and Science.* Wiley, 2nd edition, 2005.

[16] S. Lorkowski. *Fusion von Verkehrsdaten mit Mikromodellen am Beispiel von Autobahnen.* PhD thesis, TU-Berlin, 2008.

[17] S. Maerivoet, S. Logghe, and G. Gates. Validation of travel times based on cellular floating vehicle data. *6th European Congress and Exhibition on Intelligent Transport Systems and Services*, 2007.

[18] J. McAffer and J.-M. Lemieux. *Eclipse Rich Client Platform.* Addison-Wesley, 2005.

[19] Nominet. Middleware remoting protocol migration. http://blog.nominet.org.uk/tech/2007/03/13/middleware-remoting-protocol-migration/, 2007.

[20] OSGi Alliance. *OSGi Specification R4.1*, 2007.

[21] P. Přibyl. *Inteligentní dopravní systémy a dopravní telematika.* Vydavatelství ČVUT, 1st edition, 2005.

[22] J.-S. Pyo, D.-H. Shin, and T.-K. Sung. Development of a map matching method using the multiple hypothesistechnique. *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, 2001.

[23] G. Rose. Mobile phones as traffic probes: Practices, prospects and issues. *Transport Reviews - A Transnational Transdisciplinary Journal*, 26(3):275–291, 05 2006.

[24] R.-P. Schäfer, S. Lorkowski, and P. Mieth. Taxi fcd - jam detection by means of floating car data. Technical report, DLR, 2006.

[25] H. Schilling. *Route Assignment Problems In Large Networks.* PhD thesis, TU-Berlin, 2006.

[26] C. Schneebauer and M. Wartenberg. On-the-fly location referencing methods for establishing traffic information services. *Aerospace and Electronic Systems Magazine, IEEE*, 22(2):14–21, 02 2007.

[27] C. Technology. Metaprotocol taxonomy. http://hessian.caucho.com/doc/metaprotocol-taxonomy.xtp, 2008.

[28] K. Tödter and G. Wielenga. Eclipse rcp vs. netbeans platform: A developer's insights. http://blogs.sun.com/geertjan/entry/eclipse_platform_vs_netbeans_platform, 09 2006.

[29] TomTom. Satellite navigation has a positive effect on driving and traffic safety, 07 2008.

[30] Transportation Research Board. *Highway Capacity Manual*, 12 2000.

[31] A. Tuzar, P. Maxa, and V. Svoboda. *Teorie Dopravy*. Vydavatelství ČVUT, 1st edition, 1997.

[32] K. E. Wiegers. *Software Requirements*. Microsoft Press, 2003.

[33] G. Wielenga. How to choose a java desktop framework. *DZone Java*, 08 2008.

[34] Wikipedia. Asynchronous input/output. http://en.wikipedia.org/wiki/Asynchronous_Input_Output.

[35] Wikipedia. High cohesion. http://en.wikipedia.org/wiki/High_cohesion.

[36] Wikipedia. Scalability. http://en.wikipedia.org/wiki/Scalability.

[37] Y. Youngbin and C. Randall. Field operational test using anonymous cell phone tracking for generating traffic information. *Transportation Research Board 85th Annual Meeting*, 2006.