STEFAN KAUFMANN

OPENING PUBLIC TRANSIT DATA IN GERMANY

OPENING PUBLIC TRANSIT DATA IN GERMANY

STEFAN KAUFMANN

A Status Quo

Diplom Informatik (Dipl.-Inf.)
Institut für Datenbanken und Informationssysteme
Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm

2014-05-23

Stefan Kaufmann: *Opening Public Transit Data in Germany,* A Status Quo

Written as a requirement for the completion of the diploma course in Media Informatics.

SUPERVISORS:
Prof. Dr. Manfred Reichert
Prof. Dr. Frank Kargl

ADVISOR:
Dipl.-Inf. Rüdiger Pryss

LOCATION:
Institute of Databases and Information Systems
Faculty of Engineering and Computer Science
89081 Ulm, Germany

SUBMITTED:
2014-05-23

This thesis is dedicated to my parents, who provided me with the guidance I needed to grow as a child, and the freedom I needed to grow as an adult—including the patience during my long years of studying.

Also, this is to the one left behind. Thank you for letting me learn from you, FloD.

ABSTRACT

Open data has been recognized as a valuable resource, and public institutions have taken to publishing their data under open licenses, also in Germany. However, German public transit agencies are still reluctant to publish their schedules as open data. Also, two widely used data exchange formats used in German transit planning are proprietary, with no documentation publicly available. Through this work, one of the proprietary formats was reverse-engineered, and a transformation process into the open GTFS schedule format was developed. This process allowed a partnering transit operator to publish their schedule as open data. Also, through a survey taken with German transit authorities and operators, the prevalence of transit data exchange formats, and reservations concerning open transit data were evaluated. The survey brought a series of issues to light which serve as obstacles for opening up transit data. Addressing the issues found through this work, and partnering with open-minded transit authorities to further develop transit data publishing processes can serve as a foundation for wider adoption of publishing open transit data in Germany.

ZUSAMMENFASSUNG

Open Data kann als wertvolle Ressource angesehen werden. Auch in Deutschland verbreiten öffentliche Einrichtungen zunehmend Daten unter freien Lizenzen. Bei der Veröffentlichung ihrer Fahrpläne als Open Data zeigen sich deutsche Verkehrsverbünde und -unternehmen jedoch immer noch zurückhaltend. Zwei der im deutschen ÖPNV verbreiteten Datenaustauschformate sind zudem proprietäre Formate, für die keine Dokumentation öffentlich verfügbar ist. In dieser Arbeit wurde eines dieser Formate analysiert und ein Transformationsprozess für dieses Format in das offene GTFS-Format entwickelt. Dieser Prozess ermöglichte es einem Verkehrsunternehmen, seine Fahrpläne als Open Data zu veröffentlichen. Darüber hinaus wurden durch eine Umfrage unter Verkehrsverbünden und -unternehmen in Deutschland die Verbreitung verschiedener ÖPNV-Datenformate und Vorbehalte gegenüber der Veröffentlichung offener Fahrplandaten erfragt. Die Erhebung identifizierte eine Reihe offener Probleme, die der Veröffentlichung offener Fahrplandaten im Wege stehen. Die Bearbeitung dieser Probleme, sowie eine Kooperation mit aufgeschlossenen Verkehrsverbünden zur weiteren Entwicklung von Veröffentlichungsprozessen können als Grundlage dienen, die Veröffentlichung offener Fahrplandaten zu fördern.

ACKNOWLEDGMENTS

I would like to thank my advisor Rüdiger Pryss, and my principal supervisor, Manfred Reichert, for allowing me to pursue this topic and giving me the liberties to explore and learn. I also thank Frank Kargl for agreeing to be secondary reviewer for this thesis, and for being the academic host to the datalove open data student working group at Ulm University.

Special thanks go to SWU Verkehr and DING, especially Christian Burst and Martin Schiller, who were willing to spend *a lot* of their time on patiently explaining me the internals of their respective data flows, and providing me with the data sets to work on.

Further thanks go to my proofreaders, in alphabetical order, Doreen Butze, Benjamin Erb, Rens van der Heijden, Barbara Körner, Simon Lüke, Nikola Mattschas, Juliane Wessalowksi and Björn Wiedersheim, and everybody else who helped and supported me during my thesis writing period. This also includes the members of the OKF Open Transport group who inspired me to choose this topic for my thesis, and Florian Schaub, wo convinced me to actually do so.

Finally, I would like to thank my friends, who helped to shape who I am today.

Thank you!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AEG	Allgemeines Eisenbahngesetz
API	Application Programming Interface
AVM	Automated Vehicle Management
AVL	Automated Vehicle Location
BART	Bay Area Rapid Transit
CASSIOPE	Computer-Aided System for Scheduling Information and Operation of Public Transport in Europe
CEN	European Committee for Standardization
CRS	Coordinate Reference System
CSA	Connection Scan Algorithm
CSV	Comma Separated Values
CF	Compact Flash
DOM	Document Object Model

DELFI	Durchgängige Elektronische Fahrplaninformation
DING	Donau-Iller-Nahverkehrsverbund GmbH
DINO	DIVA Datenpool Nord
DIVA	Dialoggesteuertes Verkehrsmanagement- und Auskunftssystem
DRIVE	Dedicated Road Infrastructure for Vehicle Safety in Europe
EC	European Community
EFA	Elektronische Fahrplanauskunft
ERM	Entity Relationship Model
GDF	Geographic Data File
GIS	Geographic Information System
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GTFS	General Transit Feed Specification
HAFAS	HaCon Fahrplan-Auskunfts-System
HTML	Hypertext Markup Language
IATA	International Air Transport Association
IBIS	Integriertes Bus-Informations-System
IFOPT	Identification of Fixed Objects in Public Transport
ICTS	Intermodal Transport Control System
IP	Intellectual Property
KML	Keyhole Markup Language
MDT	Mobile Data Terminal
NVBW	Nahverkehrsgesellschaft Baden-Württemberg
O-D	Origin-Destination
ODbL	Open Database License
OKF	Open Knowledge Foundation
ÖPNV	Öffentlicher Personennahverkehr
PersBefG	Personenbeförderungsgesetz

PDF	Portable Document Format
RAPTOR	Round-bAsed Public Transit Optimized Router
SIRI	Service Interface for Real Time Information
SQL	Structured Query Language
SWU	Stadtwerke Ulm
VBB	Verkehrsverbund Berlin-Brandenburg
VDV	Verband Deutscher Verkehrsunternehmen
VVS	Verkehrs- und Tarifverbund Stuttgart
VÖV	Verband öffentlicher Verkehrsbetriebe
WGS84	World Geographic System 1984
XML	Extensible Markup Language

1 INTRODUCTION

In 2010, publisher Tim O'Reilly coined the expression "Government as a Platform" [68], in which he argues for a shift in providing government services. Instead of what Donald Kettl had earlier called "vending machine government" [52]—tax money is inserted, and services pop out of this metaphorical machine—O'Reilly proposes a government model in which, "[one] thought of government as the manager of a marketplace". Just like the open source software development community is likened to the image of a bazaar in *The Cathedral & the Bazaar* [72], government, in his opinion, should be thought of as a place where "the community itself exchanges goods and services".

Taking the success of computer platforms as an example, O'Reilly advocates, among other steps, for (1) the implementation of open standards, (2) using simple systems and allowing them to evolve, (3) designing for participation and allowing for the adoption of outside expertise, and, (4) lowering the barriers to experimentation.

An often cited example for a well done implementation of this platform concept is Apple's iPhone. Unlike other phone ecosystems, which only included applications designed by the phone vendor and chosen partners, it expressively allowed and published the necessary tools for anybody to develop apps for their platform. The result was the first line of smartphones that became an application platform, spawning a cottage industry of application developers that had published over a million apps in Apple's App Store by 2013 [60].

Note that Apple has rejected apps from its store for a number of not always transparent reasons, including the app of a Pulitzer-winning political cartoonist [81].

1.1 TURNING GOVERNMENT INTO A PLATFORM THROUGH OPEN DATA

Adapting Apple's principle to public administrations, proponents argue that governments should not use the data at their disposal as input to finished services they deliver to citizens, but should see open data as the finished product to serve to their end-users instead [49]. Definitions of open data have been proposed by the Open Knowledge Foundation (OKF)'s *Open Definition* [35] and the Sunlight Foundation's *Ten Principles for opening up Government Information* [37].

For works to be "open" according to these definitions, they need to be available to everybody and as a whole for no more than their reproduction costs, and re-distribution, modification and the creation of derivates need to be permitted. Furthermore open file formats must be used, the license must not discriminate against anybody, and alternative use must be permitted [35].

Von Lucke and Geiger summarize these definitions in the context of open data stemming from the public sector [39]:

> Open Government Data are all stored data of the public sector which could be made accessible by government in the public interest without any restrictions on usage and distribution.

Such data, when opened, can be used by innovators in order to implement new solutions, and to re-engineer processes and services. On their basis, engaged citizens and developers can contribute new applications and services, even for rather specialized use cases [39, 22].

1.2 OPEN DATA IN PUBLIC TRANSIT

In this work, the author focuses on transit data as an example of open government data, which has been quoted by Headd as "the clearest example of how open government data can be used to encourage the development of useful new applications" [49]. In the United States, governments had released transit data in the General Transit Feed Specification (GTFS) format, which sparked the development of numerous new transit apps. Since those applications all hinge around the same, freely available transit data specification, they can be redeployed to any given transit system, whenever a transit agency decides to release their schedule in the GTFS format. Headd argues that, while many transit agencies continued to design, develop and deliver their own transit apps, this development allowed agencies to economize on this task. Instead, transit agencies could choose to focus on their task of efficiently transporting their ridership—while riders could pick a solution that meets their specific requirements from the new app market.

Even though, in Germany, public transit has been de-regulated in the 1990s to allow for open competition between privately and publicly owned transit operators [11], it is (a) a public service required by law [1, Sec. 1][5, Sec. 8] and (b) heavily subsidized by federal, state and municipal governments [71, P. 287–290].

The history and structure of GTFS is further elaborated on in Section 3.5.

MORE FLEXIBILITY Any given app—relying on a common data format—could work in *all* transit systems for which open transit data has been released. In contrast, journey planning applications released by individual agencies tend to work only for the specific transit agency they were released for, or for a limited set of transit systems.

While relying on online journey planner interfaces instead of open transit data, the Android application *Öffi* [78] serves as a case in point for many German transit users. With Öffi, riders can use the same user interface to plan their public transit journey for 23 local transit systems in Germany, as well as for 13 local transit systems in other countries. Additionally, it allows journey planning for long-distance public transit in Germany, Switzerland, Austria, Belgium, Luxembourg, Denmark, Sweden, Norway, Poland, and the United

Kingdom. Thus, users do not have to care about what agency will provide their public transit when travelling to other cities—instead, the same, familiar workflow can be used in all cities covered by Öffi, although only separately for each transit authority.

SPECIALIZED SOLUTIONS AND ACCESSIBILITY Publicly available transit data also allows for the development of applications for very specific use cases. While transit agencies usually need to deploy generalized solutions that meet the demand of the majority of their customer base, they generally do not have the ressources to also release applications relevant only for small subsets of their riders.

One example is the author's university campus in Ulm. As of 2014, the campus is served by no less than seven bus routes and another night bus service, spread over five bus stops—however, not all routes service all of these five stops. The walking distance from the southernmost bus stop on the campus, using the public sidewalks *around* the university buildings, to the bus stops along the northern campus is, however, greater than nine minutes. Thus, the transit agency's algorithm will often dismiss connections departing from bus stops at, respectively, the other part of campus, depending on the bus stop chosen in the trip planner. However, most riders departing from the university will start their trip from *within* the university complex, where both parts of the campus are roughly the same walking distance away. Therefore, riders need to check connections between multiple bus stops and their desired destination to actually arrive at the most time-efficient bus route.

While students and university employees would benefit from an application that takes this fact into account, the transit agency can neither simply change the routing algorithm for their official trip planner—since it would require casual riders from outside the university to have access to the university buildings during transit operating hours—nor can it publish a customized trip planner for the university by itself, since the small set of affected riders would not justify such an investment.

Machine-readable transit data can also allow for more accessible trip planning services for riders with disabilites. Although German law requires transit authorities and operators to provide completely accessible transit services by 2022 [5, Sec. 8], a 2012 study in the German states of Niedersachsen and Bremen found only 10% of all railway stations to be accessible to persons with disabilities [62]. While transit data alone will not remedy structural barriers, such as missing lifts, specialized applications can, for instance, notify visually impaired riders about upcoming stops where loudspeaker announcements are not available, or guide their transfers through audible instructions [56]. Other existing applications include GPS-equipped mobile devices with braille displays, which allow users to find the near-

est transit stop in a series of US transit systems [10]. Despite current efforts not only to let riders plan accessible itineraries, but also to offer accessible web sites for doing so [61, P. 8], information continues being "trapped" within Portable Document Format (PDF) documents, which are often in-accessible to customers relying on screen-readers to access the information [55].

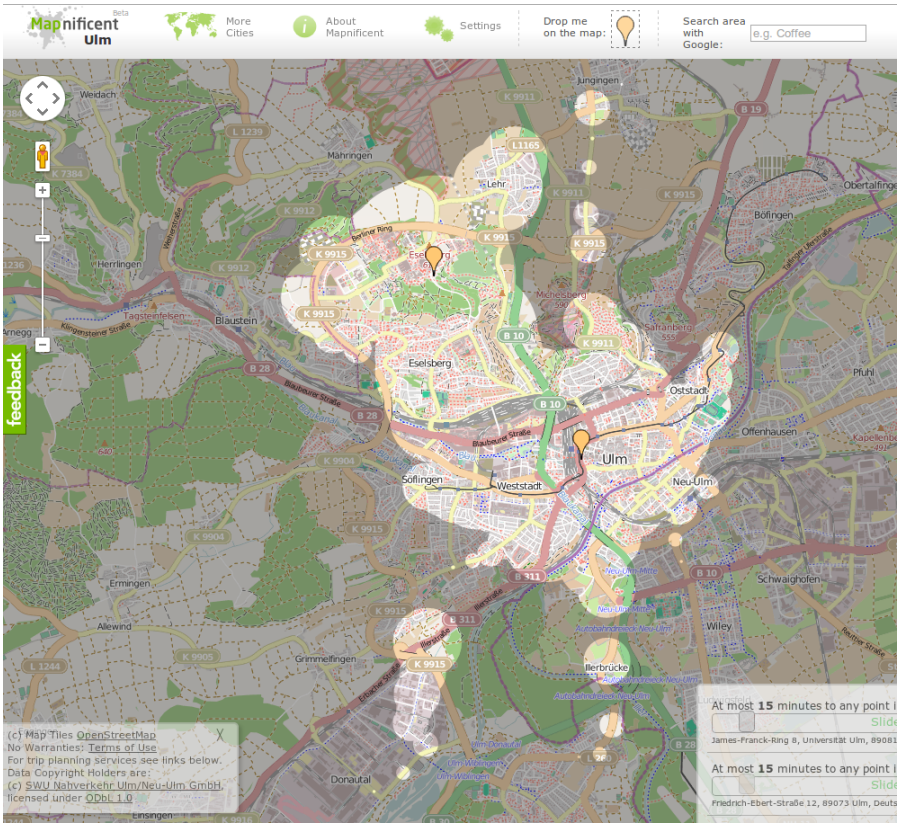


Figure 1: Screenshot of Mapnificent,[88], showing the intersection of public transit reach from Ulm central station and the university within 15 minutes each. The underlying data is the open GTFS data provided by Stadtwerke Ulm.

Mapnificent [88] is another example for a use case usually not covered by transit authorities' applications or trip planners, which can, however, be easily implemented in any transit system offering their schedule in GTFS format. It shows possible transit trips for a given start point and maximum travel time, also allowing the placement of multiple starting points and areas on which the travel radii intersect. Thus, several riders spread throughout an area could choose a venue where they could meet within a given time frame, using public transit and their respective starting locations.

IMPROVING CONVENIENCE While online trip planning services allow riders to make optimal decisions concerning their public transit itineraries without even having to consult printed timetables [84], and

one can argue that this is another form of giving relevant information to riders that serve to improve ridership [33], it still proves hard to plan door-to-door itineraries spanning over several transit systems or countries, or including more than one mode of transportation.

This is all the more important in rural areas, where citizens heavily rely on their cars as a means of transport, and unwieldy itineraries make riders percieve their cars to be more convenient than the often sparse public transit coverage. While standardization efforts are underway in order to facilitate seamless journey planning through the DELFI[1] and EU-Spirit[2] initiatives, today's online journey planners often cannot even calculate fares for itineraries depending on journeys in multiple transit systems.

Also, it is still not possible for most journey planning services offered by transit authorities to optimally take riders' existing mobility options into account. For instance, students of Ulm University could book a round-trip from Ulm to Stuttgart and back by supplementing their DING semester ticket with an inexpensive Verkehrs- und Tarifverbund Stuttgart (VVS) day pass, which includes the usage of all VVS lines for that day. This would allow stop-to-stop trips from anywhere in Ulm to any place in Stuttgart.

However, riders have to be aware of this fact when booking their ticket. The journey planner of Deutsche Bahn offers a two-way ticket, valid only for the train to and from Stuttgart, and, alternatively, a state day pass, which is valid for all regional trains and select *Verbund* lines in the state of Baden-Württemberg during that day. Both options are, however, more expensive than the aforementioned VVS ticket, and both come with different feature sets. To make matters worse, riders can buy the cheaper VVS ticket *only* through the VVS journey planner—neither Deutsche Bahn's nor DING's journey planner can even *display* the price for individual trip fares within the VVS area.

The same problem applies to new forms of multi-modal travel, e. g., using a private car to get from a smaller town to the next train station, riding a train into a larger city, and changing mode to a car-sharing solution in that city. Ideally, all thinkable modes of transport could be integrated into a single journey planning application, including privately owned vehicles, public transit, car- or bike-sharing solutions, taxis, or any other thinkable mode.

ADVANCING SCIENTIFIC PROGRESS While the problem of finding the shortest path between two nodes in a graph has been solved efficiently in principle since Dijkstra published his now classical algorithm in 1959 [30], Sanders and Schultes argue in [77] that, until 2005, speedup techniques for road network routing algorithms were difficult to compare:

1 http://www.delfi.de/
2 http://eu-spirit.eu/

> [S]tudies were either based on very small graphs or on proprietary data that could not be used by other groups. In particular, for 'newcomers' it was almost impossible to start working in this field.

After the road network of the United States was extracted out of US Census data [20], and the road network of Western Europe was made available to the scientific community by the German company PTV AG, Sanders and Schultes found [77] that variants of these graphs had been used for most comparison studies—for instance, the results of the 2005 DIMACS implementation challenge [8]. Sanders and Schultes "view it as likely that the sudden availability of data and the fast rate of innovation since then are not a coincidence".

However, as pointed out by Bast, "the algorithmic problem of computing the fastest way to get from A to B is [...] surprisingly different on road networks than on public transportation networks." [12], making public transit routing significantly slower than road network routing:

> [A] change of vehicle takes time, and we want to penalize paths with many changes of vehicle—two issues that do not arise in road networks.

A free/open source implementation of RAPTOR has since been published by bliksem labs [17]

It was not until Delling's Round-bAsed Public Transit Optimized Router (RAPTOR) in 2012 [28] and the more recent Connection Scan Algorithm (CSA) [29] that public transit routing algorithms became on a par with road network routing—without requiring substantial pre-computation as an earlier approach by Bast had [13].

However, the scope of scientific research into public transit routing does not end with the routing algorithms per se. Colpaert argues that, using linked open data, a multitude of other factors could be included into route planning, such as weather conditions, street construction, etc. [23]. Brosi presented a transit live map based on GTFS data he claims is able to display vehicle movements for the whole world [19]. Analogous to the case presented by Sanders and Schultes, it can be argued that easy access to open transit data could allow more researchers to apply themselves to the advancement of this topic, leading to faster innovation.

1.3 AIMS OF THIS WORK

Despite all the presented cases, as of mid-2014, Stadtwerke Ulm (SWU) Verkehr and Verkehrsverbund Berlin-Brandenburg (VBB) are the only German transit institutions offering open transit data to interested developers, and one more transit authority stated to the author they would follow suit in the near future. Meanwhile, German railway operator Deutsche Bahn and a series of transit authorities have been

implementing processes to export their schedules into the GTFS format, but provide them exclusively to Google [73].

This work aims to provide an overview over the data and data models commonly found in German public transit, as well as a process description for transforming one proprietary German industry format into GTFS. It also aims to provide a better understanding of the export capabilities and the mindsets of German transit authorities towards publishing their schedules as open data, and common obstacles identified by stakeholders towards doing so.

1.4 OUTLINE

After analyzing the data involved in the planning and operating of public transit in Chapter 2, transit data models commonly used in Germany are presented and compared in Chapter 3. The author then presents two approaches he developed together with local transit providers on how to transform transit data from a proprietary format into GTFS, in Chapter 4 and Chapter 5. Finally, special consideration to the reasons for the reluctant adoption of open transit data in Germany is given in Chapter 6, starting with a look at the legal ramifications of doing so, and analyzing current hurdles and problems through a survey undertaken with 47 transit institutions in Germany.

2 TRANSIT DATA

The planning and operation of public transport necessitates, generates and uses a plethora of data. Roach describes in [75] and [74] a number of domains relevant for transit data processing in the early 1990s, namely scheduling, passenger information, Automated Vehicle Management (AVM), fare collection as well as personnel disposition or driver management.

An process incorporating all these domains is described by Ceder in [21, Ch. 1.2] as a sequence of the following steps:

A tabular overview over planning activities according to Ceder is included in Table 8.

NETWORK DESIGN Land-use characteristics, authority constraints and current ridership are taken into account, the latter broken down to time-of-day and day-of-week. As a result, the names and physical locations of *stops and stations* are defined, often including more detailed information concerning specific platforms within each stop. Pairs of stops are connected to make up a *transit network,* with accurate descriptions of the distances between each pair of stops, both in a spatial as well as in a temporal sense, depending on the way taken between stops and the mode of transport used.

TIMETABLE DEVELOPMENT Using the previously defined transit network, *lines* are created to group transit services using a distinctive set and sequence of stops serviced by all vehicles within that group. Each line is assigned an identifier, e. g., Bus #8, or the Jubilee Line. Deviations within lines are possible. For instance, a line might encounter a high average ridership between a series of inner-city stops, while ridership from there onwards is comparatively low. In order to tackle this problem, every other vehicle might *short turn* [21, P. 56], i. e., turn around at the last of the high-ridership stops, while other vehicles follow the complete stop pattern. Another approach might be a design branching after the last high-volume stop, with vehicles alternating between two low-volume branches after that stop. Both short turns and which branch is serviced by an individual vehicle should be made aware to riders through outer displays on the vehicles and published timetables.

Depending on the vocabulary used, a line might be called a "journey pattern" or a "route". Even more confusingly, those names might also be used to describe subsets of what is called "line" here. See Section 3.7.1 for more information on this.

VEHICLE SCHEDULING Taking into account the times necessary to prepare a vehicle for another journey once it has reached the final stop, deadhead times from garages and depots to journey start locations and from journey end locations to garages and depots, and deadhead times between journey end and start locations, vehicles can

be assigned to *blocks* or *chains* (German: *Umlauf*, see [21, Ch. 7]). For instance, a vehicle could be assigned to follow a certain line end-to-end for a number of times, then deadhead to a depot for refueling and start service on another line afterwards. This process can be optimized towards a balance of keeping the fleet size as small as possible and minimizing deadheading distances. At the end of the process is a *working timetable* with all on-duty journeys and deadheads, defining which vehicle is where at what time. Note that the timetables made available to the public are usually sub-sets of these working timetables, excluding deadheads and not showing blocking.

CREW SCHEDULING After vehicle blocks have been created, crews (drivers) have to be assigned to them. Since from the transit agency's perspective, driver's wages are usually the largest single-cost item, commercially available transit scheduling software often heavily focuses on crew-scheduling activities [21, Ch. 10]. This personnel disposition data is usually restricted to the use by the transit operator itself and will not be shared with other agencies and even less with the public.

FARE STRUCTURE While not explicitly covered by Ceder, a fare structure is usually established—either just for the individual transit operator, or, more common in Germany, for a complete linked transit system. The fare is based on travel classes, usually derived from the travelled distance or the number of predefined spatial zones the journey passes through. Additionally, surcharges may be raised for certain product classes, e. g., express lines.

For instance, German railway operator Deutsche Bahn calculates the standard fare based on the product class (regional trains, InterCity or InterCityExpress trains) and the traversed distance, usually defining a corridor between certain stations within which any possible path can be chosen.

This usually results in a Origin-Destination (O-D) matrix defining the transit classes for each O-D pair, and a pricing index defining the tariffs per transit class for single fare tickets, transit passes and, if applicable, special fares (e. g., group passes, tickets for elderly riders, etc.)

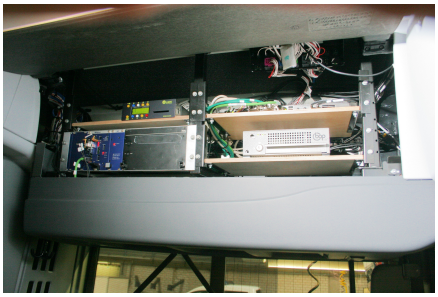
2.1 USAGE OF DATA DURING OPERATION

Once the schedule, vehicle and crew disposition as well as fares have been defined, subsets of this data are usually provided to diverse technical systems.

PRINTED TIMETABLES The information necessary for typesetting printed timetables can be exported from the planning software, for instance in some kind of spreadsheet or Comma Separated Values (CSV) format. This data can be used in desktop publishing software to create line-based or stop-based printed timetables, or both. Line-based timetables list all departures and/or arrivals at all stops serviced by a certain line, while stop-based timetables list all departures and/or ar-

rivals of all lines servicing a certain stop. Alternatively, the planning software automatically typesets line- or (mostly) stop-based timetables in a ready-to-print format that requires no or little subsequent manual treatment.

(a) Testing setup employed by SWU. The IBIS rack is at the bottom; from left to right: GSM module, IBISplus computer with connected CF storage and GPS antenna, power supply, and RF transceiver. The upper rack emulates vehicle functions, e. g., the door opening mechanism. On top sits the driver's MDT user interface.



(b) Installation in a bus, above the driver's seat. The IBISplus computer rack is the lower left part, complete with voice radio equipment. Above sits the destination display controller. On the right side, controllers for the bus's CCTV and passenger entertainment system are installed.

(c) The MDT user interface located within the driver's view, currently showing deviation from the scheduled departure time, and upcoming stops. At stops, guaranteed transfers can request the driver to wait for delayed vehicles.

Figure 2: IBIS on-board computers.

ON-VEHICLE COMPUTERS On many German public transit buses, on-board *Integriertes Bus-Informations-System (IBIS)* computers link various systems that make use of the stop pattern, e. g., destination signs, inner signs (showing the next stop or stops), automated recorded stop

announcements, ticket validators, switch control, and automated passenger counting systems [32, P. 8]. After their vehicle has been supplied with the appropriate data, drivers can cycle through all the journeys within their individual vehicle block by means of a user interface terminal located next to their seat (see Figure 2). Making use of the established distance between stops, and monitoring the bus's odometer and door-opening mechanism, it is possible to determine the remaining distance to a stop, or whether one journey on a block has been finished. Thus, the next stop can be announced via inner signs and recorded announcements, and matching destination sign information for the journey can be displayed. Positional precision throughout the journey can be enhanced through infrared beacons located at known positions—usually, lamp posts—and GPS receivers (see Figure 3). Odometer inaccuracies or smaller detours, e. g., because of construction sites, can be coped with by relying on the door-opening mechanism, which resets the distance counter to zero upon application. Combining the odometer approach with GPS capability can also be used to detect such deviations. The driver can also correct for errors manually through the IBIS computer's user interface.

(a) On-board infrared transceiver. (b) Roadside infrared beacon.

Figure 3: On-board and roadside infrared transceivers and beacons for enhanced positional accuracy

If applicable, passenger count information can be extracted from the IBIS system and be fed back to the scheduler in order to evaluate the passenger load and adjust the schedule accordingly, if necessary (See Figure 2.2 for more on this).

Furthermore, buses can interact with junction processors via infrared or radio transmission in order to request a green light from the signal controller. If an authorized vehicle approaches the intersection, it registers with the appropriate junction processor, which then decides whether to prioritize the bus over "ordinary" traffic. After the bus has passed the intersection and deregistered, the signal controller returns to its normal signal phasing [32, P. 4]

TICKET VENDING MACHINES If the transit operator intends to sell tickets based on origin and destination of their passengers, ticket vending machines—both stationary ones at stops or other points of

sale, and in-vehicle units—need to be supplied with the necessary fare structure data in order to calculate the matching price. If prices need to be determined for specific connections—i. e., a specific journey on a specific line that is served by different product classes with different pricings—, the necessary routing data must be supplied as well.

ROUTING Using the transit network data, electronic journey planners can provide riders with an itinerary for their intended journey. This service can be a web application, a telephonic interactive voice response system, or a sub-set of a ticket vending machine's capabilities. Depending on the type of journey planner used, an itinerary can include only one or several modes of transport, the latter being called an *intermodal journey planner*. Also, journey planners differ in whether they allow routing from any given point to any other given point or whether start and destination locations have to be stops on a network.

In order to allow for longer journeys being routed, transit operators and authorities may exchange their network and schedule data with others. For instance, transit authorities over a larger geographical area might partner and aggregate all their data, either for use within their own electronic journey planners—if existing—or in order to let others provide an electronic journey planner for the complete, aggregated area.

2.2 DATA ACQUISITION

For the scope of this thesis, we will assume an already established and running transit system, not one built from scratch. This means that data concerning land use, authority constraints, ridership, and previous vehicle and crew disposition is based on empirical knowledge, leaving only "data maintenaince", for instance, establishing new or changed lines or stops.

OBTAINMENT BY TEST CAR AND AVM Any data referring to the *physical locations* of stops, depots, relief-points and garages, as well as the *layout of lines* can be obtained by transit agencies' test cars equipped with highly accurate GPS receivers. In Germany, transit agencies usually employ test vehicles outfitted with an IBIS computer, including an infrared transceiver for interacting with roadside beacons. O-D matrices between stops subsequently allow for the planning of journey patterns for individual lines.

Additionally, AVM systems can contribute to this data by giving additional information on, for instance, systematic and unanticipated delays during traffic peak times, or long layover times during off-peak hours. If present, ridership can also be estimated, either using optical

Figure 4: Test car used by SWU Verkehr. The vehicle is outfitted with two IBIS computers in the back, one in the version used for city buses, one in the version for regional buses (lower left image). In order to accurately measure positions of, and distances between stops and infrared beacons, an IR transceiver sits on the vehicle's roof (lower right image).

sensors located at the vehicle doors, or measuring the vehicle's payload. The analysis of this statistical data can help identify unexpected side effects of, e. g., badly-timed traffic lights, as well as changes in ridership patterns.

2.3 EXCURSUS: REAL-TIME DATA

Automated Vehicle Location (AVL) systems can also contribute their data in order to inform a control centre and riders about deviations from the planned schedule. While originally intended to improve operational efficiency, transit agencies have later used this information to display it to passengers, e. g., through real-time information displays at transit stops, or through Application Programming Interface (API)s. Dziekan and Kottenhoff argue that "the mere existence of such a system creates a general sense of trust in the public trans-

port system", and real-time information reduces the perceived wait time by riders at transit stops significantly [31]. Ferris later extended on this work, finding compareable results with real-time information presented to the riders on mobile devices instead of fixed displays [87].

While this extension to the described transit data is therefore interesting and relevant to transit users, the focus of this work is on bulk target schedules, i. e., *complete* schedules as they were *planned* by a transit agency.

In the following chapter, relevant data models for the exchange of this data will be examined.

3 DATA MODELS

In Europe, the development of data models for data exchange within and in between transit operators began in the late 1980s, resulting in the ÖPNV-Datenmodell in Germany and CASSIOPE out of the EC DRIVE 1 research initiative. Both influenced the pan-european TRANSMODEL as part of the EC DRIVE 2 initiative, which was ultimately standardized as EN12896:2006 [4]. TRANSMODEL as a reference data model served as a basis for European standard implementations, such as the TRANSXCHANGE standard used for bus schedules in the United Kingdom, and the SERVICE INTERFACE FOR REAL TIME INFORMATION (SIRI) standard, which, however, deals with real-time schedule information.

Despite all these different standards, vendor-specific data models still play a prominent role when encountering route network and schedule data. On the German market, HAFAS and DIVA are two of the major software suites used by public transit agencies—one using a documented exchange format, the other a format with no publicly available documentation whatsoever.

Despite there never being one transit data model adopted by any world-wide regulatory body, in recent years, GTFS has become some kind of de-facto standard widely used within the open data community. First developed by Portland's TriMet transit agency together with Google, it is now used by Google's Transit Planner as well as a growing number of transit application by third parties.

3.1 THE VDV TRANSIT DATA MODEL (ÖPNV-DATENMODELL)

In the late 1980s, the German association of transport companies, Verband Deutscher Verkehrsunternehmen (VDV)—which went by the name Verband öffentlicher Verkehrsbetriebe (VÖV) back then—started standardizing data layouts and formats in order to facilitate the exchange of schedules within and between transit authorities in West Germany.

The stated goal was to increase efficiency in transit planning and cost savings, as pointed out by Goetz in [42]. The BISON research project—started in 1980 and funded by the German Federal Ministry of Research and Technology (see [51])—defined domains which were found to be suitable for computer-aided planning and management processes. Subsequently, different vendors started implementing software systems for indidivual segments—thus, different products emerged, each covering only a slice of all required functionality, with little or no interoperability between products from differ-

ent vendors. As transit operators called for a modular concept with compatibility between individual parts regardless of the software's manufacturer [42], VÖV started developing a data model out of the BISON data model, which was originally meant to be implemented as a central data storage within a relational database [41, P. 1]. The resulting ÖPNV-DATENMODELL was subsequently standardized within the *VDV-Schriftenreihe*.

Today, the ÖPNV-Datenmodell encompasses several data standards within the VDV-45x series, including the management of duty rosters (VDV-455), realtime schedule data (VDV-453, -454 and -459), infrastructural data (VDV-456) and automatic passenger counting systems (VDV-457 and -458). We will limit our attention to the exchange format first standardized in 1991 with VDV-451, as well as the network and target schedule description defined in VDV-452 since 1999.

3.1.1 *VDV-Schrift 451: File Layout*

The goals of the data exchange format definition according to VDV-451 [85] are:

- importing or exporting data from or to another hardware or software platform
- editing, inspecting or evaluating data, using standard software such as text editors or spreadsheet software

The format is based on plain text files, which are named according to DOS specifications (8+3 letters in lower case) and a defined naming pattern:

- Prefix:
  - "i" (for *interface file*)
  - three-digit number designating the data table contained within the file
  - the creation date as a three-digit Julian day number
  - a trailing zero
- Suffix:
  - ".x10"

Following this pattern, an example exchange file name would be `i4853580.x10` for a file containing table #485 according to VDV-451, created on the 358th day of the year.

Alternatively, files can also be named by the name of the table it describes, i. e., `rec_frt.x10` for a file containing the table named `REC_FRT`.

All entries within the file start with a three-letter descriptor indicating the meaning of the following line, separated by a semicolon.

Each file contains a file header, the table definition—table header, records, table trailer—and a file trailer.

The "file header" must not be confused with a file signature, which describes a file type by using a magic number.

FILE HEADER The header part describes conventiones necessary for correctly parsing the body, i. e., the table records, of the file. It specifies the date and time format and whether or not the record columns follow a fixed-width ("aligned") or free format (`mod`), information about the data source (`src`), the character set (`chs`), as well as version numbers of the source software (`ver`), the interface being used (`ifv`), and the data set itself (`dve`). A file format field is specified, but not used as of version 5.0 (`fft; ""`).

As of ÖPNV-Datenmodell 5.0, only ASCII and ISO-8859-1 can be used as character sets.

Imports must accept both the fixed-width as well as the free format. The fixed-width format was supposed to allow for easier visual inspection of raw exchange files by use of a text editor, since the columns line up on the screen.

Listing 1: Example file header in VDV-451 format

```
mod; DD.MM.YYYY; HH:MM:SS; aligned
src; "OneBusAway VDV Exporter"; "01.01.2014"; "00:00:00"
chs; "ISO8859-1"
ver; "0.1"
ifv; "1.0"
dve; "201401010000"
fft; ""
```

TABLE DEFINITION The table itself is represented as CSV with a semicolon (`;`) as field delimiter and double quotation marks (`"`) as text delimiters, all in conformance with RFC 4180 [79]. It is introduced by stating the table name (`tbl`), column designators (`atr`) and data types (`frm`), followed by a set of records (`rec`), and concluded by a table trailer specifying the number of records within the table (`end`). Comments (`com`) can be placed within the table name definition and the table trailer.

Listing 2: Example table in VDV-451 format

```
tbl; EXAMPLE_TABLE
atr; ID; TEXT; NUMBER
frm; num[6.0]; char[8]; num[3.0]
rec;       1;"Bake 1"     ; 22
rec;       2;"Bake ""2""a";  8
com; This is a comment
end; 2
```

FILE TRAILER A simple `eof`, followed by the number of tables within the file, marks the end of the exchange file. Since exchange files according to VDV-451 must contain exactly one table each, this sequence always reads as "`eof; 1`".

3.1.2 *VDV-Schrift 452: Standard Interface for Network Plans and Schedules*

Within this model, "schedule" describes the employee timetable, i. e., the complete schedule of any vehicle from the time it leaves a depot to its return, including layovers, dead mileage, etc.

The VDV Standard Interface for Network Plans and Schedules (*VDV-Standard-Schnittstelle Liniennetz/Fahrplan*) as standardized in VDV-452 [86] describes an Entity Relationship Model (ERM) for a route network and the corresponding schedule. It is meant to be implemented in a Structured Query Language (SQL) database, but can also be exported or imported by means of exchange files according to VDV-451.

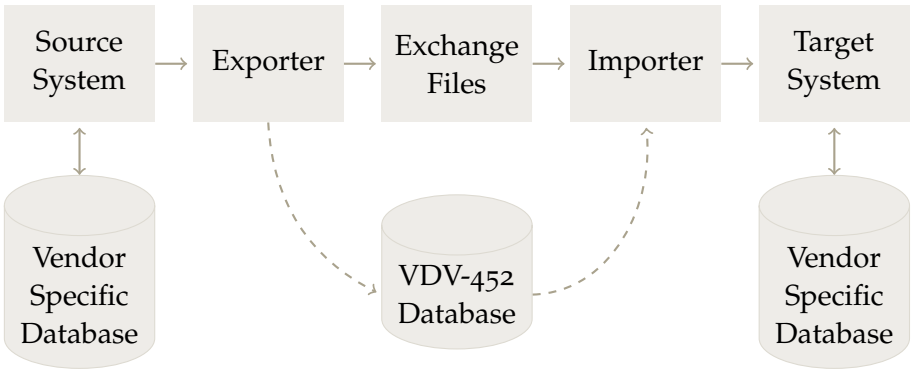


Figure 5: Data flow model using VDV-452 as exchange format between systems using vendor-specific databases. Modelled after [86, P. 14]

The standard encompasses the following items in it's current version 1.5:

- Calendar dates (what service ID runs on which dates and when they are in service)
- Operational data (vehicle types, recorded stop announcements, headsign texts)
- Location data (stop areas, stop points, beacons, depots)
- Transport network data (lines, distances, journey time groups, journey times, stops)
- Route data (routes and route shapes)
- Schedule data (journeys and stop times dependent on journey types, blocks)
- Connection definition data for guaranteed transfers

The standard's main goal is the seamless exchange of route and schedule information within different software and hardware platforms, e. g., exporting a working schedule for usage within an Intermodal Transport Control System (ICTS), or supplying ticket vending machines or IBIS computers with data sets necessary for their operation.

TABLE NAME	DESCRIPTION
BASE_VERSION_VALID	Validity starting date of referenced version
BASE_VERSION	Version identifier
PERIOD	Assignment of day types to calendar dates
DAY_TYPE	List of all day types (i. e., weekday, day before a holiday, etc.)
POINT_TYPE	List of point (location) types, e. g., stops, stations, traffic lights, depots, etc.
STOP_TYPE	List of location groups, i. e., stops and depots
STOP_POINT	Definition of location points where passengers usually (dis)embark. These points can be grouped within STOPs
ACTIVATION_POINT	Definition of other locations, e. g., beacons. Has a 1:1 relation to a STOP
STOP	All locations within the transit network, i. e., stations, depots or activation points
VEHICLE	Description of vehicles
TRANSPORT_COMPANY	Lists all transport companies
OPERATING_DEPARTMENT	Operating department (e. g., Bus, Metro, Subway, etc.)
VEHICLE_TYPE	Description of vehicle types, i. e., articulated or standard buses
ANNOUNCEMENT	List of announcement texts
DESTINATION	List of possible destination signs
LINK	Directional edges between pairs of points, with distance in meters
POINT_ON_LINK	Definition of intermediary points. Allows for defining geographically accurate display of a route.
TIMING_GROUP	Definition of timing pattern groups
WAIT_TIME	Waiting times per timing pattern group and location

TRAVEL_TIME	Scheduled travel time per link and timing pattern group (might differ over the course of a day)
DEAD_RUN	Directional edges between pairs of points for deadheads
DEAD_RUN_TIME	Driving time per deadheading edge and timing group (might differ over the course of a day)
JOURNEY_TYPE	List of journey times, e. g., on-duty trip, deadheads from a depot, to a depot, or between stops
ROUTE_SEQUENCE	Sequence of stops serviced by a route
LINE	Definition of a route and assignment to an operating department
JOURNEY	Definition of a trip
JOURNEY_WAIT_TIME	Trip-specific wait times per stop
BLOCK	Vehicle blocks, from leaving a depot until arriving at a depot

Table 1: List of data tables specified in VDV-452 [86].

3.2 UNITING EUROPEAN STANDARDISATION EFFORTS: TRANSMODEL

Starting in 1988 with the Dedicated Road Infrastructure for Vehicle Safety in Europe (DRIVE) programme, the Commission of the European Community (EC) endeavoured to improve road safety, improve transport efficiency and reduce environmental pollution through a series of research projects [14]. Within DRIVE, the Computer-Aided System for Scheduling Information and Operation of Public Transport in Europe (CASSIOPE) project provided a first specification of functional requirements for an Integrated Road Transport Environment and defined the "approach to standards and protocols for the higher communication levels" within this environment [84].

Before the end of the DRIVE programme in 1992, its Requirements Board recommended extending the findings, e. g., through field trials [14, P. 252], resulting in the DRIVE 2 programme. Within DRIVE 2, the EuroBus (1992-1994) and Harpist (1995) projects further developed the CASSIOPE findings and included the experiences with VDV's ÖPNV-Datenmodell. In order to standardize the previously parallel development of CASSIOPE and ÖPNV-Datenmodell, both project working groups were considered subgroups of Working Group 3 within

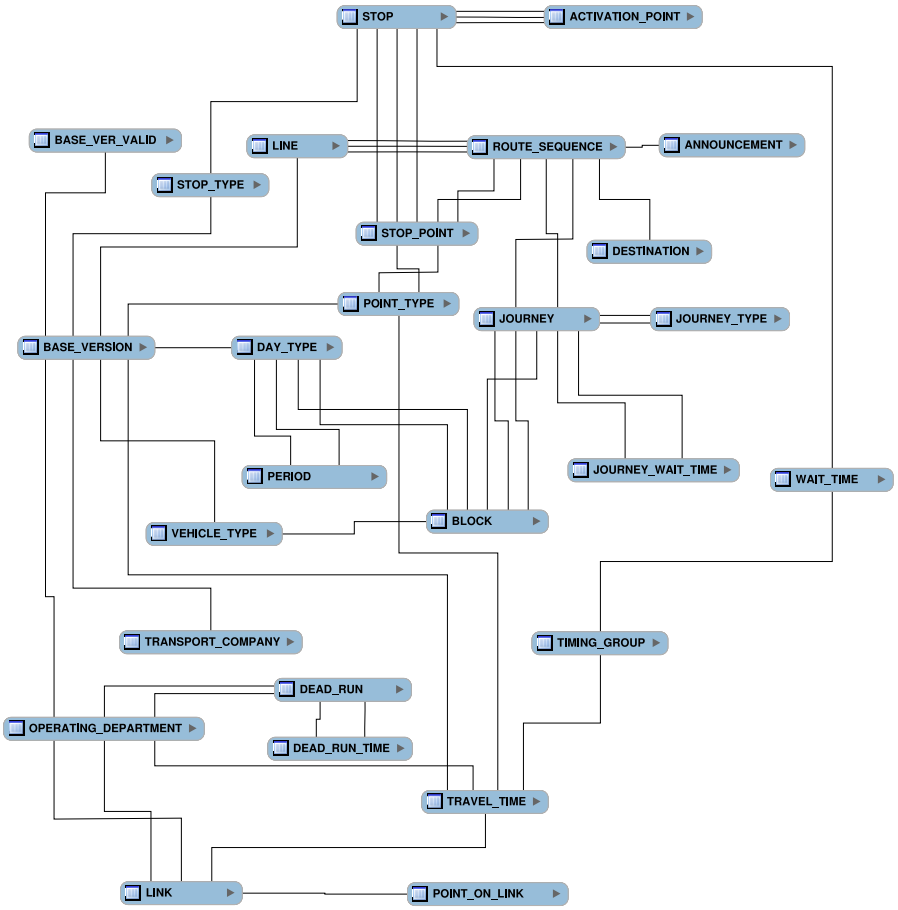


Figure 6: Entity relationship diagram of the VDV-452 data model.

the technical committee 287 of the European Committee for Standardization (CEN) [18]. CEN/TC287 is "the technical committee responsible for the development of standards related to Public Transports" [80, P. 4], and the findings of EuroBus and Harpist were presented as a pre-standard to the committee at the end of 1995 [18]. After incorporating feedback, version 4.1 of TRANSMODEL was published in 1996 and formalized into European prestandard ENV 12896 in August 1997 [16, 18]. Since this point in time, TRANSMODEL serves as "a pan-european reference model for public transport operating companies" [34, Ann. 2].

FEATURES In its revised version V5.1 adopted in 2001 [18] and standardized as EN 12896:2006 [4], the datamodel includes the following elementary data and domains:

- Network description
- Versions management
- Tactical planning (Vehicle scheduling, driver scheduling, rostering)

- Personnell disposition
- Operations monitoring and control (including AVM)
- Passenger information (offline and online)
- Fare collection
- Management information and statistics

The standard also takes multi-modal transport and the modelling of multiple operators into account. While it is focused mainly on bus operation, trolley bus and light rail (i. e., tramway or metro) transport modes are also addressed [18].

Earlier versions of the standard also included references to the Geographic Data File (GDF) [3] standard and "link[s] with traffic and road data" [34].

IMPLEMENTATIONS Since TRANSMODEL is only a reference data model [15], it has led to the development of several concrete Extensible Markup Language (XML)-based implementations, including the TRANSXCHANGE standard in the United Kingdom [66], the Identification of Fixed Objects in Public Transport (IFOPT) standard for identifying fixed, transport-related objects [6], and SIRI as a standard to exchange real time information [7]. Additionally, a TRANSMODEL-based XML schema for exchanging GTFS stop and schedule information exists [53].

3.3 GERMAN INDUSTRY STANDARDS: HAFAS

The origins of the HaCon Fahrplan-Auskunfts-System (HAFAS), developed and distributed by *HaCon Ingenieursgesellschaft mbH* in Hannover, date back to 1988. Back then, HaCon developed a first electronic journey planner as part of Deutsche Bahn's *Kurs 90* project, which was able "to load the complete [Deutsche Bahn] schedule onto a screen within six seconds" [65].

HAFAS became the electronic journey planner of choice for Deutsche Bahn, first as stand-alone software that customers could download and use on their Personal Computers, later as a online web service. Today, according to HaCon, HAFAS software is used throughout "more than 100 installations in nearly 20 countries" [47].

3.3.1 *Features*

HAFAS depends on a server installation which processes the relevant information and offers interfaces to compatible output channels. HaCon advertises the following data exchange features:

- Scheduling information, including network and geography data
- Real-time data
- Data exchange through matching modules for ticketing, GIS services and fare calculation

Possible interfaces for delivering the data include Internet frontends, mobile clients for transit personnel, print output clients, offline clients and smartphone applications.

3.3.2 *HAFAS Exchange Format*

HaCon developed their own, proprietary data format for exchanging data between HAFAS installations, the *HAFAS Rohdatenformat*. Until the Swiss SBB decided to publish their HAFAS-based raw schedules information in 2014 [50], no specification of the format was publicly available—apart from occasional excerpts that had found their way onto the Internet and were exchanged between open transit data developers [89]. While it can not be ascertained that the documentation provided by SBB covers the *general* workings of the HAFAS Rohdatenformat, or just a variant used by SBB, it offers some insights into the principal data layout.

Data is stored in text files encoded according to ISO-8859-1 and `CRLF` line terminators, with no file or column headers introducing the data that follows. Instead, each line represents a record, and follows a fixed-column format where fields are padded with spaces, if necessary. Comments are introduced by a `%` sign, and all input after them is ignored.

As an example, the `BFKOORD_GEO` file defines stop coordinates in the WGS84 Coordinate Reference System (CRS), and follows the following pattern:

- a seven-digit integer, identifying the stop, in columns 1–7,
- the X coordinate, in World Geographic System 1984 (WGS84) format (`NNN.nnnnnn`), in columns 9–18,
- the Y coordinate, in WGS84 format (`NNN.nnnnnn`), in columns 20–29,
- the Z coordinate, in metres above sea level, in columns 31–36 (left justified)
- a comment with the stop name (only for improved legibility), in columns 38pp.

As can be seen in the example excerpt in Listing 3, the X and Y coordinates are justified as if the leading zeroes were present.

Listing 3: Excerpt from the stop coordinate example file provided by SBB

```
0000168   7.589551  47.547405 277    % Basel SBB
0001522   8.310170  47.050170 436    % Luzern
0001560   9.529195  47.003835 504    % Maienfeld
0001714  11.558334  48.140232 0      % München Hbf
8503424   8.632728  47.698282 404    % Schaffhausen
```

The practise of explaining each line's contents with comments at their end is used throughout many of the files provided by SBB, although the documentation suggests this is merely an option for better legibility, and not expressly required.

See Listing 4 for an example.

While most of the files provided by SBB define stops, their coordinates, vehicle types, transfer information, etc., the actual schedule information resides in a single `FPLAN` file. Journeys are defined by a series of initialization lines, e. g., an initial line starting with `Z*`, defining the journey number, operator, variant of the journey and, optionally, journey frequencies. After the journey has been parametrized through these lines, the journey pattern is defined line by line, each line starting with the stop identifier, the stop's name (optional, for better legibility), arrival and departure times, and, optionally, new journey or operator identifiers if they should change at that stop.

Listing 4: Excerpt from the schedule example file provided by SBB

```
1  *Z 19704 000065   001                                  % 19704
       000065   001 (001)
2  *G SN  8503424 8014558  00110  00130                    % 19704
       000065   001 (002)
3  *A VE 8503424 8014558 348970  00110  00130             % 19704
       000065   001 (003)
4  *A Z  8503424 8014558         00110  00130             % 19704
       000065   001 (004)
5  *R                                                     % 19704
       000065   001 (005)
6  8503424 Schaffhausen                 00110             % 19704
       000065   001 (006)
7  8014487 Herblingen            00113  00113             % 19704
       000065   001 (007)
8  8014490 Thayngen              00118  00119             % 19704
       000065   001 (008)
9  8014491 Bietingen             00121  00121             % 19704
       000065   001 (009)
10 8014492 Gottmadingen          00124  00124             % 19704
       000065   001 (010)
11 8014558 Singen (Hohentwiel)   00130                    % 19704
       000065   001 (011)
```

While the SBB export does not provide any information on several items found in VDV-452 or TRANSMODEL, due to the functionalities advertised on HaCon's website and the widespread use of HAFAS throughout German public transit authorities, it is safe to as-

sume HAFAS is capable of modelling and exporting a number of them, and the SBB export is merely a limited subset of HAFAS's capabilities. However, due to lacking publicly available documentation, no definitive conclusion can be drawn on this.

3.4 GERMAN INDUSTRY STANDARDS: DIVA

The Dialoggesteuertes Verkehrsmanagement- und Auskunftssystem (DIVA) schedule management system was first developed by Munich-based *Mentz Datenverarbeitung GmbH* in 1979 and has been marketed to transport operators and authorities ever since [58]. Today, it is used by operators and authorities throughout Europe, Australia and the USA [59]. In Baden-Württemberg, 9 out of 22 transit authorities are using DIVA, including Nahverkehrsgesellschaft Baden-Württemberg (NVBW), a subsidiary of the State's Ministry for Transportation and Infrastructure acting as a coordinator between the state's *Verkehrsverbünde*.

See [71] for details on the Verkehrsverbund *principle.*

It is both used by SWU Verkehr—the transit branch of the municipal works run by the twin cities of Ulm (Baden-Württemberg) and Neu-Ulm (Bavaria)—as well as by Donau-Iller-Nahverkehrsverbund GmbH (DING), the transit authority responsible for the linked transport system encompassing Ulm and Neu-Ulm as well as the surrounding districts..

An electronic journey planning system, Elektronische Fahrplanauskunft (EFA), is also marketed by Mentz. It is tailored to receive data exported from DIVA in a vendor-specific format that is not publicly documented.

3.4.1 *Features and Data Exchange Compatibility*

The main features of DIVA, as advertised by Mentz, are:

- Timetable management
- Timetable and vehicle scheduling
- Optimisation of vehicle schedules
- Duty scheduling
- Duty schedule optimisation
- Personnell deployment
- Geography
- Cartography
- Generation of timetable books and posters

- Transfer optimisation
- Import and export plugins for a variety of exchange formats
- Data supply for onboard systems (e. g., IBIS computers)
- Data export for Mentz's journey planner, EFA

Not all versions of DIVA offer all these features. For instance, Mentz offers a variety of data importers and exporters, e. g., for GTFS, VDV-452, VICOS LIO. However, each and any of these import and export plugins must be licensed separately. In 2013, a single license for the DIVA2VDV exporter was quoted at 14 100 EUR, with the 2nd to 5th license costing 7 700 EUR each[1]. Thus, DIVA and EFA in their basic variant form a closed ecosystem co-dependent on each other, relying on either using both Mentz's products or licensing additional interfaces to be able to export the data for third-party route planning applications.

3.4.2 *Internal Raw Data Format*

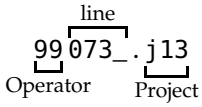
Since Mentz offered no documentation of the format, all findings within this chapter are based on the author's dissemination of schedule exports made available by DING and SWU. Note that this analysis is not necessarily a comprehensive documentation of the format, as it may differ between individual deployments of DIVA.

For data exchange between DIVA installations, raw data can be exported and then re-imported into another DIVA installation through a "DIVA2DIVA" plugin. In contrast to the exchange format used by HAFAS and described in Section 3.3, this exchange format is not documented and not intended to be used by third parties[2]. The data formatting being used in most of the files is akin to the exchange format of VDV-451: Data is stored in plain text files, with tabular definition following the syntax of VDV-451. However, each text file may contain several tables, some with primary and foreign keys and no information whatsoever about the relationship between these tables. For instance, DING provided the author with a file export containing all stop data, `haltestellen.ding`, which defines 32 distinct tables, 11 of which contain no records at all. Of the remaining 21 tables, 19 tables have a foreign key defined and 4 tables have a primary key. While primary keys are always called `_AutoKey_`, and foreign keys `_FK__AutoKey_`, the file contains no further explanation or clues as to the relationship of the tables.

FILE NAMING PATTERN Most of the files are using the German term of their content, or abbreviations thereof, e. g., `tgtyp` for *Transportgefäßtyp*[3]. In contrast, files starting with numerical identifiers appear to exclusively model information concerning a specific line. The

1 Reference: E-Mail exchange between DING and Mentz DV, October 2013.
2 Reference: E-Mail exchange between the author and Mentz DV, 2013-10-07: "The [DIVA] data format is way too complex and a purely internal format. For data exchange with other systems, we use [...] standardized formats such as VDV-452 [...]."
3 Vehicle type

naming pattern of those line descriptions is similar to the pattern described in the CSV exports used in Section 4.1.



The first two digits of the file name are the numerical identifier of the operating department responsible for the respective line. The following four characters are the line number itself.

Line numbers may be followed by distinguishing suffixes, e. g., differentiating between seasonal variants of the same line.

In most cases, when a file extension is being used, it describes a logical assignment. For instance, the extensions of the lines designate the *Project* they are part of—namely, a specific schedule with a validity start and end date.

A comparable pattern can be found for the `teilstrecken`[4] files, which uses their corresponding operating department identifiers as file extensions. Within the SWU exports, a combination can even be found: The files defining *Fahrzeitprofile*[5], *Fahrzeitgruppen*[6], and *Haltezeiten*[7], are subsequently appended with the operating department *and* project they describe or are part of, respectively.

3.4.3 *File headers*

Files containing tabular data begin with a header similar to that of VDV-451. Several header entries appear to be directly based on their respective VDV-451 counterparts, although with a minor change in one instance: The `mod` entry denotes *only* the modification time, not the formatting of the file. As in VDV-451, a `chs` field denotes the character set used within the file. The `usr` field appears to make use of the domain and user name of the system the data is exported from. In place of the `src` field found in VDV-451, an `exe` entry seems to denote the data source software, and `ver` should, again, refer to the source software version.

See Section 3.1.1 for comparison.

Listing 5: Typical DIVA file header

```
FORMAT000005;
mod;15.08.2012;15:41:26
usr;\\DING-ULM\Diva
cpt;NTDIVAAPP
exe;DIVA Grunddatenserver (3, 5, 23, 2)
lib;14
mrv;1
dvn;7
dfm;5
```

4 Partial sections
5 Travel time profiles
6 Travel time groups
7 Layover times

```
ver;0
chs;ISO_LATIN_1
sep;0x3b
eoh;
```

The author was not able to make more than educated guesses about the rest of the header entries. In between files, the first line alternated between various versions, from `FORMAT000001` to `FORMAT000050`, although no meaning could be derived from this entry so far. The rest of the entries appear to refer to version numbers of libraries—and `eoh` marks the end of the file header.

3.4.4 *Line Definition Files*

Within DIVA's exchange format, each line is completely defined in a single text file that follows a formatting pattern unique to those files.

As can be seen in Listing 6, after an initial header, each line starts with a two letter combination which appears to serve as a identifier of the dataset type in this line. The text files containing line definition data use a different format altogether.

See Chapter 5 for further reference.

Listing 6: Excerpt of DIVA line definition file for SWU bus No. 15

```
1  FORMAT002009000000000023 "\\DING-ULM2\\Doe" 27.03.13  14:49:220
       BW00000000000000000000
2  EMYYYYYYYYYq08YY0YYYYNY
3  FW012H10501058104810591060106110791078108711601241 1240
4  WdH    S   S
5  WaH
6  WzH            000000000
7  WlH
8  MSH01    01    01        01
9  STH0120001    0011    0021    0031    0041    0051    0061
       0071    0081    0091    0102    0112
10 ZZH
11 PFH  02
12 FTH00001120002*0601020504020101N0265000 -099900000000000000
13 FTH00002120002*0601020607020101N0265000 -099900000000000000
14 FAH005150000000001051510000110         0              000000000
         000015003300015010000000000026415001-0999N   0
               000
15 FAH006150000000001061510000110         0              000000000
         000015004800015010000000000026415001-0999N   0
               000
16 FAH007250000000001072510000210         0  N           000000000
         000015001800015010000000000026415001-0999N   0
               000
17 FAH008250000000001082510000110         0  N           000000000
         000015002500015010000000000026415001-0999N   0
               000
```

```
18 FAH00925000000001092510000110         0  N           000000000
          000015005700015010000000000026415001-0999N   0
                000
19 UEHO1 000000 "" 08126L03       Y00000[...]
20 UEHO2 000000 "" 08245L03       N00000[...]
21 EEH "Universität Süd" 005150000000000000_0000000000000000000
22 EEH "Universität Süd" 006150000000000000_0000000000000000000
23 EEH "Universität Süd" 007250000000000000_0000000000000000000
24 EEH "Universität Süd" 008250000000000000_0000000000000000000
25 EEH "Universität Süd" 009250000000000000_0000000000000000000
26 BUH "15" "bus" "SWU_Verkehr" "Willy-Brandt-Platz - Universität Sü
       d" "" "" "" 11111600N ""N  ""
27 LAH000000000000010000000012
28 La000001050000000000
29 La000001058000000000
30 La000001048000000000
31 La000001059000000000
32 La000001060000000000
33 La000001061000000000
34 La000001079000000000
35 La000001078000000000
36 La000001087000000000
37 La000001160000000000
38 La000001241000000000
39 La000001240000000000
```

The author reverse-engineered most of this—also undocumented—format in order to transform it into GTFS, first through intermediary CSV files of SWU's schedule, and then directly out of DIVA, with the complete schedule of the DING transit system. The findings of this process are laid out in Chapter 4 and Chapter 5, with the latter explaining the workings of the DIVA data format.

3.5 ALONG COMES GOOGLE: GTFS

What was to become GTFS started out as a side project of Google employee Chris Harrelson in mid-2005, who "monkeyed around with ways to incorporate transit data into Google Maps [...] when he heard from Tim and Bibiana McHugh, married IT managers at TriMet, the transit agency for Portland, Ore[gon]" [76, P. 3]. McHugh found it "very frustrating to try and find transit directions in [...] unfamiliar cities", it being "much easier at that time to get driving directions from popular online mapping services" [57, P. 126]. Having realized this status quo as being a potentially encouraging fact for choosing car usage over public transit, the McHughs eventually got into contact with Google. TriMet provided Google with CSV files from their existing enterprise database, based on TriMet's database schema, and in December 2005, Portland became the first city to be featured in the first version of Google's "Transit Trip Planner" [38]. This extension of Google Maps allowed querying bus and light rail schedules from

File	Describes	SWU	DING
anschlb	Transfer definition		✓
Aushangbeschreibungen	Public notices		✓
bzw	Operating branches	✓	✓
fahrzeitprofil	Journey time patterns	✓	
ferien	Holidays	✓	✓
fz_gruppe	Journey time pattern group	✓	
haltestellen	Stops	✓	✓
haltezeit	Stopping times	✓	
hinweise	Footnotes, e. g., for transfers	✓	✓
hstattr	Operators responsible for specific stops	(✓)	✓
hst_liste	unknown	✓	✓
hz_gruppe	Stopping time group	✓	
lnrlit	Lines and their operators, per project		✓
mastmat	Unknown	(✓)	(✓)
pkbez	Project definition	✓	✓
streckenzeit	unknown	✓	
tarifz	Fare zones and their adjacency	✓	✓
teilstrecken	Partial sections of journeys	✓	(✓)
tgtyp	Vehicle type	✓	✓
umstmat	Transfer matrix with minimum transfer times	✓	✓
unter	Transit operators		✓
unt_addr	Operators' addresses		✓
uvz_texte	Subdirectory names		✓
vbesch	Service restrictions	✓	✓
vmtext	Means of transport description	(✓)	✓

Table 2: File names encountered, their presence in SWU and/or DING exports, and what is defined within them. In some instances, files were present but appeared to be legacy files only—in these cases, the checkmark is enclosed within brackets.

within the service, akin to the journey planning service for drivers, pedestrians and, since 2010, bicyclists [46]. In September 2006, five more US cities were added to the Google Transit Trip Planner, and the data format released as the *Google Transit Feed Specification* [48].

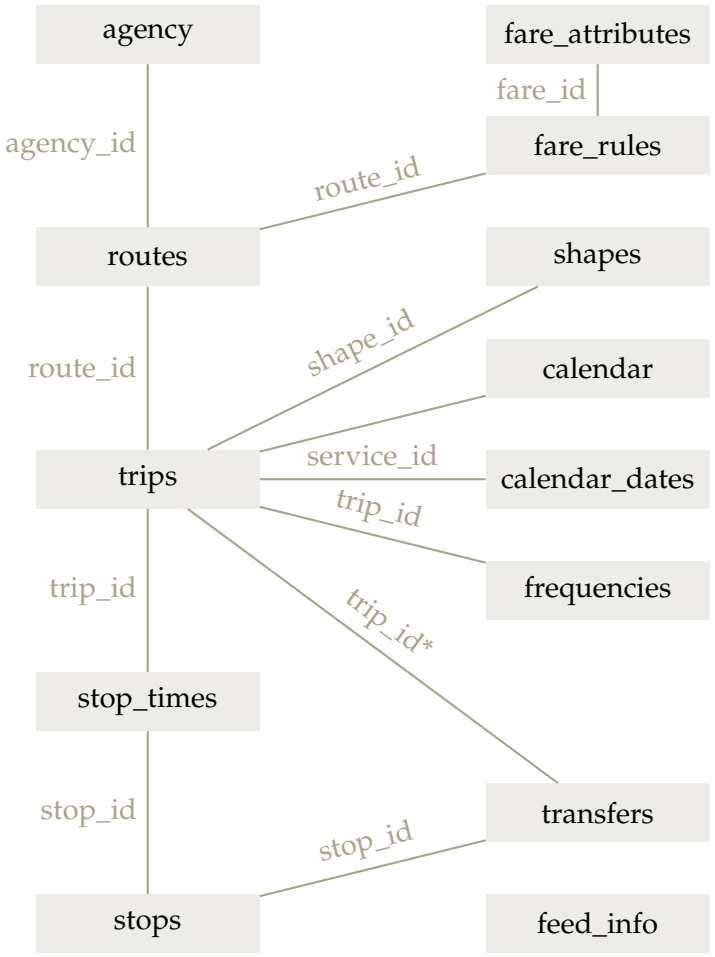


Figure 7: GTFS class diagram. The relationship between the transfers and trips table is a feature of the Google Transit extension to GTFS.

In the United States, unlike Germany, there had not been any standard for public transit timetables prior to the advent of GTFS, not even a de-facto standard. According to long-time Bay Area Rapid Transit (BART) website manager Timothy Moore, before the advent of GTFS, BART had to provide different data consumers with different formats, making a standardized transit format very desirable [76, P. 23]. The publicly and freely available format specification, as well as the availability of GTFS schedules, quickly made developers base their transit-related software on the format. This resulted in "hundreds of useful and popular transit applications" [57, P. 129] as well as catalogues listing available GTFS feeds, such as the GTFS Data Exchange [26]. Due to the common data format those applications adhere to, solutions do not need to be custom-tailored to one transit

FILENAME	REQUIRED	DEFINES
agency.txt	✓	One or more transit agencies that provide the data in this feed.
stops.txt	✓	Individual locations where vehicles pick up or drop off passengers.
routes.txt	✓	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	✓	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	✓	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	✓	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt		Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file can fully replace calendar.txt.
fare_attributes.txt		Fare information for a transit organization's routes.
fare_rules.txt		Rules for applying fare information for a transit organization's routes.
shapes.txt		Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt		Headway (time between trips) for routes with variable frequency of service.
transfers.txt		Rules for making connections at transfer points between routes.
feed_info.txt		Additional information about the feed itself, including publisher, version, and expiration information.

Table 3: Overview of necessary and optional files of a GTFS feed, and their content according to the specification [44]. The GTFS vocabulary is used.

operator, but can easily be extended to any region where a GTFS feed is available.

3.5.1 *File format*

A GTFS feed consists of at least six, and up to 13 plain text files, preferably in UTF-8 text encoding with `CR` or `CRLF` line endings [44]; see Table 3 for a complete listing of all files cited from the GTFS reference.

Each file contains tabular data as CSV, formatted according to RfC 4180 [79]. The first line of each file must specify the column names used within the file, with each file having a set of mandatory and another set of optional columns. Field values are case-sensitive, must not contain Hypertext Markup Language (HTML) tags, comments or escape sequences, and may not contain tabs, carriage returns or newlines.

In its minimal form with only the required files, GTFS can model one or more transit operators' schedule, as well as the geolocation of stops. Making use of the optional files, the feed can be enriched with fare information, exact line shapes, transfer rules in between lines, as well as service exceptions (i. e., journeys which run or don't run on specific dates).

Listing 7: Excerpt of SWU's GTFS stops.txt

```
1 "stop_id","stop_code","stop_name","location_type","parent_station
      ","stop_lon","stop_lat"
2 9001745,"Arena","Arena",1,,10.00564,48.38168
3 900174501,"Arena","Arena",0,9001745,10.00577,48.38237
4 900174502,"Arena","Arena",0,9001745,10.00564,48.38168
```

3.5.2 *Features*

As outlined in this chapter's introduction, GTFS is aimed towards providing bulk schedule data for applications tailored towards the needs of riders for journey planning purposes. Therefore, the format lacks operational information provided by other standards that are used for internal purposes, such as anything relating to personnell, or parts only relevant for a working timetable—e. g., deadheads.

In its basic form, GTFS allows to model a transit schedule, both timetable-based and frequency-based, for light rail[8], subway or metro lines, rail, bus, ferry, cable car, gondola, and funicular service. This does not suffice to model the subtleties of, e. g., rail services, which usually differentiate between a multitude of different train classes, ranging from regional service over inter-regional trains and long distance trains to high speed rail service. Also, the basic GTFS allows only

8 "Light rail", "Tram" and "Streetcar" all fall under the same identifier in GTFS.

for stop-based transfers, i. e., whether transfers are possible from one stop to another, not for line-to-line or journey-to-journey transfers. Therefore, transfers which are only guaranteed during certain times or in between specific lines can not be modeled in the basic version of GTFS.

The Google Transit Extension to GTFS resolves these issues, and allows for a series of other features commonly found in transit modelling: Station entrances can be defined, as well as stop point names (i. e., Track #4), or stop points only served by a particular set of vehicles [70]. This extension already partly implements the Extended Route Types proposal [43], which expands on the possible line types and allows to differentiate between a number of different rail, coach, bus, urban railway, and taxi services—right up to horse-drawn carriages.

3.6 COMPARISON OF MODELS

The feature sets and modelling capabilities of the presented data models vary widely: While some allow for the modelling of many (if not all) aspects listed in Table 8, GTFS is only fit for the data targeted towards the end-users of public transit. As VDV-451/VDV-452 and TRANSMODEL were the results of a long-lasting development in conjunction with European transit agencies, they have over time evolved to incorporate many necessities in modelling and exchanging the complex data encountered in public transit. Or, as Kizoom and Miller put it, TRANSMODEL "has already encountered and adressed many of the additional requirements that GTFS is encountering piecemeal is it follows a path already well trodden in Europe" [53].

While, due to the lack of publicly available documentation, no definitive assessment can be made about the capabilities of the DIVA and HAFAS data formats, it is safe to assume they have been following the development of TRANSMODEL and VDV-452 and offer comparable feature sets—especially so since the VDV-452 specification lists interfaces distributed by each of the two vendors quoted as being inter-operable with the VDV-452 data model.

GTFS, on the other hand, makes it hard to model some of the conditions commonly encountered in public transit in Germany, for instance when multiple transit operators offer transit services within a *Verkehrsverbund*. GTFS offers the choice of either assigning all lines to the transit authority, losing information as to what operator offers which line, or assigning lines to the respective operator, and consequently losing the logical connection in between those operators that is offered by the transit system. Also, some concepts prevalent in European public transit (especially concerning railway systems) prove difficult to implement into GTFS. As an example, the S1 line of the Munich S-Bahn incorporates *portion working*, i. e., upon arriving in Ne-

ITEM	VDV	TRANSM.	HAFAS	DIVA	GTFS
Specification publicly available	✓	✓	(✓)		✓
Freely licensed specification		(✓)			✓
Schedule information	✓	✓	✓	✓	✓
Fare information		✓	?	✓	✓
Operational data (vehicle sizes etc)	✓	✓	?	✓	
Location data (stops etc)	✓	✓	✓	✓	✓
Line- and stop-specific URI fields					✓
Connections and guaranteed transfers	✓	✓	✓	✓	✓
Rosters and duty information		✓	?	✓	

Table 4: Comparison of data models. Note that "VDV", in this context, refers to VDV-452 only. Additional VDV standards are available, e. g., for rostering. Only the publicly available feature information on the HAFAS-Rohdatenformat is used.

ufahrn, the train is divided, with the front part continuing its journey to Freising, while the back part of the train goes to Munich Airport.

Furthermore, in his process description of transforming the schedules by Stadtwerke Münster into GTFS, Müller points out that the source working timetable included operational data not intended for publishing, e. g., information on vehicle blocks and deadheads, or rostering information [63]. While the rostering information cannot even be modelled in GTFS, vehicle blocks including deadheads must be removed from the working timetable in order to arrive at a resulting GTFS schedule fit for public release. Apart from this internal information concerning deadheads, GTFS constitutes a clearly specified subset of transit data that can safely be distributed to the public without accidentally disclosing internal data that might be considered a corporate secret.

In interviews and the evaluation described in Section 6.2, complete vehicle blocks with pull-outs and deadheads were examples of data which must not be made known to potential competitors.

3.7 CHALLENGES OF TRANSFORMING AND MERGING TRANSIT DATASETS

Establishing a process to create open transit data poses a series of challenges. For one, even if transit authorities use a standardized

format to model their schedules, e. g., VDV-452, there are still different approaches on how to actually model the data. Also, the previously addressed issue of creating presentational timetables out of the working timetables, e. g., removing pull-outs, deadheads and pull-ins, must be addressed if one wants to create a schedule that can be publicly distributed.

See the HAFAS excerpt in Listing 3, where SBB models Deutsche Bahn stations with SBB identifiers.

Another challenge, especially when merging datasets from different transit authorities, lies in different identifiers used for the same items throughout different transit operators or authorities. For instance, while Deutsche Bahn uses their own system of identifying train stations [64], railway operators in other countries or regional transit authorities may integrate the same station, albeit in their own identification system. Some data models allow to identify stops by different identifiers—DIVA, as an example, has multiple ways of adressing stop areas, including optional fields for Deutsche Bahn identifiers and IATA codes, where applicable.

To address this issue, OKF's Open Transport working group has proposed building a referential database for stop identifiers, which might serve as a translation aid between different nomenclatures [45].

The same problem arises with journeys that span the area of multiple transit authorities and will subsequently show up in the schedule data of more than one authority. This could be mitigated by either cross-checking the operator identifiers in each schedule and merging overlapping journeys accordingly, or by only using the schedule of the transit *operator* as a reference for these journeys and ignoring them in the transit *autorities'* schedules. Nonetheless, manual inspection and correction of the resulting schedules is most likely necessary.

3.7.1 *Transit Vocabulary*

Another problem is the result of the differing capabilities in between transit data models explained in Section 3.6. While VDV-452 and TRANSMODEL differentiate between *stop points* as locations where passengers can embark and disembark, and *stop areas* which aggregate groups of stop points, this distinction is less pronounced in GTFS, where everything is a "stop"—although logical connections can be assigned to model the stop point/stop area dichotomy.

See the VDV-452 table reference in Table 1

The data models also use different terminology when defining the same principle. For instance, what is known as a "route" in GTFS is called a "line" in VDV-452, and a GTFS "trip" corresponds to a VDV-452 "journey". TRANSMODEL uses an even more refined vocabulary, distinguishing, for example, between a "trip" as a journey made by a passenger, and a "vehicle journey" as the journey undertaken by a vehicle.

These differences between the different data models can be a source of confusion when converting from one format to another. Open

transit data developers have made efforts to build transit vocabularies that allow the mapping of format semantics, i. e., [27] or [36].

For the scope of this document, the author closely follows the vocabulary proposed by the OKF's Open Transit group [36]:

STOP POINT A point where a vehicle stops to let passengers embark and disembark

STOP AREA A collection of *stop points*

JOURNEY PATTERN An ordered list of *stop points*

JOURNEY A single run of a vehicle along a *journey pattern*

ROUTE A collection of *journey patterns* following the same commercial direction

LINE A collection of—usually two—*routes*

STOP TIME The time when a vehicle is scheduled on a *stop point*. Extending on the Open Transit group's vocabulary, a *stop time* for an scheduled arrival is called an *arrival time* within this work, and a *stop time* for a scheduled arrival is called the *departure time*

3.8 EXAMPLE DATA FLOW AT SWU VERKEHR AND DING

SWU Verkehr is the public transit subsidiary of Stadtwerke Ulm, the municipal works owned by the twin cities of Ulm (in the state of Baden-Württemberg) and Neu-Ulm (in the state of Bavaria). It is responsible for most bus and tram public transit within the city boundaries, making it the public transit operator offering the second-most transit services within the DING linked transport system, surpassed only by the bus and rail services offered by Deutsche Bahn and its subsidiaries. In turn, DING is the transit authority responsible for coordinating the 33 transit operators offering services within the city of Ulm and the districts Alb-Donau-Kreis, Biberach and Neu-Ulm.

Through interviews with employees of SWU Verkehr and DING, the following process was identified, from the planning stage to supplying the vehicles with operational data.

3.8.1 *Planning Stage*

In 2001, SWU procured DIVA by Mentz Datenverarbeitung as a planning tool. Pre-existing stop names and identifiers were entered into DIVA and supplemented with additional information, such as ICTS identifiers. Based on the stop locations, route segments were created between stop pairs to create an O-D matrix, and distances acquired

through SWU's test car and on-board IBIS computers. Finally, the pre-existing timetables were entered into DIVA, completing the software's roll-out.

If new stops or new O-D edges are introduced, distances are usually first estimated in a Geographic Information System (GIS) software, and later refined through test car measurements. Note that the spatial shapes of the O-D edges are *not* recorded, and actually never show up in SWU's data store—all of the measurements only concern the accurate distance between stops.

After having entered lines and journey patterns into DIVA, the *Bildfahrplan*[9] is created. In this step, pre-determined guaranteed transfers serve as guidelines as to where vehicles must meet at certain stops. Building up from these anchors, the rest of the working timetable and blocks are determined. After this step, variants are set up for different operating day types.

If the resulting working timetable has passed all plausibility checks, it is exported to the ICTS system, and to the personnell disposition department, where matching service shifts are created based on it. Afterwards, the working timetable is reduced to representational timetables showing only on-duty journeys. This representational timetable is then exported for further use through DING, and serves as the base for typesetting line- and stop-based printed timetables.

3.8.2 *Operational Stage*

After transferring the planning data into an operational exchange format for export onto SWU's vehicles and its ICTS system, the distances between stops are re-measured, if necessary. Afterwards, matching destination texts are added for each journey pattern, both for the vehicles' internal signs, as well as for on-stop information displays, and the appropriate recorded messages for each stop. Finally, operational details are added, such as roadside infrared beacons, activation points for registering with junction processors, and the necessary data for displaying protected transfers between journeys on the driver's Mobile Data Terminal (MDT) display.

After this operational dataset is complete, it is rolled out through a *Depot Data Manager*, which distributes it to all affected vehicles through the Wireless LAN throughout the depot, and it is imported into the ICTS.

3.8.3 *Data Handling by DING*

Within the DING linked transit systems, schedule rollover occurs once a year in December.

Upon every schedule change, DING requests all the transit operators within its area of responsibility to send in their respective changes

9 A graphical working time table in the form of a time-distance diagram.

from last year's schedule. In SWU's case, the data can be directly imported from SWU's DIVA data export; after importing the data through the DIVA2DIVA interface, only minor manual corrections are necessary. Other operators usually submit the changes from last year's schedule as spreadsheet documents or PDF files with relevant schedule changes highlighted within the document. The deviations from last year's schedule are then manually entered into the DIVA installation maintained by DING.

Interestingly, though one other operator within DING's area of responsibility also uses DIVA in their planning stage, their data has proven unfit for directly importing them into DING's DIVA system. This is a result of that operator using a different modelling approach—i. e., though two parties use the same planning software and the same underlying data model, data compatibility is not automatically achieved.

After having finalized the aggregated schedule, timetables in poster and book form are typeset automatically, and PDF files for each route and line are exported for upload on DING's website. The DIVA files are then imported into DING's online journey planner EFA, from where the data is exchanged on a daily basis with the EFA installation maintained by NVBW. Through this data exchange system, NVBW's EFA system is able to plan journeys in and through all the transit systems in Baden-Württemberg contributing to their database, and additional transit systems bordering on the state. In return, the aggregated data is re-distributed to the partnering transit authorities.

4

EXPORTING DIVA DATA: A FIRST APPROACH THROUGH CSV EXPORTS

The author first attempted in cooperation with SWU to make their timetable data available in GTFS format in 2012. Neither SWU nor DING own licenses for bulk schedule export into any standardized exchange format other than the undocumented DIVA exchange format. However, schedule information can be exported through the following interfaces:

- Timetables can be exported from DIVA as CSV tables with one file containing a tabular schedule for one service type (weekdays, saturdays or sundays) per route, each. These tables are meant to be used for typesetting printed timetables in desktop publishing software. An example table is listed in Table 5.
- Stop information and geolocation can be exported from VICOS LIO as a Keyhole Markup Language (KML) file.
- A *Tagesartenkalender* (type of day calendar) is maintained by SWU Verkehr as a Excel spreadsheet, allocating a numeric type of day key to each calendar day.
- Additionally, shapes for each route were hand-drawn by a SWU employee once and provided in KML format.

Since, at this point, the author considered the DIVA exchange format to be too obscure to be used as a base for a GTFS transformation, he chose to analyze whether the data provided by SWU was sufficient in order to create at least a basic GTFS feed.

4.1 EXPORTING DATA FROM CSV TIMETABLES

The data provided by SWU consists of a series of CSV files which are named according to a common pattern `[10,11][0-9]{3}z[H,R][0,2,3]\.xls`. For example, the schedules of line number 3 would be found in the following files:

```
10003zH0.xls
10003zH2.xls
10003zH3.xls
10003zR0.xls
10003zR2.xls
```

0	1	2	3	4	5	6	7	8	9	10	11	12
				Fahrtenschlüssel	05:23	06:23	07:22	07:32	08:23	08:33	09:23	09:33
				Verkehrsbeschränkung								
				Hinweis				Su		Su		Su
1	1050	Willy-Brandt-Platz		1	05.23	06.23	07.22	07.32	08.23	08.33	09.23	09.33
2	1058	Ostplatz		1	05.24	06.24	07.23	07.33	08.24	08.34	09.24	09.34
3	1048	Örlinger Straße		1	05.25	06.25	07.25	07.35	08.25	08.35	09.25	09.35
4	1059	Steinhövelstraße		1	05.26	06.26	07.26	07.36	08.26	08.36	09.26	09.36
5	1060	Safranberg		1	05.27	06.27	07.27	07.37	08.27	08.37	09.27	09.37
6	1061	Albecker Steige		1	05.28	06.28	07.28	07.38	08.28	08.38	09.28	09.38
7	1079	Eichenplatz		1	05.29	06.29	07.29	07.39	08.29	08.39	09.29	09.39
8	1078	Ludwig-Beck-Straße		1	05.30	06.30	07.30	07.40	08.30	08.40	09.30	09.40
9	1087	Egertweg		1	05.32	06.32	07.32	07.42	08.32	08.42	09.32	09.42
10	1160	Hörvelsinger Weg		1	05.37	06.37	07.38	07.48	08.37	08.47	09.37	09.47
11	1244	Staudingerstraße		1	05.41	06.41	07.48	07.58	08.41	08.51	09.41	09.51
12	1245	Kliniken Wissenschaftsstadt		1	05.42	06.42	07.49	07.59	08.42	08.52	09.42	09.52
13	1246	Universität West		1	05.43	06.43	07.50	08.00	08.43	08.53	09.43	09.53
14	1247	Manfred-Börner-Straße		1	05.44	06.44	07.51	08.01	08.44	08.54	09.44	09.54

Table 5: Example CSV timetable for SWU line number 15 from Willy-Brandt-Platz to Science Park on a weekday. The first line with column names is not part of the CSV file but was manually inserted for better legibility.

`10003zR3.xls`

For the files provided by SWU Verkehr, operating branch, line number, direction of travel and weekday validity can be inferred from the file names, according to the following pattern.

- The first two digits of the file name define the operating branch. For SWU Verkehr, this is either `10` for all bus services or `11` for tram service. In other contexts, different branches can be used to differentiate between operators.
- The following three digits describe the line number. This is not necessarily the line number usually used for customers. For instance, SWU night bus service is marketed as lines N1, N2,...—within the aforementioned naming convention, however, the matching timetables can be found under the names `10901zH0.xls`, `10902zH0.xls`, etc.
- After the letter `z`, the direction of travel is specified. This is either `H` (German, *hin*: towards) or `R` (German, *rück*: back).
- Finally, the last digit describes the service period. In the context of SWU Verkehr, all tables ending in `0` describe weekday service from monday through friday, `2` is for saturday service, and `3` marks the sunday schedule.

Note that, though all files end with `.xls`, the data is stored as CSV data with tab stops as delimiters and no text string escaping.

4.1.1 *Data Layout*

In Table 5 and Table 6, two example timetables are shown. The data representation works as follows:

The four-digit identifier of a stop area matches the identifier used by DING's EFA, although the latter prefixes "900" to all identifiers.

A. *Column 0* enumerates the stop arrivals or departures. This is not necessarily the journey pattern followed by *all* the vehicles on that route, as can be seen in Table 6, and can safely be ignored.

B. *Column 1* is the identifier of the stop area. In SWU's case, this is a four-digit numeric value that is also found in the KML file for the stop location (see Section 4.2) and can be used as a unique identifier for the stop area.

C. In *column 2*, the name of the stop area—as it would be displayed on the physical stop itself—is saved.

D. *Column 3* is only used for layovers at stops. See Table 6 for an example at "Kliniken Wissenschaftsstadt": The line at `an (0)` marks the arrival time, while `ab (1)` is the departure time for the same stop. If no such differentiation is present, all times are departure times only.

E. In *columns 4 ... n*, after a series of labels for the first line's remaining columns, SWU Verkehr exports the identifier suffix for the exact stop point the vehicle services within this journey pattern. For instance, in Table 6, the vehicle services stop area number `1010`—*Theater*, a stop area with four different stop points for bus and tram service—at stop point number `3`. Stop area identifier and stop point suffix can be concatenated, i. e., stop point number 3 at stop area number 1010 can be identified through the unique stop point id `101003`.

F. In the following columns, the timing patterns for the individual journeys are defined.

4.1.2 *Deciphering A Journey Timing Pattern Column*

Arrival and departure times greater than 24.00 are being used for departure/arrival times past midnight. A vehicle starting a journey at 11:50 P.M. and arriving at the final destination at what is technically 1:13 A.M. on the following day, would start its journey at 23.50 and arrive at 25.13 of the same schedule day.

Each column represents one journey, with the *row 0* serving as a unique identifier within the scope of the current file, i. e., within the same route and operating day. This identifier is frequently—but not necessarily—the departure time from the first stop. Since this identifier is not unique once the context of this particular route and day is left, it can be made unique within the scope of all journeys by being prefixed with line name, route direction and day. The first journey in Table 6 could consequently be designated `005H0-0435`.

In *row 2*, scheduling exceptions and guaranteed transfers are saved, if applicable. For SWU Verkehr, guaranteed transfers are assigned alphanumeric identifiers: `[A-D]{1}[0-5]{1}`. These are ignored for the approach described in this chapter. Scheduling exceptions are two letter designators, and their meaning has to be delivered along with the CSV files by SWU Verkehr or deducted by comparing the footnotes of the finished print-ready timetable for a given journey. For instance, `Su` describes journeys only occuring on weekdays during lecture periods of Ulm University, `Fb` journeys only take place on sundays before a holiday.

If *row 3* is *not* the row of the first stop within the pattern, optionally, the vehicle size can be specified. In the data provided by SWU, a difference is being made between single buses and articulated buses, as well as between own vehicles and vehicles provided by contractors. For the scope of this transformation process, this information is ignored.

If vehicle size information is not present, *row 3* marks the departure time for the first stop point. If this journey should start at a later stop point, or if it short turns, a single hyphen (`-`) will appear at all stops before the starting point or after the final stop. Otherwise, all departure times follow the pattern `hh.mm`.

If a route branches in a way that, after a series of stops has been serviced, another series of stops is *not* serviced by a particular jour-

0	1	2	3	4	5	6	7	106	107	108	109	110
				Fahrtenschlüssel	04:35	05:20	05:35	23:39	24:05	24:07	24:09	24:20
				Verkehrsbeschränkung								
				Hinweis	A5		A5	A5 Fa		A5	Fa	A5 Fa
1	1765	Hasenweg		1	04.35	-	05.05	23.20	-	23.50	-	24.20
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
12	1741	Fachoberschule		1	04.47	-	05.17	23.32	-	24.02	-	24.32
13	1744	Washingtonallee		1	$	04.53	$	$	23.33	$	24.03	$
14	1743	Neue Hochschule		1	$	04.53	$	$	23.33	$	24.03	$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
21	1719	Waldeck		1	$	05.01	$	$	23.41	$	24.11	$
22	1729	Meininger Allee		1	$	05.02	$	$	23.42	$	24.12	$
23	1700	ZUP		3	04.48	05.03	05.18	23.33	23.43	24.03	24.13	24.33
24	1710	Rathaus Neu-Ulm		1	04.50	05.05	05.20	23.35	23.45	24.05	24.15	24.35
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
29	1008	Hauptbahnhof		1	05.00	05.15	05.30	23.45	23.55	24.15	24.25	24.45
30	1010	Theater		3	05.02	05.17	05.32	23.47	-	24.17	-	24.47
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
46	1245	Kliniken Wissenschaftsstadt	an (1)	2	05.21	05.36	05.51	24.06	-	-	-	-
47	1245	Kliniken Wissenschaftsstadt	ab (0)	2	05.25	05.40	05.55	24.25	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
50	1240	Universität Süd		2	05.29	05.44	05.59	24.29	-	-	-	-

Table 6: Excerpt of a more complex CSV timetable for SWU line number 5 from Neu-Ulm to Science Park, including different journey patterns. The first line with column names is not part of the CSV file but was manually inserted for better legibility.

ney, before yet another series of stops is serviced again, the omitted stops are marked with a dollar ($) symbol instead of a departure time. This kind of branching can be observed in Table 6: The journeys in columns 5, 7, 106, 108 and 110 all start at *Hasenweg* stop, servicing all stops until *Fachoberschule* and then proceeding directly to *ZUP*. In contrast, the journeys in columns 6, 107 and 109 start only at *Washingtonallee*, servicing a different series of stops until following the same stop pattern after also arriving at *ZUP*. Note that the journeys in columns 107, 108, 109 and 110 also short turn.

4.1.3 *Programmatical Transformation*

For the initial transformation from SWU's files to GTFS, the author used a object-oriented approach using Java. In it, a hashmap of lines was populated with journeys, each with the according stop times. Thus, journeys could be checked online against DING's journey planner EFA, also making up for missing headsign information in the CSV files.

The following pseudo-code suffices for transforming DIVA CSV exports into GTFS. The ProcessFile procedure is called for each file, extracts line number, type, direction and service ID—in this case, 0 for Monday through Friday, 2 for saturdays and 3 for sundays—and makes use of a library function ParseCSV in order to parse the file's contents into a two-dimensional array structure.

```
ProcessFile(filename)
 1  if Substring(filename, 0, 2) = 11
 2     then lineType ← 0
 3     else lineType ← 7
 4  lineNr ← Substring(filename, 2, 5)
 5  if Substring(filename, 6, 7) = H
 6     then routeDirection ← 0
 7     else routeDirection ← 1
 8  serviceId ← Substring(filename, 7, 8)
 9
10  table[][] ← ParseCSV(filename)
11
12  PrintRoutes(lineNr, lineNr, lineNr, lineType)
13
14  while startLine = nil
15        do
16           if table[0][i] = 1
17              then startLine ← i
18              else i ← i + 1
19  TripIterator(table, lineNr, routeDirection, serviceId, startLine)
```

The PrintRoutes, PrintTrips and PrintStopTimes routines can be used to append any input passed to them onto a `routes.txt`, `trips.txt` and `stop_times.txt` file respectively. This will result in duplicate lines ending up in `routes.txt`—up to six occurrences per unique line—which can, however, be filtered out by using GNU `sort(1)` or manually editing the file in a spreadsheet program.

Iterating over the first row of the table array until the first stop is found is not an elegant approach, but it suffices in determining the start line in which all journeys of the current table have their first departure entry.

Afterwards, the array and the line information is passed to TripIterator, which will parse all columns starting with index 4. The first cell of each column is used as a journey identifier, which is then being made unique within the whole GTFS dataset's scope by prepending it with line number, direction and service identifier.

```
TripIterator(table,lineNr,routeDirection,serviceId, startLine)
 1  columns ← getColumns(table)
 2  lines ← getLines(table)
 3  for column ← 4 to columns
 4       do
 5          journId ← table[column][0]
 6          uJournId ← Concat(lineNr, routeDirection, serviceId, journId)
 7          if Contains(table[2][column], Fa)
 8            then
 9                 serviceId ← Fa
10          ▷ proceed accordingly for all service exceptions
11          PrintTrips(lineNr, serviceId, uJournId)
12          for line ← startLine to lines
13               do
14                  A ← table[line][column]
15                  if A ≠ $ and A ≠ -
16                    then
17                         Replace(A, "\.", " : ")
18                         Concatenate(A, " : 00")
19                         if table[line][3] = an (1)
20                           then
21                                line ← line +1
22                                D ← table[line][column]
23                                Replace(D, "\.", " : ")
24                                Concatenate(D, " : 00")
25                           else D ← A
26                         stopId ← table[line][1]
27                         sequence ← table[line][0]
28                         PrintST(ujournId, A, D, stopId, sequence)
```

Service exceptions are expected in line 2 and handled by superseding the current service identifier with the service exception's identifier. In the pseudocode example, only the "Fa" exception is handled—in reality, this would be solved by means of an if-ladder. Once this is taken care of, the current journey can be appended to `trips.txt`.

Afterwards, iterating over all lines of the current column, the departure time A is extracted, and—if it is an actually serviced stop,

i. e., neither - nor $—its decimal point replaced with a colon, and a zero-seconds-appendix is attached to the end.

Should column 3 at the current line contain the string `an (1)`, the departure time D is not identical to the arrival time, but will be found in the following line. Thus, the line counter is increased, the departure time D extracted from that next line, and transformed and appended accordingly. In all other cases, D gets the same value as A.

Note that in some instances this will result in gaps in the sequence numbering. This is acceptable in accordance with the GTFS specification.

The current stop identifier is then extracted from column 1 of the current line, and the stop sequence number can be extracted from column 0 of the current line. Using the PrintST procedure, the current unique journey id, arrival and departure times, the identifier of the current stop and the sequence number are then written into `stop_times.txt`.

4.2 CREATING STOPS.TXT FROM KML STOP LOCATIONS

SWU Verkehr was able to export stop geolocation information as a KML file out of VICOS LIO (see Section 3.8 for more information on SWU's data flow). This file is formatted according to the following pattern:

Listing 8: DIVA KML geolocation excerpt

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.
    google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml
    /2.2" xmlns:atom="http://www.w3.org/2005/Atom">
<Document>
  <name>Haltepunkte.kml</name>
    <!-- Omitted: A series of placemark styling definition -->
  <Folder id="layer fmain">
    <name>Haltepunkte</name>
    <!-- Omitted: Styling definition -->
    <Folder id="layer217">
      <name>Ha ß lerstra ß e</name>
      <Placemark id="layerP407">
        <name>HASS - 01</name>
        <Snippet maxLines="0" id="s407"></Snippet>
        <description>OLIFID: 135901</description>
        <LookAt>
          <longitude>9.97761777777778</longitude>
          <latitude>48.3896536111111</latitude>
          <altitude>0</altitude>
          <heading>0</heading>
          <tilt>0</tilt>
          <range>1000</range>
          <altitudeMode>relativeToGround</altitudeMode>
        </LookAt>
        <styleUrl>#msn_placemark_circle30</styleUrl>
        <Point>
```

```
      <coordinates>9.97761777777778,48.38965361111111,0</
          coordinates>
    </Point>
  </Placemark>
  <Placemark id="layerP408">
    <name>HASS - 02</name>
    <Snippet maxLines="0" id="s408"></Snippet>
    <description>OLIFID: 135902</description>
    <LookAt>
      <longitude>9.97882166666667</longitude>
      <latitude>48.3904411111111</latitude>
      <altitude>0</altitude>
      <heading>0</heading>
      <tilt>0</tilt>
      <range>1000</range>
      <altitudeMode>relativeToGround</altitudeMode>
    </LookAt>
    <styleUrl>#msn_placemark_circle30</styleUrl>
    <Point>
      <coordinates>9.97882166666667,48.3904411111111,0</
          coordinates>
    </Point>
  </Placemark>
  <Placemark id="layerP409">
    <name>HASS - 03</name>
    <Snippet maxLines="0" id="s409"></Snippet>
    <description>OLIFID: 135903</description>
    <LookAt>
      <longitude>9.97862138888889</longitude>
      <latitude>48.3893241666667</latitude>
      <altitude>0</altitude>
      <heading>0</heading>
      <tilt>0</tilt>
      <range>1000</range>
      <altitudeMode>relativeToGround</altitudeMode>
    </LookAt>
    <styleUrl>#msn_placemark_circle30</styleUrl>
    <Point>
      <coordinates>9.97862138888889,48.3893241666667,0</
          coordinates>
    </Point>
  </Placemark>
  <Placemark id="layerP410">
    <name>HASS - 04</name>
    <Snippet maxLines="0" id="s410"></Snippet>
    <description>OLIFID: 135904</description>
    <LookAt>
      <longitude>9.97806722222222</longitude>
      <latitude>48.3884594444444</latitude>
      <altitude>0</altitude>
      <heading>0</heading>
      <tilt>0</tilt>
```

```
          <range>1000</range>
          <altitudeMode>relativeToGround</altitudeMode>
        </LookAt>
        <styleUrl>#msn_placemark_circle30</styleUrl>
        <Point>
          <coordinates>9.97806722222222,48.3884594444444,0</
              coordinates>
        </Point>
      </Placemark>
    </Folder>
    <!-- Omitted: More folders, one for each stop -->
  </Folder>
</Document>
</kml>
```

As can be seen, within the KML Document Object Model (DOM), a single `Document` node contains—besides a document name, and a series of styling description used by the individual points—a single `Folder`. Within this folder resides a series of furter `Folders`, each describing a stop area with all its stops. Each stop is assigned a `Placemark` node, with the stop id—i. e., the four-digit identifier for the stop area and the two-digit suffix for the platform—in its `Description` node. Latitude and longitude are doubly encoded, once as `Longitude` and `Latitude` nodes within a `LookAt` object, and once as a comma separated concatenation of longitude, latitude and altitude within a `Point/coordinates` node. Longitude and latitude use the WGS84 coordinate reference system, and are in decimal degree notation.

Note that the stop name can be prepended with a municipality name. For instance, the *Lindenstraße* stop in the Blaustein municipality west of Ulm would show up in the KML file as `Blaustein, Lindenstraße`.

4.2.1 *Programmatical transformation*

Transforming the KML file into a `stops.txt` adhering to the GTFS specification can be achieved by simple XPATH selections and subsequent text replacement. The following pseudocode example will make use of a FIND procedure returning XML nodes matching a XPATH expression, *ignoring namespaces*. Actual implementations might need to prepend a matching `kml:` namespace prefix. Again, the output is abstracted by a PRINTSTOPS procedure that will take care of opening a file, deleting its contents if not empty and redirecting all output into the file after writing the header row according to GTFS specification.

The outer loop iterates over all `Folders` containing stop sets, extracts the name and retains two forms of it: The full GTFS `stop_code`, including the municipality prefix, if present; and, by removal of any such prefix, the `stop_name`.

The inner loop iterates over all the `Placemarks` within the `Folder`, locating and appropriately truncating the stop identifier, latitude, and longitude. Since each `Placemark` defines a stop point, the first four digits are extracted in order to obtain the `parent_station`, i. e., the stop area.

After printing all thusly obtained "stops located within a stop", the stop area is output as a matching parent station.

```
ProcessFile(kml)
 1  for each stopfolder ← Find(kml, "/kml/Document/Folder/Folder")
 2       do
 3          code ← Find(stopfolder, "./name")
 4          name ← code
 5          Replace(name, ".*", "")
 6          for each stop ← Find(stopfolder, "./Placemark")
 7               do
 8                  id ← Find(stop, "./description")
 9                  Replace(id, "OLIFID : ", "")
10                  parent ← Substring(stop_id, 0, 4)
11                  lat ← Find(stop, "./LookAt/longitude")
12                  lon ← Find(stop, "./LookAt/latitude")
13                  PrintStops(id, \"code\", \"name\", 0, parent, lon, lat)
14          PrintStops(parent, \"code\", \"name\", 1, , lon, lat)
```

4.3 TRANSFORMING THE TYPE OF DAY CALENDAR

The type of day calendar was provided by SWU Verkehr as a spreadsheet with several different versions of the calendar. This spreadsheet was apparently not meant for automated processing, but as a (printed) visual reference for employees—for instance, as a wall calendar. In SWU's case, three major distinctions are being made:

- School days ↔ school holidays
- University session ↔ semester break
- Days with night bus operation ↔ no night bus service

Apart from this distinction, service days are categorized into the following groups:

- Monday through thursday
- Friday
- Saturday
- Sunday

Weekday	ID	School Day	University	Night service
Mon—Thu	10	✓	✓	
	11			
	12	✓		
	13		✓	
Friday	50	✓	✓	✓
	51			✓
	52	✓		✓
	53		✓	✓
Saturday	60			✓
Christmas Eve	61			✓*
New Year's Eve	62			✓*
Sunday	70			
	71			✓
Preholiday	80	✓	✓	✓
	81			✓
	82	✓		✓
	83		✓	✓

Table 7: Type of day calendar matrix used by SWU Verkehr. Note that on Christmas Eve and New Year's Eve, night bus service is customized and differs from the usual night service during the year.

- Day before a holiday
- Special holidays[1]

Combining these two categorizations results in the types of day matrix laid out in Table 7.

MANUAL EXCEPTION HANDLING In the author's first approach, a GTFS `calendar.txt` was prepared in which the obtained trips from Section 4.1.3 were automatically in service mondays through fridays (service id `0`), saturdays (service id `2`), or sundays (service id `3`). Night service was set by default to fridays and saturdays. Afterwards, ex-

1 Additional night service is offered on Christmas Eve and New Year's Eve. Also, Ulm has a special "holiday" in July, dating back from Imperial City days. During the week surrounding this holiday, additional service is offered to cope with increased ridership. The same applies to larger cultural and sports events. These additional services are, however, not present in the CSV exports mentioned in this chapter and therefore omitted here.

ceptions were manually looked up in the time of day calendar and inserted into `calendar_dates.txt`.

For instance, a holiday occurring in the middle of a week meant disabling the regular weekday service for that date; enabling sunday/holiday service; optionally enabling night bus service (e. g., if the specified date was a friday, or if the following date is also a holiday); and enabling pre-holiday service for the previous day. This quickly proved cumbersome and error-prone, requiring time and detail to attention, so that all service types were correctly enabled or disabled for each exception from the regular calendar.

See Table 7

PROGRAMMATICAL ALTERNATIVE Alternatively, the GTFS specification allows omitting the `calendar.txt` file altogether, relying only on `calendar_dates.txt` to enable all service types in use at a certain date. Thus, after manually preparing the calendar spreadsheet, it was possible to programmatically transform the type of day calendar into a working `calendar_dates.txt` file. First, using spreadsheet software, the spreadsheet layout of the most compact calendar version was manually adjusted. In the beginning, each month used two columns, the first one being the date, the second the day type. After arranging the data of all calendar months subsequently in the first two columns and changing the date formatting to `YYYYMMDD`, the data was saved as a CSV file. By means of a Perl script, each line was then split and services enabled accordingly for each day. An example for the transformation can be seen in Listing 9.

In order to maintain compatibility with GTFS consuming applications that rely on a `calendar.txt` being present, one might also consider providing one with all services being disabled on each and every day of the week.

Listing 9: Transformed type of day calendar and the resulting calendar_dates.txt entries

```
Source CSV:
20131227,51
20131228,60
20131229,70
20131230,11
20131231,62

Result:
Sf,20131227,1
Na,20131227,1
Yr,20131227,1
Fa,20131227,1
2,20131228,1
3,20131229,1
Nb,20131229,1
0,20131230,1
Sf,20131230,1
Na,20131230,1
2,20131231,1
Yu,20131231,1
```

4.4 OPTIONAL: MATCHING ROUTE SHAPES

Having completed all previous steps results in a fully functional and working GTFS data set with all required data for usage, e. g., by a journey planning software. However, there is no information on the actual ways vehicles take in between stops.

SWU Verkehr provided the author with a KML file containing all the shapes of all SWU lines. This file, however, turned out not to be an export from one of SWU's data sources, but was instead hand-traced by an employee in a GIS software. Thus, this input file can neither be procured by an automated process, nor do the shapes match any of the O-D distances within either DIVA or VICOS LIO. While it would have been possible to match the shape variants to the journey pattern variants exported through the previously described steps, the author chose to disregard this approach and focus on alternative means of assigning shapes to routes.

ONLINE ROUTE PLANNER VARIANT Another option to obtaining shapes was to group all journeys with identical journey patterns, and query DING's EFA with one arbitrary journey of each set. The result XML output of EFA contains WGS84 coordinate pairs that can be parsed through XPATH queries and converted into GTFS notation. The resulting shapes were then assigned to all journeys of each group, accordingly.

Note that, when requesting a shape for a journey from its origin to its destination, journeys other than the intended one might be returned. This occurs mostly when other lines present a more direct alternative of getting from origin to destination; therefore, the intended journey might not even be returned by the online route planner.

A heuristic to mitigate this effect is to request a journey from its origin to its destination via a stop area in the middle of the planned journey pattern. Nonetheless, the resulting shapes have to be checked for their correctness. This is all the more true for journeys occuring only on special dates, e. g., on Christmas eve. These journeys might not be available through the journey planner outside a certain period surrounding this date.

4.5 DRAWBACKS OF THIS APPROACH

The approach described in this chapter has some shortcomings.

TIMETABLE EXTRACTION One problem with extracting GTFS data as described is that guaranteed transfers cannot be extracted without further information that is not present in the CSV timetables. This is all the more unfortunate in cases like the presented SWU bus line 5, which turns into bus line 3 within Ulm's Science Park, allowing riders

to "transfer" from line 5 to line 3—and vice versa—without changing the vehicle.

To make matters worse, the described process uses the *representational timetables*, which make use of a peculiar notation when it comes to describing the change from line 3 to line 5. While a vehicle approaching "Hochschule Eselsberg" from the Eselsberg quarter does so as a No. 5 bus, it departs from there onwards as a No. 3 bus back towards the city center. The printed timetables for the No. 5 bus, however, continues the schedule until "Universität Süd". In the other direction, a No. 3 bus approaches "Universität Süd", and subsequently leaves the stop area as a No. 5 bus; the printed timetable, however, continues the No. 3 bus schedule until "Hochschule Eselsberg". While this overlapping line display may be helpful for riders, it results in a GTFS file in which not only transfers are not displayed correctly, but phantom buses appear to accompany each other alongsides, when they are, in fact, one and the same vehicle.

Also, no predictions can be made on what headsign a certain vehicle uses. While the `headsign` column is optional as per GTFS specification, it can help riders identify whether a certain vehicle short turns, or which of the journey patterns within a route it follows. While journey planners using GTFS data can use the name of the last stop area as destination information, this is not necessarily the headsign actually used on vehicles by the transit operator.

Figure 8: Geodata example of Haßlerstraße, plotted in QGIS onto map material from OpenStreetMap. The green markers are the stop point positions from the KML file, while the red marker is the stop area location as found in DIVA's internal exchange format and lies approximately between all four stop points. Map background © OpenStreetMap contributors [67].

GEOLOCATION REPRESENTATION The stop location conversion algorithm arbitrarily chooses one stop point to represent the whole stop area—in this case, the last one. In many cases, this will not be a good choice. For instance, the four stop points of *Haßlerstraße*, the example used in the KML excerpt and visualized in Figure 8, are spread over three road segments around a junction, and are up to about 180 me-

ters apart. The code example would use stop point number 4—the lowermost green marker in Figure 8—to represent *Haßlerstraße*, which rather mis-represents the actual location of all its other stop points.

A more refined approach would be averaging the locations of all stop points in order to represent the stop area. However, keep in mind that averaging coordinates is *not* identical with calculating the arithmetic mean of, respectively, all decimal latitude and longitude values. While a simple arithmetic mean of coordinates might suffice for small distances between stop points, and all the more in lower latitudes, it produces a noticeable distortion for larger distances between stop points. This distortion becomes more prominent in higher latitudes. A coordinate conversion into cartesian coordinates, subsequent averaging of their (x, y, z) components and re-conversion into longitude and latitude would result in a more representational "average" coordinate.

However, since a representational coordinate for each stop area already exists within DING's DIVA data set—which is, however, not present in VICOS LIO and therefore also not in the KML export—one might as well choose between the approach described within this chapter, or directly exporting DIVA data as described in Section 5.5.

4.6 CONCLUSION

The approach outlined in this chapter suffices for exporting schedule data from DIVA into the GTFS format, and SWU subsequently decided to publish this data under the Open Database License (ODbL) license, allowing interested developers to freely use and share this data. During the schedule change in December 2013, the author was able to use the process described in this chapter to export the new schedule to GTFS within only a few hours and little manual correction.

However, the shortcomings of this process led the author to re-inspect the DIVA exchange data to find out whether the problems could be solved by directly exporting a GTFS feed from this data.

5 EXPORTING DIRECTLY FROM DIVA DATA

Both DING and SWU provided the author with datasets of their schedule in the DIVA exchange format. In SWU's case, the dataset matched the set they had regularly shared with DING; in DING's case, it was directly copied from DIVA's working directory.

GENERAL APPROACH TO EXPORTING DIVA TO GTFS By reverse-engineering the datasets provided by both parties, the author established an export process using a source and target database. The process consists of the following steps:

1. importing tabular DIVA data into a database,
2. extracting and converting stop and service information from the database,
3. parsing and converting the line description files (optionally by means of a DIVA line list table), and,
4. optionally: Extracting and converting transfer information from the DIVA database.

This will result in a generally complete GTFS feed, although without route shapes.

5.1 FILE STRUCTURE AND LAYOUT

The export consists of a series of plain text files which appear to closely follow the plain text files within DIVA's working directory. As mentioned in Section 3.4, the internal data structure of the files closely resembles that of VDV-451, with some notable exceptions. All relationships between tables presented within this work were reverse engineered by the author by comparing the data with printed timetables, or by plotting coordinates onto OpenStreetMap material in GIS software[1].

Also, the character encoding of the files could not be determined with certainty. While several files contain a `chs;` entry akin to VDV-451, which usually claimed the file encoding to be either Windows-1252 or ISO Latin-1, several files contained characters illegal to both character sets, e. g., pairs of NULL bytes. The author suspects that these character sequences *might* be leftovers from earlier versions of the respective files that were originally encoded using Code Page 437.

1 In this case, the free and open source Quantum GIS [82].

Since simply transforming the text files from Code Page 437 to, e. g., UTF-8 through GNU `iconv (1)` would lose special characters such as Umlauts, a character translation was used as a workaround. Using `tr (1)` from the GNU coreutils package, the offending character `\000` was substituted by a blank space.

Both SWU's and DING's exports included a number of files with different content structures than all of the other files. Since all those different files' last modification timestamp dated from up to 21 years back, it can be assumed they are legacy files from either earlier DIVA versions, or from manual edits. Why the export, which was done directly out of DIVA by it's export function, would include those files, is not clear.

5.2 IMPORTING TABLES INTO AN INTERMEDIARY DATABASE

DIVA exchange files containing tabular data can easily be imported into a SQL database for further processing. Since they all follow a pattern akin to that of VDV-451—and assuming they are all well-formed according to this specification—the conversion is merely a matter of text transforming the tabular entries into SQL statements.

DIVA2SQL(*filename*)

```
1  for each line in filename
2       do
3          if line ← s/tbl;//
4            then
5                 tableName ← line
6          elseif line ← s/atr;//
7            then
8                 columns[] ← SPLIT(line, ";")
9          elseif line ← s/frm;//
10           then
11                formats[] ← SPLIT(line, ";")
12         elseif CONTAINS(line, NUM;)
13           then
14                CREATETABLE(tableName, attributes, formats)
15         elseif line ← s/rec;//
16           then
17                INSERTVALUE(tableName, line)
```

Depending on the type of database being used, column data types might need to be adapted to the database's requirements.

As outlined in the DIVA2SQL pseudo-code example, it suffices to parse each tabular data file line by line. In this example, creating the actual SQL statements is abstracted into a CREATETABLE and INSERTVALUES procedure, respectively. The CREATETABLE procedure has little left to do, apart from iterating over the column names and their data types, resulting in a table creation statement like the example laid out in Listing 10.

Each table entry is then transformed into a SQL insert statement by an InsertValue procedure. In this step, each `rec` line from the source file is split along the semicolons *not* enclosed in quotes. Afterwards, `true` and `false` text entries might need to be replaced with `1` and `0`, respectively. Finally, single quotes—which might be used as apostrophes, for instance—need to be properly escaped.

The author used Perl's quotewords function.

Listing 10: Example SQL table creation statement

```
CREATE TABLE IF NOT EXISTS StopAreaKoord (
        _FK__AutoKey_ INTEGER,
        _FK_ARR_IDX INTEGER,
        plan VARCHAR(4),
        status CHAR,
        x INTEGER,
        y INTEGER,
        z INTEGER,
        disp_x INTEGER,
        disp_y INTEGER,
        text_x INTEGER,
        text_y INTEGER
        );
```

Note that this resulting SQL import will not convey any relational information at all—thus, update or delete actions on the database will not propagate correctly. However, since it is only meant to be a reference from which data is extracted for conversion to GTFS, this is of no further relevance.

5.3 SETTING UP A TARGET DATABASE

In order to save the converted GTFS information, a target database proved useful. The author used SQLite [83], both for the intermediary DIVA database, as well as for the target GTFS database. Since SQLite stores data in a single file, this solution allowed the easy querying, insertion and modification of data, while not requiring a full-fledged database installation, which was not available on all the computers used by the author throughout the prototyping process.

The utilized SQL table structure followed the GTFS specification with the Google Transit extension [70] to allow for journey-specific transfers. Furthermore, indices were added at relevant points to speed up database transactions (see Listing 21 for the statements used).

5.4 TRANSFORMING THE LINE INFORMATION FILES

The line information files can be parsed line by line in order to gather all necessary data to transform it into GTFS compliant notation. Each file describes exactly one line, with up to two routes per line. All parameters are identified by a two-letter designator at the beginning

See Listing 6 as a reference for the transformation outlined in this section.

of a line and are assigned to one route of the line by either the letter `H` or `R`.

A Perl script used by the author to transform the DIVA data according to the description in this section is included in Listing 22.

5.4.1 *Choosing Relevant Line Definition Files*

Analyzing the exports provided by SWU and DING, it appeared as if some of the line definition files referred to variants that were no longer active. Furthermore, some lines are described within several files, each specifying certain variants of the line.

Therefore, it is not sufficient to simply parse all and any files delivered within a DIVA export, but to rely on the `TabelleLnrlit` table within the `lnrlit` file. All active lines are referenced here through one or more records, specifying the file name, a (non-unique!) line short identifier, the schedule period they are part of, etc.

Through a SQL SELECT statement, a complete list of relevant files with their path (relative to the export's base directory) can be constructed:

Listing 11: Selecting relevant line definition file information

```
SELECT
        uvz,
        lierg,
        kbez,
        textpfp,
        TextBAlang
        FROM TabelleLnrlit;
```

A file list of the relevant files can be built from this query's result according to the following pattern:

[base directory]/[uvz]/[lierg].[kbez]

Note that some characters need to be replaced. In the DIVA database, the `lierg` field was found to be a six-character string, padded with spaces at the end. The associated file name is identical, albeit with underscores (_) in place of the whitespace(s).

Furthermore, it was found that, in the analyzed exports, two files reference the same line if

1. the first five characters of the `lierg` field are identical, *and*
2. the `textpfp` and `TextBAlang` fields are identical, respectively.

See Section 3.4.2

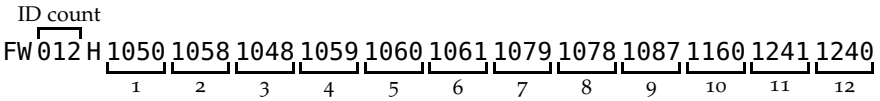
Due to the first two characters of each file name being the operator identifier, the relevant `routes.txt` entries can already be made from the information available at this point. A combination of the operator identifier and the `textpfp` line name proved sufficient to uniquely

identify each line within one dataset, with the operator identifier by itself serving as the agency ID foreign key for the line. The `textpfp` field entry can serve as the line's short name, and the `TextBAlang`—if different from `textpfp`—as the line's long name.

After inserting this information into the target database, each line definition file is parsed line-by-line according to the patterns outlined in the following subsections.

5.4.2 *Journey Patterns*

Each journey pattern description starts with `FW`[2], followed by three digits specifying the number of stop area identifiers that follow, and the route direction it applies to. Taking the example from Listing 6, the journey pattern can be deciphered as follows:

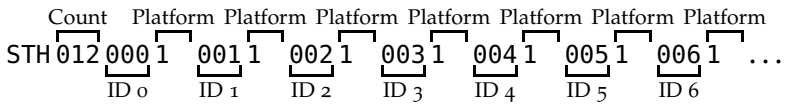


In the exports provided by both DING and SWU, all station area identifiers had four-digit numerical values. Since this might differ for other transit agencies, one can obtain the list of station identifiers by splitting the rear of the `FW` line into n equal parts, with n being the three digit ID count after the initial `FW`.

If the same stop area is referenced twice in a row within a journey pattern, this is in order to model layovers, in conjunction with the timing pattern explained in Section 5.4.4.

5.4.3 *Stop Points*

Starting with *ST*, followed by the direction and three digits specifying the number of stop points. Example: `STH012`, followed—in this case—by twelve three-digit enumerators for the previously defined stop areas, and a platform suffix.



Each of the three digit identifiers references the respective stop area identifier previously defined. The five following characters specify the suffix for identifying the specific stop platform and are padded with spaces, if necessary.

As already mentioned in Section 4.1.1, the stop area and stop platform identifiers can be concatenated in order to uniquely identify

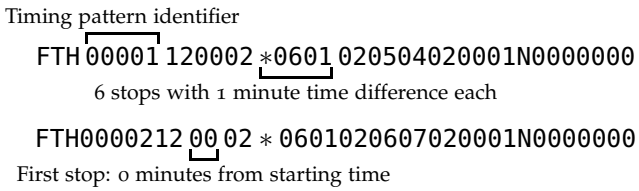
2 Probable meaning: *Fahrweg* (German for "journey pattern")

stop points. Thus, the journey pattern of this example journey would start at stop point `105001`, servicing stop points `105801`, `104801`, etc., until ending at stop point `124002`.

Note that the platform identifiers are not limited to numerical values and might take the form `Zug`[3] or `Gleis`[4].

5.4.4 *Timing Patterns*

Timing patterns, i. e., series of time differences between arrivals and departures at stops, are defined in lines starting with `FT`, again followed by either a `H` or `R` specifying the direction of travel they apply to.



Afterwards, a five-digit value serves as an identifier for this particular timing pattern within this direction of travel.

After two characters without any discernible meaning, the actual timing pattern follows. In its most simple form, it is a succession of two-digit numerical values, with each one defining the time difference in minutes between the last time reference point and the next stop area. Thus, in the example above, timing pattern `00001` would depart from stop point `105001` zero minutes after the journey's start time, departing from the next stop point (`105801`) two minutes later.

In order to condense identical time differences, a simple run-length encoding is used. In the example above, stops three through eight have a departure time difference of one minute each. This is encoded as `*0601`, i. e., six times a one-minute difference.

Note that the timing pattern string is longer than the required twelve values for this string.

By using this timing pattern string in conjunction with the start time of a journey, the absolute arrival/departure times for the journey can be deduced. Using a fictional starting time of 12:00 for a journey following the `00001` pattern, the respective arrival/departure times are 12:00, 12:02 (+2 minutes), 12:03 (+1 minute), 12:04 (+1 minute), 12:05 (+1 minute), 12:06 (+1 minute), 12:07 (+1 minute) 12:08 (+1 minute), 12:10 (+2 minutes), 12:15 (+5 minutes), 12:19 (+4 minutes) and 12:21 (+2 minutes).

The | character was not encountered in the CSV exports used in Chapter 4 as SWU does not use this modelling approach.

Just as in the CSV export timing patterns described Section 4.1.2, apart from numerical values, the special symbols `-`, `$` and `|` can occur.

Again, `-` defines stops at the start or end of an itinerary that are not served by this particular timing pattern (short turns); `$` specifies

3 Train
4 Railway platform

branches within journey patterns not served by particular journeys, and | is used for skipped stops.

Note that, since stop areas can appear as pairs in the journey patterns, layovers can be modeled by this method. If a layover occurs, a non-zero timing value for the second occurrence of the stop describes the time offset in minutes between arrival and departure. A value of zero for the second occurrence marks a pattern where arrival and departure time are identical, and a - can occur when this stop is the final destination of a pattern that short turns.

5.4.5 *Journey Definition*

Each line starting with `FA`[5] defines one specific journey. The `FA` lines were found to follow a pattern that can be matched by means of the regular expression outlined—with named capture groups—in Listing 12. The observed pattern was found to be as follows.

FA: Marks this line as a journey definition

ROUTE DIRECTION IDENTIFIER: Either `H` or `R`.

DAY OF WEEK IDENTIFIER: `0` for Mondays through Fridays, `2` for Saturdays, `3` for Sundays[6]

JOURNEY KEY: Four digits, mostly—but not necessarily—identical to the journey start time[7]. To uniquely identify a journey within a line, the combination of route direction identifier, day of week identifier and journey key is sufficient.

FIVE ZEROES

FOUR FURTHER CHARACTERS (OPTIONAL): In all of the provided files, these were digits. No discernible meaning could be derived as to what they mean.

JOURNEY START TIME: Four digits, specifying the base time reference point for a journey's timing pattern in the local time zone.

SINGLE DIGIT: No discernible meaning could be deduced.

TIMING PATTERN IDENTIFIER: References the timing pattern that applies to this journey.

WHITESPACE (OPTIONAL)

VEHICLE TYPE IDENTIFIER (OPTIONAL): Two characters, alphanumeric, which reference a vehicle type defined in the `Transportgefaesse` table.

5 Probably meaning *Fahrt*, i. e., journey.
6 This is identical to the type-of-day identifiers encountered in Section 4.1.
7 This is identical to the *Fahrtenschlüssel* in Section 4.1.

WHITE SPACE(S) (OPTIONAL): The number of white spaces encountered appears to be arbitrary from `0...n`.

SERVICE RESTRICTION IDENTIFIER (OPTIONAL): References a service type defined in the `ServiceRestriction` table through two characters, alphanumeric.

WHITESPACE(S): At least one

Train types were found to follow the usual names encountered in Germany, e.g., `RE` for "Regionalexpress".

TRAIN CAPTURE GROUP: If the journey is part of a train line, the following string is found: An alphanumeric train number, followed by `0...n` whitespace(s), an optional character with no discernible meaning, and one or more characters referencing the train type.

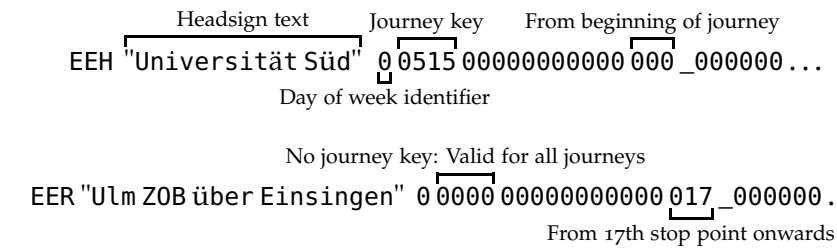
NOTICE IDENTIFIER (OPTIONAL): After an arbitrary number of characters, of which no meaning could be deduced, three optional digits at the end of the line reference a "Notice". This identifier is encountered elsewhere in the DIVA data set; it does not seem relevant for a transformation to GTFS, though.

Listing 12: Regular expression matching the relevant parts of a journey definition line

```
^FA(?<journeyid>(?<direction>[H,R])(?<serviceid>[0,2,3])(?<
    journeykey>[0-9]{4}))0{5}(.{4})?(?<starttime>[0-9]{4}).(?<
    timingpattern>[0-9]{5})\s?(?<vehicletype>[A-Z0-9]{1,2})?\s
    *(?<servicerestriction>[A-Za-z][a-z0-9]{1,2})?\s+((?<trainid
    >[A-Z]?[1-9][0-9]{0,5})\s*[A-Z]?\s*(?<traintype>[A-Z]+))
    ?.*[0-9]{3}(?<notice>\".*\")*
```

5.4.6 *Headsigns*

Information on what headsign is being used during a journey—or a part thereof—is referenced in lines starting with `EE`, followed by the route direction (`R` or `H`) it applies to.



Headsign texts can be defined

A. for an individual journey, referenced by its previously defined direction and day of week identifier, and its journey key.

B. for all journeys of a route, if the journey key is set to `0000`.

Furthermore, headsign texts can be defined

A. for a complete journey, i. e., from its origin to its destination.

B. from the nth stop point onward.

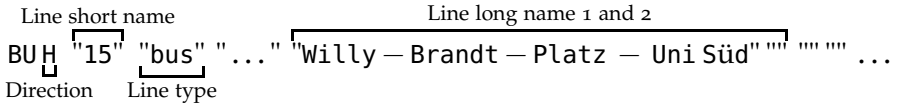
Any combination of these two modes is possible. A regular expression suitable for capturing the relevant information is referenced in Listing 13.

Listing 13: Regular expression matching the relevant parts of a journey definition line

```
^EE(?<direction>[HR])\s\"(?<headsign>.*)\"\s+(?<journey>[0-9]{5})
    .*(?<startingstop>[0-9]{3})_
```

5.4.7 *Line Name and Description*

Information on the line's name and the mode used—i. e., whether it is a bus, train, etc.—can be extracted from lines starting with `BU`.



After the direction identifier (`H` or `R`), a series of strings within double quotes describe the following items

- A short identifier, e. g., the line number.
- The line type. In DING's example, those could be "bus", "bahn"[8], "strab"[9], etc.
- A text string referring to the operator (omitted in the example above).
- One or two strings with the line's long name, i. e., a short description of its itinerary from start to destination.
- A series of further characters whose purpose could not be deduced.

The following regular expression suffices to extract all relevant information:

8 Train
9 Tram

Listing 14: Regular expression for relevant parts of a line name line

```
^BU(?<direction>[HR])\s\"(?<shortid>.*)\"\s\"(?<linetype>.*)\"\s
    (\".*\")\s\"(?<longid1>.*)\"\s\"(?<longid2>.*)\"\s(\".*\")\s
    (\".*\")\s[0-9]*[NY]
```

Since it is unclear whether there exists a standardized nomenclature for the line types, one would propably have to create their own transformation if-ladder to assign a correct line type identifier for the specific DIVA export one would like to transform. Furthermore, DING differentiates between regular buses and, e. g., "Anrufsammeltaxi"[10]. For a precise transformation, one might consider using the extended GTFS route type proposal [43] in order to map such subtleties accordingly.

5.5 TRANSFORMING STOP STRUCTURES AND COORDINATES

Importing the `haltestellen` files into the intermediary database as outlined in Section 5.2 should suffice for exporting all stop-relevant information into GTFS. In the analyzed data sets, the `haltestellen` file contained a total of 32 tables, 11 of which were completely empty. Of the remaining 21 tables, five tables proved sufficient to extract all necessary information regarding stop areas, stop points, their respective relationship and coordinates, as well as the fare zones they are part of.

STOP AREAS The *Stop* table is the main pivot around which all stop information hinges. Apart from its `_AutoKey_`, which is referenced by several other tables, it provides a numerical identifier for each stop area (`hstnr`), validity start and end dates, and the name of the stop prefixed with the name of the municipality it is located in.

FARE ZONES Fare zones are assigned to stop areas in the *Stop_tzonen* table. Stop areas can be part of more than one fare zone, e. g., if they are located on a boundary between fare zones. Since, in GTFS, a stop can be part of only *one* fare zone, the author formed "virtual", new fare zones for these occasions. For instance, if a stop was assigned to both fare zones A and B, the stop was assigned to fare zone AB.

The coordinate model is analyzed in Section A.2.2

STOP AREA COORDINATES The *Stop_hst_koord* table provides coordinate pairs for the stop areas defined in the *Stop* table. Each entry is outfitted with an identifier for the custom CRS being used, as well as a x and y coordinate.

10 Dial-a-ride-transit making use of small vans or a taxi

STOP POINTS A table named *Stop_hst_steig* defines all *Steige*[11] attributed to one stop area. Each stop point references the `_AutoKey_` of one stop area, and is itself identified by both a integer (`nummer`) and a five-character string (`steig`). While both identifiers would suffice to identify individual platforms uniquely (within their stop area), only the string matches the platform identifiers used in Section 5.4.3.

STOP POINT COORDINATES An entry exists within the *StopPlatformKoord* table for each georeferenced stop point from *Stop_hst_steig*. Again, each entry comes with a CRS identifier, and a x and y coordinate.

5.5.1 *Querying Stop Areas and Stop Points*

The relevant queries from the intermediary database are similar, with minor distinctions between the following:

- Stop areas containing two or more platforms are covered by the query in Listing 15
- Stop "platforms" as parts of a stop area (i. e., stop points) can be obtained by means of the query in Listing 16
- Solo stop points, i. e., stop points not affiliated to a stop area, will be the result of Listing 17's query

Through this distinction, the mapping to the three GTFS stop types—parent stops or stations, child stops, and "solo" stations—is already made.

Listing 15: SQL query for DIVA stop areas containing stop points

```
SELECT  S.hstnr AS stop_id,
        S.hstname AS stop_name,
        group_concat(tz.tzonen,"") AS zone_id,
        HK.x AS stop_lat,
        (-1 * (HK.y - 6160000)) AS stop_lon,
        HK.plan AS plan
FROM    Stop AS S
        LEFT OUTER JOIN Stop_hst_koord as HK
                ON S._AutoKey_=HK._FK__AutoKey_
                AND S.input=HK.input
        LEFT OUTER JOIN Stop_tzonen as tz
                ON S._AutoKey_=tz._FK__AutoKey_
WHERE   S._AutoKey_ IN (
        SELECT SHS._FK__AutoKey_
        FROM Stop_hst_steig AS SHS
        WHERE S.input = SHS.input)
GROUP BY stop_id, HK.x
```

11 Platforms

Listing 16: SQL query for DIVA stop points which are part of a stop area

```
SELECT  S.hstnr AS stop_id,
        S.hstname AS stop_name,
        group_concat(tz.tzonen,"") as zone_id,
        SHS.steig AS platform,
        SPK.x AS stop_lat,
        (- 1 * (SPK.y - 6160000)) AS stop_lon,
        SPK.plan AS plan
FROM    Stop AS S
        LEFT OUTER JOIN Stop_tzonen as tz
                ON S._AutoKey_=tz._FK__AutoKey_
        LEFT OUTER JOIN Stop_hst_steig AS SHS
                ON S._AutoKey_ = SHS._FK__AutoKey
        LEFT OUTER JOIN StopPlatformKoord AS SPK
                ON SHS._AutoKey_ = SPK._FK__AutoKey_
WHERE SHS.platform NOT LIKE "Eing%"
GROUP BY stop_id, platform, SPK.x
```

Listing 17: SQL query for DIVA stop points which are not part of a stop area

```
SELECT  S.hstnr AS stop_id,
        S.hstname AS stop_name,
        group_concat(tz.tzonen,"") as zone_id,
        HK.x AS stop_lat,
        (-1 * (HK.y - 6160000)) AS stop_lon,
        HK.plan AS plan
FROM    Stop AS S
        LEFT OUTER JOIN Stop_tzonen as tz
                ON S._AutoKey_=tz._FK__AutoKey_
        LEFT OUTER JOIN Stop_hst_koord as HK
                ON S._AutoKey_=HK._FK__AutoKey
WHERE S._AutoKey_ NOT IN (
        SELECT SHS._FK__AutoKey_
        FROM Stop_hst_steig AS SHS)
GROUP BY stop_id, HK.x
```

5.5.2 *Coordinate Transformation*

See Section A.2.2 for details on the DIVA coordinate model.

Note that the northing offset compensation had already been taken care of in the SQL queries beforehand.

As the stop coordinates are in a DIVA specific format, they need to be transformed to the WGS84 CRS. In the dataset provided by DING, all relevant coordinates were in Gauss-Krüger Zone 3 with a custom offset. Coordinate transformation was simply achieved by passing the x and y coordinates to `cs2cs`[12] with the matching source and target CRS identifiers.

Finally, the thusly obtained stops and their coordinates are written back into the `stops` table within the target GTFS database.

Listing 24 in the Appendix shows the Perl script used to extract and transform the stop data.

12 Available through the `proj(1)` package.

5.6 IMPORTING SERVICE TYPES AND DATES

Service types and their validity dates are defined in the *ServiceRestriction* table—with the exception of the "regular" weekday types `0`, `2` and `3` for Monday through Friday, Saturday, and Sunday, which need to take the current year's holidays into account.

Therefore, the definition of the GTFS `calendar` and `calendar_dates` consists of two steps:

- Determining local holidays and changing those days' service to Sunday service within `calendar_dates`
- Inserting the matching validity dates for all other service types into `calendar_dates` and defining them in `calendar`.

5.6.1 *Determining Local Holidays*

This task can be taken care of by standard calendar libraries, such as Perl's `Date::Holidays::DE` package for German holidays.

First, regular services for Monday through Friday, Saturday and Sunday are inserted into `calendar`, with the corresponding weekdays set to `1`.

Afterwards, exceptions to this rule are determined by iterating over the list of nation-wide and state holidays, obtained by a matching library. If a holiday falls on anything between Monday and Friday, the `0` service is disabled in `calendar_dates`, and a sunday service (`3`) is enabled, instead. In case of a Saturday, the `2` service is disabled and, again, a Sunday service is enabled. If the holiday falls onto a Sunday, no action is necessary.

Date and Time libraries like Perl's `DateTime` are helpful here

5.6.2 *Importing All Other Service Types*

All other services are defined in the *ServiceRestriction* table and can be obtained by the SQL query in Listing 18.

Listing 18: Querying service exceptions from DIVA

```
SELECT  anfjahr,
        code,
        vbt_von,
        vbt_bis,
        vt
FROM    ServiceRestriction
```

`anfjahr` sets a date in YYYY format that sets a reference point against which to calculate the matching service dates—i. e., if `anfjahr` is `2013`, all dates are calculated from `2014-01-01`. `code` is the equivalent to GTFS's `service_id`. `vbt_von` and `vbt_bis` are offsets in months from the `anfjahr` reference date, defining the validity start and end

date. Therefore, if `anfjahr` is `2013`, and `vbt_von` is `23`, the current service validity starts on December 1st, 2014. The same logic applies to `vbt_bis`, as the service validity end date.

The `code` field serves as the identifier for the service in question, and is inserted into the GTFS `calendar` table with all day types from monday through sunday set to zero, and the validity start end end dates from the SQL result. The dates on which the respective service types are valid can be inserted into the GTFS `calendar_dates` table afterwards.

Finally, `vt` defines the individual days on which a given service is valid. To achieve this, it assigns a logical 0 or 1 value to each day of the month—depending on whether the service is valid for that day, or not—, resulting in a 32-bit binary pattern for each month. The least significant bit of each month's binary pattern is the first day of the month, and the leading bits are padded with as many zeroes as necessary to arrive at a 32-bit value. The `vt` column for each service comes with one such 32-bit value, in hexadecimal notation, for each month it is valid, separated by a whitespace.

A month with 31 days will always start with one zero in this notation; a month with 30 days will have two leading zeroes, etc.

Programmatically, this day-service-assignment was solved by iterating over the `vt` column, one month at a time. First, the hexadecimal value was converted into an array with 32 fields of one bit each, and the first and last day of the month in question were calculated through the `DateTime` function. All that was left to do now was to iterate over the array, starting with the last field and the first day of the month, and incrementing the date on each iteration, until the last day of the month was reached. Whenever the array held a `1` value, the current day and the service identifier were inserted into the GTFS `calendar_dates` table with the `exception_type` set to `1`, thus enabling the service for that particular day.

5.7 HANDLING TRANSFERS

Protected transfers are modelled in the *TransferProtection* table, and the relevant information can be extracted through the query in Listing 19.

The individual fields are:

The table offered more fields, which are, however, apparently not being used by the DING and SWU modelling approach.

- `linie_erg_an` and `linie_erg_ab`, the line identifiers for the arriving (`an`) and departing (`ab`) lines between which transfers are guaranteed
- `richt_an` and `richt_ab`, the route direction identifiers for the respective lines
- `wttyp_an` and `wttyp_ab` identify the type of day for which this transfer is valid; the usual DIVA `0`, `2` and `3` distinction applies, as well as an `A` for *all* types of day

- `zeit_von_an`, `zeit_bis_an`, `zeit_von_ab` and `zeit_bis_ab`, the time period for which this transfer is valid, with a start and end time for both the arriving and the departing line. Time points are formatted in minutes since midnight.
- `hst_nr_an` and `hst_nr_ab`, the stop area identifiers to which this protected transfer applies
- `sitz_blb`, which is `Y` if a transfer can be made by staying in the same vehicle

Listing 19: Querying DIVA transfers

```
SELECT  hst_nr_an,
        linie_erg_an,
        richt_an,
        wttyp_an,
        zeit_von_an,
        zeit_bis_an,
        hst_nr_ab,
        linie_erg_ab,
        richt_ab,
        wttyp_ab,
        zeit_von_ab,
        zeit_bis_ab,
        sitz_blb
FROM    TransferProtection
```

The validity timestamps can easily be translated into the GTFS-style timestamps by simple modulo and string manipulation operations; i. e., an input value of `1530` results in an output value of `25:30`.

The author used a handler that distinguished between transfers being made by staying on a vehicle or not, and assigned either the same block identifier in the GTFS `trips` table or modelled the transfer through the `transfers` table, accordingly. A Perl script that takes care of this handling is included in Listing 23.

5.8 EXPORTING THE GTFS FEED FROM THE DATABASE

As the resulting GTFS feed already exists in the required structure, all that is left is to export the individual tables into the matching CSV files. In order to export only stops that are actually serviced by journeys, and only service ids to which journeys are assigned, the series of commands in Listing 20 was used.

Listing 20: Exporting the GTFS database into a text file feed

```
sqlite3 -header -csv diva2gtfs.db "select * from stops AS s where
     s.stop_id in (select distinct parent_station from stops AS
    st where location_type = 0 and st.stop_id in (select distinct
```

```
     stop_id from stop_times)) UNION select * from stops where
    stop_id in (select distinct stop_id from stop_times);" >
    stops.txt
sqlite3 -header -csv gtfs.db "select * from calendar where
    service_id in (select distinct service_id from trips);" >
    calendar.txt
sqlite3 -header -csv gtfs.db "select * from calendar_dates where
    service_id in (select distinct service_id from trips);" >
    calendar_dates.txt
sqlite3 -header -csv gtfs.db "select * from trips;" > trips.txt
sqlite3 -header -csv gtfs.db "select * from routes;" > routes.txt
sqlite3 -header -csv gtfs.db "select * from stop_times;" >
    stop_times.txt
```

5.9 ISSUES

The GTFS feed resulting out of this process solved the drawbacks of the first approach outlined in Chapter 4, namely the better representation of stop area coordinates, journey headsigns, and transfers resulting out of vehicle blocks. It even expanded the feed's capabilities by providing guaranteed transfers and required no manual work on the service exception calendar.

However, the resulting feed still lacks journey shapes, as the required data was also not found within the DIVA data files. Furthermore, the way lines are modelled in DING's data results in a GTFS feed that represents single lines (as they appear in the official schedule) through several distinct lines, albeit with similar names. This is usually the case if variants of a line exist for very specific dates, e. g., on Christmas Eve, or during sports events; or when a line is served by more than one operator.

Another issue arises out of the variety of line types offered in German public transit. Similar to the problem described by Müller in his transformation process description for Münster [63], DING models less frequented suburban lines not with precise stop times: Series of stops are assigned the same departure time, which serve as an *approximation* of actual stop times. While this seems reasonable, given that the lines in question are served by taxi-like minibuses, and passengers have to request a pick-up in advance via telephone, the GTFS validation tools will treat such a schedule as erronous. Furthermore, this pickup-on-request-pattern for specific lines and operating hours has proven difficult to transform automatically.

The journey shapes can still be extracted from the EFA system by the process described in Section 4.4. One way of addressing the remaining issues would be the provision of an GTFS feed editor with an easy-to-use user interface, which would allow the manual editing of the feed without an intimate knowledge of the data format, or having to manually edit the source database or text files.

5.10 CONCLUSION

The author implemented the approach outlined in this chapter in a series of Perl scripts handling the individual parts. At the end of the implementation process, he was able to transform the DIVA data sets supplied by DING into a working GTFS feed, including transfers, service exceptions, and the assignment of stop areas to fare zones, automatically. Only a limited set of issues remained, which could be addressed by further expansion of the process.

While lacking data from other DIVA users leave open the question whether this process could be easily adapted to other DIVA deployments, it should have lowered the bar to transforming the data into open transit datasets.

6

WHAT IS HOLDING BACK OPEN TRANSIT DATA IN GERMANY?

While governments aim to implement open data strategies at national, state and municipal level in Germany, and processes in order to transform schedule data to GTFS exist, the question remains why, as of mid-2014, only two German transit authorities have chosen to release their schedules as open data, with a third one having announced to follow suit. In this chapter, the author analyzes legal obligations, as well as a survey taken with transit authorities and operators throughout Germany.

6.1 LEGAL MATTERS

Public transit and the publishing of schedule data is regulated by a series of legal questions.

One issue that could potentially prohibit the sharing of transit data is contract obligations between transit operators and authorities, and with service contractors. If data is aggregated based on specific restrictions on the usage of data, e. g., if participating operators provide their data only for the integration in existing journey planning systems, such contracts might need to be adjusted accordingly. Such specifics need to be addressed on an individual basis, and no general statement can be made.

6.1.1 *Transit Legislation*

Through the Personenbeförderungsgesetz (PersBefG), regional transit operators are required to create schedules which include the journey patterns, with origin and destination, stops en route and travel times between routes, for each line; to issue these schedules in the customary manner; to post the valid schedules in designated waiting areas; and, upon request, to provide the schedule data to the transit permit authority in a suitable electronic format [5, Sec. 40, Sec. 45].

Railway undertakings—which are not covered by PersBefG but Allgemeines Eisenbahngesetz (AEG)—are required to include information about possible transfers to trains of all other operators in their publicised schedule information [9, Sec. 12]. Gennaro et al. point out that no requirement for the provision of schedules for passengers or competing railway undertakings can be found in AEG [40, P. 52]. Additionally, [25, Annex II] requires railway undertakings to inform their passengers, before a journey, about the time schedules and

conditions for the fastest trip and for the lowest fares as a minimum requirement, therefore also not requiring the publication of complete schedules.

6.1.2 *Intellectual Property Rights*

In the 2010 final report of a research project by transit authorities and transit service providers, Gennaro et al. analyze the legal framework conditions as to whether schedule data can be considered protected through German Intellectual Property (IP) legislation, and what other legislation applies to schedule data. [40, P. 46–66].

Personal, intellectual creations are protected through the German *Gesetz über Urheberrecht und verwandte Schutzrechte* [2, Sec. 2], i. e., a creation has to be the work of a human person, and a certain threshold of originality must be crossed for IP rights to apply. Since the implementation of the EU Database Directive into German copyright law, databases can constitute a protected work, if "by reason of the selection or arrangement of their contents, constitute the author's own intellectual creation" [24, Art. 3][2, Sec. 4]. German IP legislation also grants sui generis database rights [2, Sec. 87 a pp.] in order to protect databases into the creation, verification or presentation of which a qualitatively or quantitatively substantial investment was made by the creators. These sui generis rights also extend to databases which do not meet the originality criteria to be a protected work by themself. Lastly, computer software constitute another class of works to which IP rights can be applied [2, Sec. 69 a pp.].

Gennaro et al. differentiate on this basis between four classes of databases [40, P. 51]:

- if the database is an intellectual creation due to the selection and arrangement of their contents, *and* if a substantial investment was made, the database is a protected creative work, and sui generis database rights apply
- if the database is an intellectual creation due to the selection and arrangement of their contents, but no substantial investment was made, the database is a protected creative work
- if the database cannot be considered an intellectual creation, but a substantial investment was made, only sui generis database rights apply
- if the database can neither be considered an intellectual creation, and no substantial investment was made, the database is not covered by IP protection

Gennaro et al. further elaborate on the different stages of collecting and processing transit data, from the raw data [1], through content-related editing of the data,[2] and technical conversions [3] to the final schedule that can be used to offer schedule information to riders.

They argue that the database does not constitute a protected work or database work from the raw data stage through the technical conversion stage, since the originality criteria are usually not met. Instead, the mere purpose of aggregating and editing the data drastically reduces the leeway in which the data can be ordered in a logical and purposeful manner. Only in exceptional cases could schedules differ from other schedules as a result of personal creative work by a schedule creator. While, in the final stage, the typeset schedule might be considered a result of a personal and creative process, and therefore a protected work, they argue that IP protection is usually not extended to the schedule data itself.

Sui generis database rights were also found unlikely by Gennaro et al. to apply for the databases in the different stages, with the notable exception of the final journey planning system, since a substantial investment is usually made to create and maintain the system.

6.2 EVALUATION: THE STATUS QUO

In order to better understand German transit authorities' position concerning the handling and publishing of schedule data, the author designed a questionnaire in Ulm University's online evaluation system. Subsequently, institutions responsible for regional public transit were invited via E-Mail to participate in this evaluation. The list of 136 E-mail recipients encompassed the list of all *Verkehrsverbünde*, *Tarifgemeinschaften*, *Nahverkehrszweckverbände*, and *Landratsämter*. Recipients were invited to share the link with their respective operators and service contractors, if applicable.

The complete questionnaire is appended in Section A.1

Participants were asked to give basic information to the type of their institution and its area of operation, as well as whether it handled schedule data and, if yes, whether the schedule data could be queried through an electronic journey planner.

If the participants had stated that they handled schedule information, they were subsequently asked with whom they shared the data, and what kind of exchange formats they used. Afterwards, the open data concept and criteria were briefly explained, and participants were asked whether they had heard of the term before taking the survey, and whether their institution's distribution of schedules

1 In this context: The acquisition of arrival and departure times.
2 In this context: Linking to stop identifiers and coordinates, entering foot paths, and linking with data from other transit operators.
3 E. g., format transformations.

matched a series of criteria concerning open data, including license information.

A series of approval questions closed the survey. On a scale from 1 (agree strongly) to 5 (disagree strongly), participants were asked for their personal judgement concerning open transit data: Whether their institution possessed the technical and legal expertise to publish open transit data, and whether free and open conversion tools and step-by-step instructions, including legal advice, would help them in doing so. Furthermore, participants where asked whether they thought new, innovative transit apps could help improve the attractiveness of public transit, and whether transit authorities could save money through third parties developing transit apps. Finally, opinions were asked about whether third-party developers should be obligated to pay money to schedule distributors if they made money off their applications, whether the provision of online journey planners was primarily the responsibility of transit authorities, and whether riders would make transit authorities responsible for third-party apps giving faulty information out of correct schedules.

As a last question, participants were asked for a short statement about what, in their opinion, is currently the greatest obstacle in publishing open transit data.

Participants were able to give feedback and remarks in a final step, and leave their E-Mail address if they were interested in the results of the study or in participating in further studies concerning open transit data.

6.2.1 *Findings of the Evaluation*

Up until the submission date of this work, 47 surveys were completed and evaluated. Institutions from all German states, except for Hesse and Mecklenburg-Western Pomerania participated in the survey, with the majority of them from Bavaria and Baden-Württemberg. This is not surprising, since both states cover a large area, with their transit systems spanning only a few counties, whereas Berlin and Brandenburg are both covered by only one linked transit system.

With the exception of one single institution[4], all participants handled schedule data in one form or another. Only in the case of a single county administration in Bavaria is schedule data not contributed to an electronic journey planning software. All other participants allow riders to query their schedule through electronic journey planners, be they run by themselves (46,8%), by a third party (29.8%), and/or by partner institutions (55.3%).

4 A group of transit operators; the individual transit operators can be assumed to handle (their) schedules, though.

6.2.1.1 *Data Exchange Formats*

When receiving schedule data, spread sheets, PDF and free-form text together form the most popular class of exchange data (28 mentions, 63.3% aggregated). The proprietary DIVA (n=14, 31.82%) and HAFAS (n=17, 38.64%) exchange formats were more frequently mentioned than VDV-452 (n=12, 27.3%). This is especially true for the state of Baden-Württemberg, where practically all of the *Verkehrsverbünde* and their state-level coordination agency NVBW exchange data in DIVA format. Four institutions (9.09%) receive their data in the DIVA Datenpool Nord (DINO) format, two each (4.55%) mentioned the IVU.pool and ISA format, and a series of formats was mentioned only once (2.27% each), including TRANSXCHANGE, RailML and formats encountered in other European countries. One *Verkehrsverbund* also claimed to receive schedule data in GTFS format.

When sharing schedule data with others, free-form text and spread sheets played a much smaller role (16 mentions, 36.36% aggregated), surpassed by the HAFAS (n=18, 40.91%) format, and followed by DIVA (n=14, 31.82%) and VDV-452 (n=12, 27.27%). 5 instutions share schedule data in the GTFS format (11.36%), one uses a TRANSMODEL based format (2.27%), and 8 noted different formats (18.18%), including, again, RailML, DINO, IVU.pool and ISA.

6.2.1.2 *Open Transit Data*

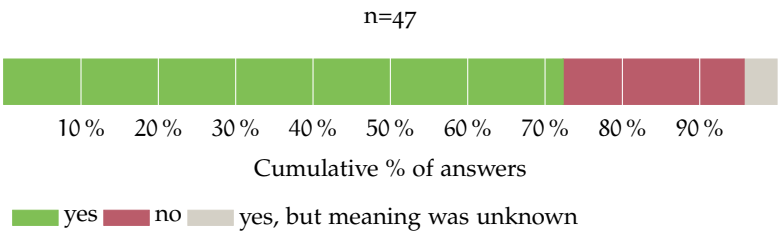


Figure 9: "Had you heard about the term 'open data' before this survey?"

72.3% of the participants had previously known the term "open data", and 4.3% had at least heard of the term. Unfortunately, the questions whether schedules published by the respective institutions meet open data criteria might have been worded ambiguously, as many participants considered the schedules they published as PDF to be a machine-readable exchange format—which is technically true, but not in the sense of open data. Others regarded APIs to their online journey planner as a means of downloading the schedule data. One participant noted that their agency places emphasis on application developers only using their own journey planner API to allow for consistent journey results, regardless of the frontend being used. Another comment stated that both for the raw data, as well for the API, third parties have to sign a user agreement.

Of the 23 participants who had earlier stated that their institution provided schedule data to download for anyone—and be it only in PDF format—, 82.6% stated that there was no explicit license attached to the data, or that they did not know the license terms it was made available under.

Through the free-text comments, three participants noted that they planned to publish open transit data in the near future, or were currently preparing neccessary steps to do so, with one of them pursuing the release for non-commercial purposes only. One institution commented that they considered providing Google with a GTFS schedule feed, but a distribution to others was not their goal.

6.2.1.3 *Personal Opinions: Capabilities*

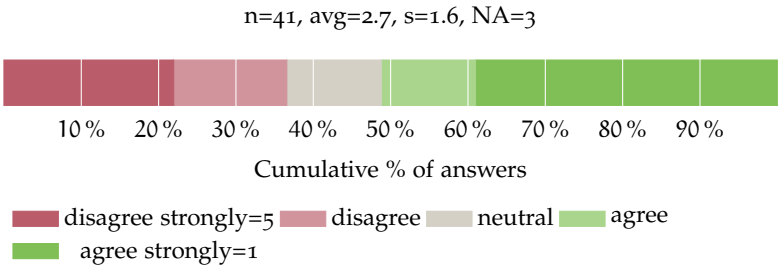


Figure 10: "Are the necessary tools and technical expertise for exporting open transit data available in your institution?"

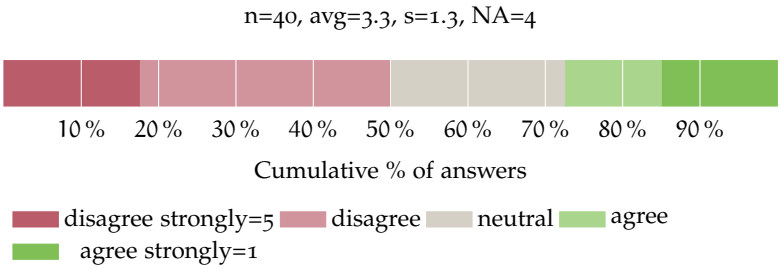


Figure 11: "Is the legal expertise necessary for publishing open transit data available within your institution?"

The self-assessment whether the technical capabilities and the legal expertise for providing open transit data were present in the participants' institutions varied widely. Similarly, no general consensus could be found whether free export tools could facilitate the provision of open transit data.

However, the proposal of a step-by-step instruction "manual" for providing open transit data, including the necessary legal advice, was seen favourably by a majority of the participants (54.54% aggregated agreement and strong agreement), while only an aggregated 25.72% disagreed, or disagreed strongly.

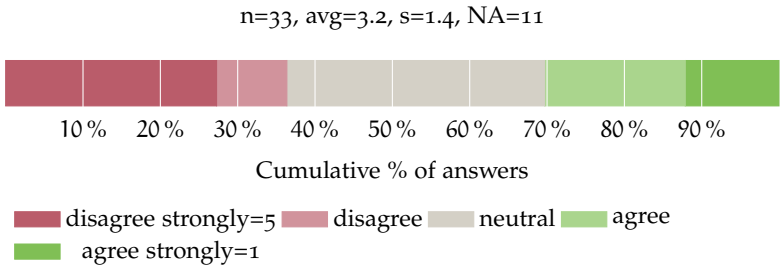


Figure 12: "Free exporting tools for the data formats we use could help us in publishing open transit data"

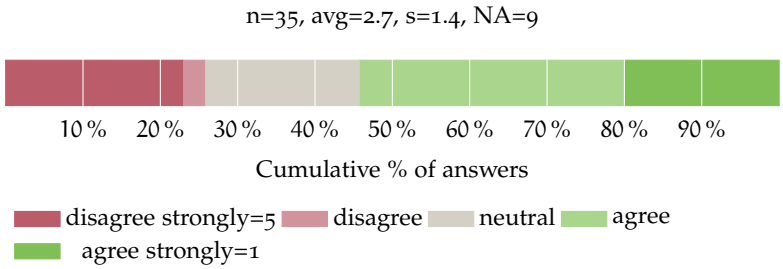


Figure 13: "A step-by-step manual, including an explanation of the legal framework, would help us in publishing open transit data"

6.2.1.4 *Personal Opinions: Opportunities*

A majority of the survey participants (69%) agreed or strongly agreed that innovative third-party applications built on schedule data could help improve the attractiveness of public transit. However, the opinions were split on the question whether transit authorities could save money through not needing to develop own transit applications, if third parties offered such solutions based on open transit data.

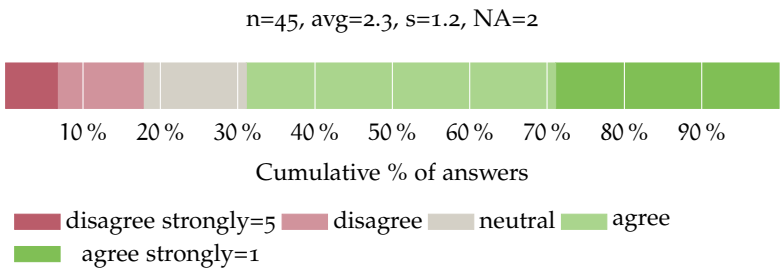


Figure 14: "Third-parties developing new, innovative applications based on schedule data can help improve attractiveness of public transit"

6.2.1.5 *Personal Opinions: Obstacles*

The scepticism about being able to save money through third-party solutions might be linked to a sense that transit authorities themselves are obligated to provide an "official" journey planning service.

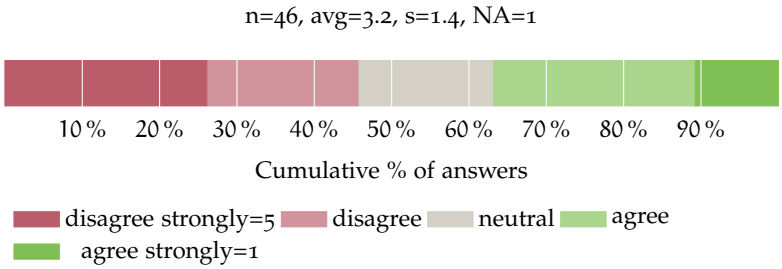


Figure 15: "If third parties develop solutions based on schedule data, transit authorities can save money since they do not have to develop applications themselves"

Asked whether the responsibility for providing online services based on schedule data is primarily the responsibility of transit authorities or their service partners, two thirds of the participants responded positively, with only 11.63% and 2.33% disagreeing or disagreeing strongly, respectively. It can therefore be assumed that even if transit authorities publish their schedule as open data and third parties publish applications based on the data, the transit authorities might still want to supplement their online journey planners with, e. g., "official" smartphone applications.

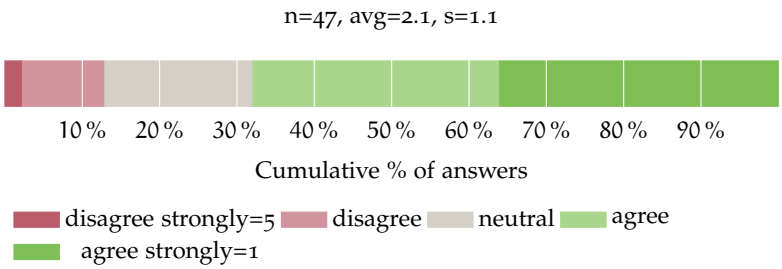


Figure 16: "Providing online services based on schedule data is primarily the responsibility of transit authorities or their service contractors"

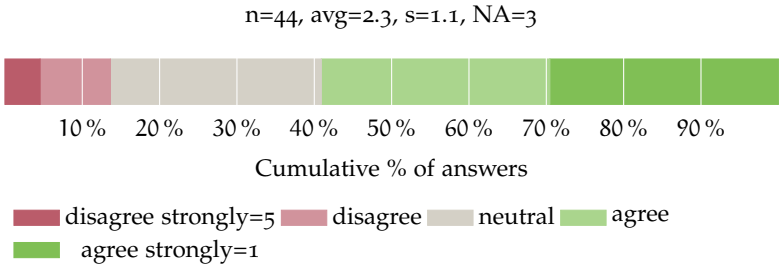


Figure 17: "If third-party developers make profit off applications based on schedule data, they should pay fees to the schedule publishers"

Also, an aggregated 58.54% of the participants agreed or strongly agreed to the question whether third-party application developers

should pay fees to schedule creators if they made money off their product.

One interesting finding concerned the question whether the participants thought users would hold transit authorities responsible if third-party applications gave false advice based on the correct schedules provided by the respective authorities. 48,84% of the participants strongly agreed to this idea, and a further 37.21% agreed, with only 11.63% being neutral and a single strong disagreeing opinion.

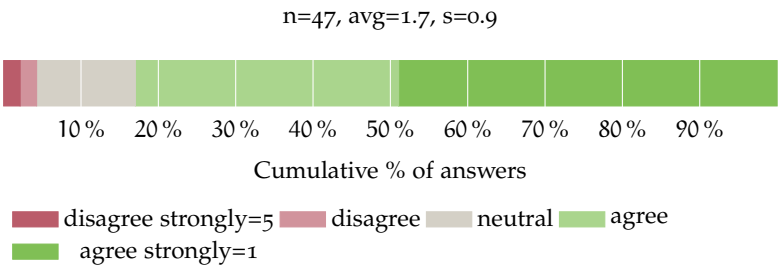


Figure 18: "If third-party applications give false information based on correct schedule data, riders will seek the fault at the transit authority"

The free text answers reinforced this issue by providing examples when, for example, schedule updates did not propagate into partners' journey planning services for weeks, leading to customers complaining to the respective authority instead of to the partner with the outdated schedule information. In total, nine free text answers related to third-party journey planners possibly giving false or not optimal results, often with remarks about metadata concerning footpath routing between stop points, or the up-to-dateness of the schedule data—especially as far as short-notice schedule changes are concerned. Another seven comments pointed out legal concerns, mostly relating to intellectual property rights, but also with regards to possible liability for false information being given. A total of four comments were concerned with possible malicious use of transit data, mostly through competing operators to whom sensitive information must not be spread, but also through third parties compiling delay statistics, or maliciously altering the schedule data. One comment explicitly pointed out a general uneasiness of transit stakeholders' perceived losing control over their data when it is made publicly available, and one comment each mentioned a lack of understanding for the sense and purpose of open transit data, and a perceived lack of willingness to develop an understanding for the topic. Another eight comments mentioned a lack of ressources—both personnell-wise and financially—, lacking know-how, or lacking interfaces or software to pursue opening transit data.

Through the closing comments, the emphasis on correct journey planning results through the transit autorities' own services and APIs

was pointed out five more times, again with emphasis on result quality and once pointing out that VBB's API was in higher demand by developers than the raw open transit schedules. One participant mentioned the susceptibility to being dependent on third parties providing journey planning services, which could "blackmail" transit authorities into paying them money for correctly providing results to riders, or for not disadvantanging them over competing modes of transport, e. g., on-demand car-sharing solutions. Lastly, one participant welcomed open transit data as an opposing pole to not being dependent on, e. g., Google, since everybody was able to get access to the data on the same terms, and no individual contracts needed to be negotiated.

6.3 OUTLOOK

While many transit authorities appear to be sceptical about publishing open transit data, the agencies willing to implement such a process could serve as torch-bearers for a wider adoption. By making open transit data available at least in limited regions in Germany, transit developers and researchers can be incited to redeploy or freshly develop applications that solve specific problems not addressed by the current "official" planning services. These solutions could, in turn, serve as positive examples for the actual usefulness of publishing the data in the first place.

The legal framework concerning IP rights and liability issues also deserves closer analysis, since these were the second-most frequently cited concerns, but literature seems sparse apart from the work by Gennaro et al. [40]. An easy-to-follow explanation of the most important legal concerns and how they might be addressed could remove reservations as to possible liability claims. Furthermore, an analysis on whether open transit data feeds could be published as "unofficial", experimental schedules for developing the proper publishing process without invoking liablity issues might prove interesting. This is all the more true since there is no real way to prevent open transit enthusiasts from accessing the data anyway in one form or another—and be it by scraping the existing PDF schedules, as McHugh pointed out had been done in the US [57].

The problem of transit authorities unjustifiably getting the blame for third-party journey planners giving false information based on correct schedules also deserves further exploration. User studies might provide insights on how well users can differentiate between false advice being given due to the routing software, and false advice based on incomplete or erroneous transit data.

Finally, process definitions for exporting only the schedule data fit for public dissemination might address the reservations concerning

possible competitive disadvantages if trade secrets are accidentally published.

One possible approach to implement an open transit data strategy without having to license new export interfaces is to export the relevant schedule data through the already existing interfaces. Ideally, this would be VDV-452 or a TRANSMODEL-based standard, but the publishing of SBB's HAFAS data and the DIVA transformation process introduced by the author in this work might serve as foundations to also unlocking these proprietary industry standards. This is also the strategy taken by OVapi [69], which takes the official, TRANSMODEL-based open transit data feeds by Dutch transit authorities, and transforms them into GTFS—even supplying them with shape information in the process if the original feed does not contain it [54]. This approach would allow developers to directly work with the data models the transit authorities use themselves, which might lead to the creation of a suite of conversion tools from those standards to GTFS.

7 CONCLUSION

The situation in Germany concerning transit data appears to be profoundly different from the situation that led US transit agencies to open their transit data to interested developers. While GTFS was the first standardized transit data exchange format in the USA and has become the de-facto standard for open transit data throughout the world, transit data model standardization is much further advanced in Europe. German transit agencies rely on a series of established data exchange standards, two dominant ones of which are proprietary with little or no documentation publicly available to open transit developers. Also, in contrast to the United States, where providing developers with transit data was the first step in arriving at online journey planning services for a series of transit agencies, such services already exist literally nationwide in Germany. Transit authorities seem furthermore reluctant to let third parties use their schedules to provide what they see as potentially inferior services. As such, the emphasis for most agencies in Germany appears to be on providing interfaces to their own planning services, which can then, in turn, be used by third party applications—that is, the route computation is done by the "official" route planning service, and third parties can integrate the results in their custom user interface.

The obstacles for open transit data found through the survey evaluation seem to predominantly be less of a technical than of a structural and cultural nature. They closely follow the weaknesses and threats Geiger and von Lucke outlined for open government data [39]. Letting third parties do their own routing on transit data is a *cultural shift* away from the model where transit agencies control the schedules and are the sole providers of definite information; and it is also a *danger to existing business models*, where service contractors are charged with providing and maintaining journey planning systems. This reservation against a *missing interpretive predominance* is also in tune with the perceived fear of *misinterpretation* and a possible *populist mobilisation of mass* through maliciously mis-interpreted data. The *uncertainty of existing copyright laws* appears to be a further deterrent, even for agencies principially willing to open their data. Also, existing *standardization processes* have led to a situation where the data format of choice for developers is different from the data models prevalent in German public transit. This includes the problem of clearly defining the data subset that is fit for publishing without inadvertently disclosing internal data.

However, the survey also found a number of German transit authorities to be actively pursuing open data strategies, and others who were interested in starting to do so. Cooperating with these agencies in order to develop the required processes for releasing *experimental* transit data sets seems to be a promising approach. This could both provide a data basis for transit developers in Germany—potentially leading to showcase projects that could serve to illustrate the meaningfulness of open transit data—and show by example that no negative repercussions need to be feared.

The author has contributed to such a process by dedicating two months of this work's timeframe to analyzing the proprietary and previously undocumented DIVA data format, and subsequently defining transformation processes in order to extract the relevant data for publishing a GTFS data feed from a transit authority's DIVA data set. Further work can build upon these findings by contributing to better GTFS editors, and implementing processes to make the resulting transit data feeds even more complete.

Also, since the author made the decision of focusing on the design and execution of the survey only after this transformation process was developed, survey results continued arriving until the finalizing stage of this work. For this reason, the survey evaluation could not include more sophisticated analysis and deserves closer inspection in later work.

By following up on the findings of this work, researchers and open transit developers can further contribute to implementing transit data publishing processes, together with those agencies identified as willing to also contribute to such processes—thus serving as examples for the spirit of government as a platform.

A

APPENDIX

A.1 EVALUATION QUESTIONNAIRE

A.1.1 *Page 1: Basic Questions*

- Für welche Art von Einrichtung arbeiten Sie?
  - Verkehrsunternehmen
  - Verkehrsverbund
  - Tarifgemeinschaft
  - Servicedienstleister
  - Nahverkehrszweckverband
  - Landratsamt
  - Kommune/Stadt/Gemeinde
  - anderes (siehe Freitext)
- Ergänzung zur Art der Einrichtung
- Freiwillige Angabe: Wie ist der Name Ihrer Einrichtung?
- Freiwillige Angabe: Was ist Ihre Rolle innerhalb Ihrer Einrichtung?
- Meine Einrichtung ist in folgendem Bundesland bzw. folgenden Bundesländern aktiv: (Mehrfachnennung möglich)
  - Baden-Württemberg
  - Bayern
  - Berlin
  - Brandenburg
  - Bremen
  - Hamburg
  - Hessen
  - Mecklenburg-Vorpommern
  - Niedersachsen
  - Nordrhein-Westfalen
  - Rheinland-Pfalz
  - Saarland
  - Sachsen

  - Sachsen-Anhalt
  - Schleswig-Holstein
  - Thüringen
  - weitere (siehe Freitext)
- Ergänzung zur Tätigkeitsregion
- Postleitzahl meiner Einrichtung
- Fließen Ihre Fahrplandaten in eine elektronische Fahrplanauskunft ein?
  - Ja, in eine von uns selbst betriebene Auskunft
  - Ja, in eine von einem Dritten in unserem Auftrag betriebene Auskunft
  - Ja, in die Fahrplanauskunft eines oder mehrerer Partner
  - Nein
  - Weiß ich nicht/keine Angabe
  - Meine Einrichtung führt keine Fahrplandaten
  - Etwas ganz anderes! (siehe Freitext)
- Ergänzung zur Fahrplanauskunft

A.1.2 *Page 2: Schedule exchange*

- Meine Einrichtung nimmt Fahrplandaten von folgenden anderen Einrichtungen entgegen (Mehrfachnennung möglich)
  - Sub- bzw. Schwesterunternehmen
  - Eisenbahnverkehrsunternehmen (z.B. DB)
  - andere Verkehrsunternehmen
  - andere Verkehrsverbünde (oder vergleichbare)
  - sonstige Einrichtungen (bitte im Freitext ausführen)
  - meine Einrichtung nimmt keine Fahrplandaten von anderen Einrichtungen entgegen
- Falls Sie Daten entgegennehmen: Bei der Entgegennahme von Fahrplandaten kommen die folgenden Austauschformate vor (Mehrfachnennung möglich)
  - Freitext („liniertes Papier“)
  - Excel- oder OpenOffice-Tabellen (oder vergleichbares)
  - ÖPNV-Datenmodell (VDV-451/-452)
  - Transmodel
  - Echtzeitfahrpläne (VDV-454, SIRI o.ä.)

  - DIVA-Austauschformat
  - HAFAS-Austauschformat
  - GTFS
  - sonstiges Format (bitte im Freitext angeben)
- Ergänzungen zur Entgegennahme von Fahrplandaten:
- Meine Einrichtung gibt Fahrplandaten an folgende Einrichtungen weiter (Mehrfachnennung möglich)
  - Eltern- oder Schwesterunternehmen
  - Eisenbahnverkehrsunternehmen (z.B. DB)
  - andere Verkehrsunternehmen
  - andere Verkehrsverbünde (oder vergleichbare)
  - Google Transit
  - sonstige Einrichtungen (bitte im Freitext ausführen)
  - meine Einrichtung gibt keine Fahrplandaten an andere Einrichtungen weiter
- Falls Sie Daten weitergeben: Bei der Weitergabe von Fahrplandaten kommen folgende Formate vor (Mehrfachnennung möglich)
  - Freitext („liniertes Papier“)
  - Excel- oder OpenOffice-Tabellen (oder vergleichbares)
  - ÖPNV-Datenmodell (VDV-451/-452)
  - Transmodel
  - Echtzeitfahrpläne (z.B. VDV-454 oder SIRI)
  - DIVA-Austauschformat
  - HAFAS-Austauschformat
  - GTFS
  - sonstiges Format (bitte im Freitext angeben)
- Ergänzungen zur Weitergabe von Fahrplandaten:

A.1.3 *Page 3: Open Transit Data*

- War Ihnen der Begriff „Open Data“ bereits vor dieser Umfrage bekannt?
  - Ja
  - Nein
  - Ich hatte den Begriff gehört, kannte aber die Bedeutung nicht

- Meine Einrichtung stellt den Soll-Fahrplan jedermann (z.B. per Download) zur Verfügung
  - Ja
  - Nein
  - Weiß nicht/keine Angabe
- Der bereitgestellte Soll-Fahrplan steht ohne Anmeldung, Freischaltung oder Vereinbarung zur Verfügung
  - Ja
  - Nein
  - Weiß nicht/keine Angabe
- Der bereitgestellte Soll-Fahrplan liegt in einem maschinenlesbaren Austauschformat vor
  - Ja
  - Nein
  - Weiß nicht/keine Angabe
- Der bereitgestellte Soll-Fahrplan ist folgendermaßen lizenziert:
  - keine explizite Lizenz angegeben
  - Lizenz, welche die Nutzung, Weiterverbreitung und Weiterverwendung nur zu nichtkommerziellen Zwecken erlaubt
  - Lizenz, welche die Nutzung, Weiterverbreitung und Weiterverwendung auch zu kommerziellen Zwecken erlaubt
  - sonstige Lizenz (bitte in den Anmerkungen ergänzen)
  - weiß nicht/keine Angabe
- Anmerkungen und Ergänzungen:

A.1.4 *Page 4: Personal judgements, Part 1*

- In meiner Einrichtung sind das technische Wissen und notwendige Werkzeuge vorhanden, um einen kompletten Soll-Fahrplandatensatz für die Veröffentlichung als Open Data zu exportieren
- Das rechtliche Wissen (zum Beispiel zu Lizenzrechten) zur Veröffentlichung eines Soll-Fahrplandatensatz als Open Data ist in meiner Einrichtung vorhanden
- Kostenlose Exportsoftware für die von uns verwendeten Datenformate würde uns helfen, Fahrplandaten als Open Data bereitzustellen
- Eine Schritt-für-Schritt-Anleitung samt Erklärung der rechtlichen Rahmenbedingungen würde uns helfen, Fahrplandaten als Open Data bereitzustellen

Ihre Einschätzung zur Verwendung offener Fahrplandaten

- Wenn Dritte auf Basis von Fahrplandaten neue, innovative Lösungen entwickeln, kann dies die Attraktivität des ÖPNV steigern.
- Wenn Dritte Onlinedienste, Apps und Ähnliches auf Basis von Fahrplandaten anbieten, können ÖPNV-Anbieter Kosten einsparen, da sie diese nicht mehr aus dem eigenen Budget entwickeln (lassen) müssen.
- Falls Dritte durch Produkte Geld verdienen, die auf Fahrplandaten basieren, sollen sie für diese Daten auch Geld an die Herausgeber der Fahrpläne bezahlen
- Für die Bereitstellung von Onlinediensten, die auf Fahrplandaten basieren, sind in erster Linie die ÖPNV-Anbieter oder von ihnen beauftragte Unternehmen verantwortlich
- Wenn Dritte auf Basis (korrekter) Fahrplandaten falsche Auskünfte geben, werden die Anwender die Schuld beim Herausgeber der Fahrplandaten suchen

Worin sehen Sie die größte(n) Hürde(n) bei der Bereitstellung von Fahrplandaten als Open Data?

A.1.5 *Closing questions*

- Anmerkungen, Ergänzungen und Feedback
- Ich möchte über die Ergebnisse der Studie informiert werden
- Ich bin einverstanden, bei Rückfragen per E-Mail kontaktiert zu werden
- Ich würde gerne weiterführende Studien zu Open Transit Data unterstützen
- E-Mail-Adresse (Angabe freiwillig, wird nicht veröffentlicht. Ihre Antworten können bei der Veröffentlichung der Ergebnisse nicht mit Ihnen in Verbindung gebracht werden.)

A.2 DIVA EXCHANGE FORMAT

A.2.1 *Folder Structure and Naming Conventions*

Figure 19 shows an abbreviated directory tree of DING's DIVA export. Note the distinction between operational files in the root directory, and line definition files ending in "`.j13`" within the `din` folder. A file

		Input element
Timetable	1	Line number
	2	Nodes (stops and stop times on a line)
	3	Pattern (sequence of nodes on a route)
	4	Avg. passenger loads between adjacent
	5	Load factor (desired number of passengers on board the transit vehicle)
	6	Policy headway (the inverse of the minimum frequency standard)
	7	Vehicle type
	8	Vehicle capacity
	9	Avg. running time (travel time between stops/-timepoints)
Vehicles	1	Journey recovery-time tolerances (maximum and minimum time to be prepared for next journey)
	2	Journey departure-time tolerances (maximum departure delay and maximum advance departure)
	3	List of garages (names and locations)
	4	List of start and end locations
	5	Average deadhead times from garage locations to each journey start location (pull-outs)
	6	Average deadhead times from journey end locations to garage locations (pull-ins)
	7	Average deadhead time matrix between all journey end and start locations (by time-of-day)
Personnell	1	Relief-point location (stops, start and end points, garages)
	2	Average travel times between relief points
	3	Journey-layover time (minimum and maximum rest times between two adjacent journeys)
	4	Type of duty (early, late, split, full, tripper, etc.)
	5	Duty length (maximum spread time)
	6	Number of vehicle changes on duty
	7	Meal breaks
	8	Duty composition
	9	Other work rules
	10	List of drivers by name and type
	11	Driver priority and equality rules
	12	One-day-on, one-day-off work pattern

Table 8: Input elements for schedule-planning according to Ceder [21, P. 8]

`uvz_texte`[1] appears to assign a longer description to the three-letter directory name, in this case *Donau-Iller-Nahverkehrsverbund GmbH*.

In comparison, the exports provided by SWU lacked subfolders, placing all files within the root directory. Furthermore, DING's export includes a series of files related to transit operators lacking in SWU's version. In return, the SWU exports include operational tables not present in DING's version (see Table 2 for details).

A.2.2 *DIVA Coordinate Model*

The coordinate model used within DIVA's data model took some comparing and puzzling to make sense of. As a reference, take this abbreviated record:

```
"NAV4";E;4352240;794216;
```

At first glance, the author took this to be a fixed-digit coordinate in WGS84 decimal degree notation, specifying a place at 43.52240° northern latitude, 7.94216° eastern longitude. This idea quickly turned out to be wrong. While the number of decimal places would have matched, this location marks a spot in the Mediterranean, roughly 30 kilometers off the coast of Sanremo, instead of anything even remotely near Ulm—which lies at, approximately, 48.4° N, 9.98° E.

The Gauss-Krüger CRS makes use of zones spanning 3° of longitude each. GK zone 4 uses 12° of longitude as its central meridian.

The "NAV4" identifier and the leading digit 4 of the supposed easting value hinted towards a Gauss-Krüger coordinate located within zone 4 of the GK CRS. However, a valid Gauss-Krüger coordinate would consist of two seven-digit values, while the supposed northing value is specified to only six digits. Plotted directly in the Gauss-Krüger CRS, this coordinate pair resolves to only 7.18423° N latitude and 10.66115° E longitude—while the longitude looks plausible, the latitude places this coordinate in Cameroon, way closer to the Nigerian border than to any means of public transit provided by DING.

Note that DIVA uses `x` to specify the easting and `y` for the northing, which deviates from usual cartographic practise.

Further comparison of coordinates suggested that MentzDV had, for reasons unknown, introduced a custom offset to the northing value of their coordinates, which do, in fact, rely on the Gauss-Krüger CRS. The actual northing is the absolute value of subtracting 6 160 000 from the northing specified in DIVA.

Apart from NAV4 coordinates, DIVA appears to make use of several other custom reference systems listed in Table 9. These reference systems and their parameters were acquired by analyzing the parametrization files of transit applications provided by MentzDV through transit agencies to Android handset users. A number of the identifiers used suggest the agency they might have been developed for: The *TFLV* set makes use of the Ordnance Survey Great

1 Propably an abbreviation for "*Unterverzeichnis_Texte*", which translates to "subdirectory texts"

Figure 19: Example list of DIVA files provided by DING. Some files were omitted.

DIVA	ELLIPSOID	REFERENCE SYSTEM	OFFSET
NAV2	Bessel 1841	Gauss-Krüger Zone 2	6 160 100
NAV3	Bessel 1841	Gauss-Krüger Zone 3	6 160 100
NBWT	Bessel 1841	Gauss-Krüger Zone 3	6 160 100
NAV4	Bessel 1841	Gauss-Krüger Zone 4	6 160 100
MVTT	Bessel 1841	Gauss-Krüger Zone 2	6 160 100
NAV5	Bessel 1841	Gauss-Krüger Zone 5	6 160 100
GIP1	Bessel 1841	ÖBMN M34	6 000 000
VVTT	Bessel 1841	ÖBMN M28	1 000 000
STVH	Bessel 1841	ÖBMN M34	1 000 000
TFLV	Airy 1830	OSGB	1 000 000
ITMR	GRS 80	ITM	1 000 000
MTCV	WGS 1984	UTM Zone 10N	5 000 000
GDAV	WGS 1984	UTM Zone 55S	10 000 000

Table 9: Identifiers of Coordinate Reference Systems used by DIVA and their meaning. The offset is subtracted from the northing and the absolute value of the result is used.

Britain datum, suggesting a relation to Transport for London (TFL), which uses EFA. Also, the identifiers making use of the Austrian Bundesmeldenetz (ÖBMN) datum bear resemblance to the abbreviations of the *Verkehrsverbünde* of Tirol and Styria, which, respectively, also lie within the matching meridian bands.

A.3 SCRIPTS REFERENCE

A.3.1 *GTFS Target Database Creation Statements*

Listing 21: SQL Create statements for setting up a target GTFS database

```
CREATE TABLE IF NOT EXISTS stops (
        stop_id TEXT,
        stop_code TEXT,
        stop_name TEXT,
        stop_lat REAL,
        stop_lon REAL,
        zone_id TEXT,
        location_type INTEGER,
        parent_station INTEGER,
        wheelchair_boarding TEXT
        );

CREATE TABLE IF NOT EXISTS routes (
```

```
        route_id TEXT PRIMARY KEY,
        agency_id TEXT,
        route_short_name TEXT,
        route_long_name TEXT,
        route_type TEXT,
        route_color TEXT,
        route_text_color TEXT
        );

CREATE TABLE IF NOT EXISTS trips (
        route_id TEXT,
        service_id TEXT,
        trip_id TEXT PRIMARY KEY,
        trip_headsign TEXT,
        trip_short_name TEXT,
        direction_id INTEGER,
        block_id INTEGER,
        shape_id TEXT
        );
CREATE INDEX IF NOT EXISTS tr_rid ON trips(route_id);

CREATE TABLE IF NOT EXISTS stop_times (
        trip_id TEXT,
        arrival_time TEXT,
        departure_time TEXT,
        stop_id TEXT,
        stop_sequence INTEGER,
        stop_headsign TEXT,
        pickup_type INTEGER,
        drop_off_type INTEGER,
        shape_dist_traveled REAL
        );
CREATE INDEX IF NOT EXISTS st_trid ON stop_times(trip_id);
CREATE INDEX IF NOT EXISTS st_stid ON stop_times(stop_id);
CREATE INDEX IF NOT EXISTS st_starrtime ON stop_times(
    arrival_time);
CREATE INDEX IF NOT EXISTS st_stdeptime ON stop_times(
    departure_time);

CREATE TABLE IF NOT EXISTS calendar_dates (
        service_id Text,
        date TEXT,
        exception_type INTEGER,
        PRIMARY KEY (service_id, date)
        );
CREATE INDEX IF NOT EXISTS cd_service ON calendar_dates(
    service_id);
CREATE INDEX IF NOT EXISTS cd_date ON calendar_dates(date);

CREATE TABLE IF NOT EXISTS calendar (
        service_id Text PRIMARY KEY,
        monday INTEGER,
```

```
        tuesday INTEGER,
        wednesday INTEGER,
        thursday INTEGER,
        friday INTEGER,
        saturday INTEGER,
        sunday INTEGER,
        start_date TEXT,
        end_date TEXT
        );

CREATE TABLE IF NOT EXISTS fare_attributes (
        fare_id TEXT PRIMARY KEY,
        price REAL,
        currency_type TEXT,
        payment_method INTEGER,
        transfers INTEGER,
        transfer_duration INTEGER
        );

CREATE TABLE IF NOT EXISTS fare_rules (
        fare_id TEXT,
        route_id TEXT,
        origin_id TEXT,
        destination_id TEXT,
        contains_id TEXT
        );

CREATE TABLE IF NOT EXISTS agency (
        agency_id TEXT PRIMARY KEY,
        agency_name TEXT,
        agency_url TEXT,
        agency_timezone TEXT,
        agency_lang TEXT,
        agency_phone TEXT,
        agency_fare_url TEXT
        );

CREATE TABLE IF NOT EXISTS shapes (
        shape_id TEXT,
        shape_pt_lat REAL,
        shape_pt_lon REAL,
        shape_pt_sequence INTEGER,
        shape_dist_traveled REAL
        );

CREATE TABLE IF NOT EXISTS transfers (
        from_stop_id TEXT,
        to_stop_id TEXT,
        transfer_type INTEGER,
        min_transfer_time INTEGER,
        from_route_id TEXT,
        to_route_id TEXT,
```

```
        from_trip_id TEXT,
        to_trip_id TEXT
        );

CREATE TABLE IF NOT EXISTS feed_info (
        feed_publisher_name TEXT,
        feed_publisher_url TEXT,
        feed_lang TEXT,
        feed_start_date INTEGER,
        feed_end_date INTEGER,
        feed_version TEXT
        );
```

A.3.2 *Converting DIVA Journeys To GTFS*

Listing 22: Transforming DIVA journeys into GTFS

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use Switch;
use DBI;
use Getopt::Long;
use open ':encoding(cp850)';

# take care of windows newlines
$/ = "\r\n";

my $divadbh;
my $dbh;
my $basename;
my $tripname;
my $operator;
my $textpfp;
my $textbalang;
my $basepath = '';


dbconnect();

GetOptions ( "path=s"  => \$basepath)
      or die("Error in command line arguments\n");
```

```
my $sth = $divadbh->prepare('SELECT uvz,lierg,kbez,textpfp,
    TextBAlang FROM TabelleLnrlit');
$sth->execute();

while (my $row = $sth->fetchrow_hashref()) {

 # tripname: Everything, e.g. 11310a or 11310_
 # basename: Just the operator and route, e.g. 11310
 $tripname = $row->{lierg};
 $tripname =~ /(?<basename>(?<operator>.{2}).{2}[^_]?).+/;
 $operator = $+{operator};
 $basename = $+{basename};
 # trim trailing spaces
 $tripname =~ s/\s/_/;
 $basename =~ s/\s+$//;

 $textpfp = $row->{textpfp};
 $textpfp =~ s/\s+$//;

 $textbalang = $row->{TextBAlang};
 $textbalang =~ s/\s+$//;


 # build the path to each file. Pattern is uvz/lierg.kbez
    with trimmed spaces
 my $path = $basepath . $row->{uvz} . "/" . $tripname . "."
    . $row->{kbez};
 print "Route: $basename, tripname $tripname, Path: $path\n"
    ;



 if($textpfp eq $textbalang) {undef $textbalang;}

 my $newroute = $dbh->prepare('INSERT OR REPLACE INTO routes
     (route_id, agency_id, route_short_name,
    route_long_name) VALUES (?, ?, ?, ?)');
 $newroute->execute($operator."-".$textpfp,$operator,
    $textpfp,$textbalang);

 $dbh->commit;

 my %job = ('path' => $path, 'tripname' => $tripname, '
    operator' => $operator ,'textpfp' => $textpfp, '
    textbalang' => $textbalang, 'route' => $operator . "-"
    . $textpfp);
```

```
 process(%job);
}


# ---------------------------------------
# SUBROUTINE TO EXPAND TIMING PATTERNS

sub expandtimes {
 my @timearray;
 # push minutes, -, |, $ to array
 foreach (@_) {

  # Expand * sequences. First capture is the amount of
      occurrences, second the content.
  if ($_ =~ /\*([0-9]{2})(\-|\||\$|[0-9]{2})/) {
   for (my $i = 0; $i < $1; $i++) {
    push @timearray, $2;
   }
  }
  # Deal with single occurrences
  else {
   push @timearray, $_;
  }
 }
 # Done pushing the timing pattern to the array
 return @timearray;
}

# ---------------------------------------


# ---------------------------------------
# PROCESS FILE
# ---------------------------------------

sub process {
 my %process = @_;
 my $file = $process{path};
 open (FILE, "<", "$file") or die("Could not open inputfile:
      $!");

 my $line;
 my @stops;
 my %platforms;
 my $route_type;
 my $direction;
```

```
my $route_long_name;
my %FT;

foreach $line (<FILE>) {
 chomp $line;

 #
    ------------------------------------------------------------

 # HEADERS FOR EACH DIRECTION TO BE TAKEN CARE OF
   # These are: Journey Patterns, Stop Points, Timing
       Patterns
 #
    ------------------------------------------------------------



 # Recognize Fahrwege (Journey patterns)
 if ($line =~ s/FW[0-9]*[H,R]//) {
  @stops = ();
  push @stops, substr $line, 0, 4, '' while $line;
   print $log " FW recognized: ";
   print $log "$_ " for @stops;
   print $log "\n";
  }

 # Recognize Stop Platforms
 elsif ($line =~ s/ST[H,R][0-9]{3}//) {

  while ($line =~ /([0-9]{3})(.{5})/g) {

   my $stid = $1;
   my $plat = $2;

   $plat =~ s/\s+$//; # trim trailing spaces

   if ($plat ne '-' and $plat ne '0') {
    $stops[$stid] = $stops[$stid] . $plat;
   }
  }
 }

 # Recognize Timing Patterns
 elsif ($line =~ /FT(?<ftid>[HR][0-9]{5}).{2}(?<pattern>.*)
     [ ,N].* .*/) {
   # create identifier for current pattern
```

```
    my $ftid = $+{ftid};
    my $pattern = $+{pattern};

    # match: *00-, *0000, *00|, *00$ or 00 or - or | or $
    # write everything in temporary tmparray for later
        expansion of * sequences
     my @tmparray = $pattern =~
         /(\*[0-9]{2}\-|\*[0-9]{4}|\*[0-9]{2}\||\*[0-9]{2}\$
         |[0-9]{2}|\-|\||[\$])/g;
    # expand * sequences
    @{ $FT{$ftid} } = expandtimes(@tmparray);
   }
   # Done with timing pattern

  # -------------------------------
  # HEADERS BEEN TAKEN CARE OF HERE
  # -------------------------------


  # ----------------------
  # HERE COME ACTUAL TRIPS
  # ----------------------

  elsif ($line =~ s/^FA//) {

   if ($line =~ /(?<tripid>(?<direction>[H,R])(?<serviceid
      >[0,2,3])(?<tripkey>[0-9]{4}))0{5}(.{4})?(?<starttime
      >[0-9]{4}).(?<timingpattern>[0-9]{5})\s?(?<
      vehicletype>[A-Z0-9]{1,2})?\s*(?<servicerestriction>[
      A-Za-z][a-z0-9]{1,2})?\s+((?<trainid>[A-Z
      ]?[1-9][0-9]{0,5})\s*[A-Z]?\s*(?<traintype>[A-Z]+))
      ?.*[0-9]{3}(?<notice>\".*\")*/) {
    #" regular expressions detailed in the DIVA
       transformation chapter
    my $tripid;
    my $trip_short_name;

    if (defined $+{servicerestriction}) {
     $tripid = $+{direction}.$+{servicerestriction}.$+{
         tripkey};
    }
    else {
     $tripid = $+{tripid};
    }

    my $timingpattern = $+{direction} . $+{timingpattern};
```

```
# Taking care of directions
if ($+{direction} eq "H") {
 $direction = 0;
} else {
 $direction = 1;
}

# If train, use train number as trip id
if (defined $+{trainid}) {
 $trip_short_name = $+{traintype} . $+{trainid};
}

# Take care of service restriction. If a restriction is
    defined, the previous service id is replaced

my $service_id = $+{serviceid};

if (defined $+{servicerestriction}) {
 $service_id = $+{servicerestriction};
}

my $sth = $dbh->prepare('INSERT OR REPLACE INTO trips (
    route_id, service_id, trip_id, trip_short_name,
    direction_id, shape_id) values (?, ?, ?, ?, ?, ?)');
$sth->execute($process{route},$service_id,$process{
    tripname}.$tripid,$trip_short_name,$direction,
    $process{route}.$timingpattern);

# Analyze timing pattern for trip and save stop times

my $hours = substr($+{starttime},0,2);
my $minutes = substr($+{starttime},2,2);
my $arrival_time;
my $departure_time;

for my $i (0 .. $#stops) {
 if ($FT{$timingpattern}[$i] ne '|' and $FT{
     $timingpattern}[$i] ne '$' and $FT{$timingpattern}[
     $i] ne '-') {

  my $sth = $dbh->prepare('INSERT OR REPLACE INTO
      stop_times (trip_id, arrival_time, departure_time,
       stop_id, stop_sequence) values (?, ?, ?, ?, ?)');

  $minutes = $minutes + $FT{$timingpattern}[$i];
```

```
    if ($minutes > 59) {
     $minutes -= 60;
     $hours++;
    }
    $minutes = sprintf("%02d", $minutes);

    $arrival_time = "$hours:$minutes:00";

    # Handle departures/arrivals at same stop: Take time
        of next stop and use it as departure

    if ($i < $#stops and $stops[$i] eq $stops[$i+1] and
        $FT{$timingpattern}[$i+1] ne '-' and $FT{
        $timingpattern}[$i+1] ne '$' and $FT{
        $timingpattern}[$i+1] ne '|') {
     my $dep_hours = $hours;
     my $dep_minutes = $minutes + $FT{$timingpattern}[$i
         +1];
     if ($dep_minutes > 59) {
      $dep_minutes -= 60;
      $dep_hours++;
     }
     $dep_minutes = sprintf("%02d", $dep_minutes);
     $departure_time = "$dep_hours:$dep_minutes:00";

    # If the above procedure has been performed, the next
        iteration is skipped
    } elsif ($i > 1 and $stops[$i] eq $stops[$i-1] and $FT
        {$timingpattern}[$i-1] ne '-' and $FT{
        $timingpattern}[$i-1] ne '$' and $FT{
        $timingpattern}[$i-1] ne '|') {
     next;

    # regular arrival/departure handling
    } else {
    $departure_time = $arrival_time;
    }

    $sth->execute($process{tripname}.$tripid,$arrival_time
        ,$departure_time,$stops[$i],$i);
   }
  }
 }
}

# -------------------------
```

```
# END OF TRIPS
# ------------------------


# ------------------------
# HEADSIGN HANDLING
# ------------------------

elsif ($line =~ s/^EE//) {

 if ($line =~ /(?<direction>[HR])\s\"(?<headsign>.*)\"\s
     +(?<tid>[0-9]{5}).*(?<startingstop>[0-9]{3})_/) {
 #" regular expressions detailed in the DIVA
     transformation chapter
 my $tripid;
  if ($+{tid} == 0) {
   $tripid = "$process{tripname}$+{direction}%";

   # Discriminate: If startingstop is 1 (first stop), Set
       headsign for routeuid within TRIPS table. Otherwise
       , update STOP_TIMES table
   if ($+{startingstop} == 1) {
    my $sth = $dbh->prepare('UPDATE trips set
        trip_headsign = ? where trip_id LIKE ?');
    $sth->execute($+{headsign},$tripid);
   }
   else {
    my $sth = $dbh->prepare('UPDATE stop_times set
        stop_headsign = ? where trip_id LIKE ? and
        stop_sequence >= ?');
    $sth->execute($+{headsign},$tripid, $+{startingstop
        }-1);
   }
  }

  else {
   $tripid = $process{tripname} . $+{direction} . $+{tid};

   if ($+{startingstop} == 0) {
    my $sth = $dbh->prepare('UPDATE trips set
        trip_headsign = ? where trip_id = ?');
    $sth->execute($+{headsign},$tripid);
   }
   else {
```

```
    my $sth = $dbh->prepare('UPDATE stop_times set
        stop_headsign = ? where trip_id = ? and
        stop_sequence >= ?');
    $sth->execute($+{headsign},$tripid, $+{startingstop
        }-1);
   }
  }
 }
}

# -------------------------
# END OF HEADSIGN HANDLING
# -------------------------


# ---------------------------------
# BUS NAME AND DESCRIPTION PARSING
# ---------------------------------

elsif ($line =~ s/^BU//) {

 if ($line =~ /(?<direction>[HR])\s\"(?<shortid>.*)\"\s
    \"(?<routetype>.*)\"\s(\".*\")\s\"(?<longid1>.*)\"\s
    \"(?<longid2>.*)\"\s(\".*\")\s(\".*\")\s[0-9]*[NY]/)
    {
  #" regular expressions detailed in the DIVA
     transformation chapter
  if (not defined $process{textbalang}) {
   $route_long_name = $+{longid1} . $+{longid2};
  }
  else {
   $route_long_name = $process{textbalang};
  }

  # take care of route types

  switch ($+{routetype}) {
   case "bus"  { $route_type = 3 }
   case "bahn"  { $route_type = 2 }
   case "strab" { $route_type = 0 }
   case "SAM"  { $route_type = 99999 } #This needs to be
       corrected manually
   case "AST"  { $route_type = 99998 } #This, too
   case "Fahrradbus"  { $route_type = 99997 } #And this!
   else     { $route_type = 99} # This probably, too.
  }
```

```
   }
  }

  # -------------------------------------
  # END OF BUS NAME/DESCRIPTION PARSING
  # -------------------------------------
 }

 my $sth = $dbh->prepare('UPDATE routes SET route_type = ?,
     route_long_name= ? where route_id IS ?');
 $sth->execute($route_type,$route_long_name,$process{route})
     ;

 $dbh->commit;

 close FILE;
}

# ---------------------------------
# END OF FILE PROCESSING SUBROUTINE
# ---------------------------------


# ---------------------------------
# CLEANING UP!
# ---------------------------------

close $log;
$dbh->disconnect();
$divadbh->disconnect();
print "Database closed. ";
print "Everything done. Bye!\n";

sub dbconnect {
 # --------------------
# CONNECT TO DATABASE
# --------------------

my $driver   = "SQLite";
my $database = "gtfs.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
$dbh = DBI->connect($dsn, $userid, $password, { RaiseError
    => 1 })
                      or die $DBI::errstr;
```

```
my $divadatabase = "divadata.db";
my $divadsn = "DBI:$driver:dbname=$divadatabase";
$divadbh = DBI->connect($divadsn, $userid, $password, {
    RaiseError => 1 })
                    or die $DBI::errstr;

# sacrificing security for speed
$dbh->{AutoCommit} = 0;
$dbh->do( "PRAGMA synchronous=OFF" );

 print "Opened database successfully\n";

# -------------------------
# END OF DATABASE SETUP
# -------------------------
}
```

A.3.3 *Transfer Handling Script*

Listing 23: Excerpt: DIVA transfer information transformation to GTFS

```
# Database functions have been previously established; $dbh
    is the database handler for the target GTFS database,
    and divadbh the database handler for the source DIVA
    database.

sub findtransfers {

 my $sth = $divadbh->prepare('SELECT hst_nr_an, linie_erg_an
     , richt_an, wttyp_an, zeit_von_an, zeit_bis_an,
     hst_nr_ab, linie_erg_ab, richt_ab, wttyp_ab,
     zeit_von_ab, zeit_bis_ab, sitz_blb FROM
     TransferProtection');
 $sth->execute();

 while (my $row = $sth->fetchrow_hashref()) {

  # In DIVA, the time frames are calculated in seconds from
      midnight. The conversion is encapsulated in a
      subroutine that essentially does nothing else than
      return (sprintf ("%02d", int($_/60)) . ":" . sprintf
      ("%02d", $_%60) . ":00")
  my $from_starttime = min2gtfs($row->{zeit_von_an});
  my $from_endtime = min2gtfs($row->{zeit_bis_an});
  my $to_starttime = min2gtfs($row->{zeit_von_ab});
```

```
my $to_endtime = min2gtfs($row->{zeit_bis_ab});

# Startroutes look like "87005" or "87004e", or "219E_e"
    in DIVA. They translate to 87005_ or 87004e or 219E_e
    in the GTFS trips table
my $startroute = $row->{linie_erg_an};
$startroute =~ /(?<basename>.{2}.{2}[^_]?)(?<suffix>.*)/;
if ($+{suffix} eq '') {
 $startroute = $startroute . '\_';
}

my $endroute = $row->{linie_erg_ab};
$endroute =~ /(?<basename>.{2}.{2}[^_]?)(?<suffix>.*)/;
if ($+{suffix} eq '') {
 $endroute = $endroute . '\_';
}
my $from_stop = $row->{hst_nr_an};
my $to_stop = $row->{hst_nr_ab};

# If wttyp_an is A, this transfer is valid for _all_ day
    types starting with this route/stop combination within
     the given time frame.
if ($row->{wttyp_an} eq "A") {
 # ALL day types! First, from day type 0.

 my %job = ('starttrip' => $startroute.$row->{richt_an}."
     0%", 'from_stop' => $from_stop, 'from_starttime' =>
     $from_starttime ,'from_endtime' => $from_endtime, '
     to_stop' => $to_stop, 'to_starttime' => $to_starttime
     , 'to_endtime' => $to_endtime, 'block' => $row->{
     sitz_blb});
 if ($row->{wttyp_ab} eq "A") {
  $job{endtrip} = $endroute.$row->{richt_ab}."0%";
  blockhandler(%job);
  $job{endtrip} = $endroute.$row->{richt_ab}."2%";
  blockhandler(%job);
  $job{endtrip} = $endroute.$row->{richt_ab}."3%";
  blockhandler(%job);
 }
 else {
  $job{endtrip} = $endroute.$row->{richt_ab}.$row->{
      wttyp_ab};
  messyblockhandler(%job);
 }
```

```
   # Now, from day type 2, and then from day type 3 (omitted
       for brevity)

  }

  else {

   # Handle individual days
   my $starttrip = $startroute.$row->{richt_an}.$row->{
       wttyp_an}."%";
   my %job = ('starttrip' => $starttrip, 'from_stop' =>
       $from_stop, 'from_starttime' => $from_starttime ,'
       from_endtime' => $from_endtime, 'to_stop' => $to_stop
       , 'to_starttime' => $to_starttime, 'to_endtime' =>
       $to_endtime, 'block' => $row->{sitz_blb});

   # Again, handling of day type A for the departing trips
   if ($row->{wttyp_ab} eq "A") {
    $job{endtrip} = $endroute.$row->{richt_ab}."0%";
    blockhandler(%job);
    $job{endtrip} = $endroute.$row->{richt_ab}."2%";
    blockhandler(%job);
    $job{endtrip} = $endroute.$row->{richt_ab}."3%";
    blockhandler(%job);
   }
   # Otherwise, the transfer applies just to the individual
       day type
   else {
    $job{endtrip} = $endroute.$row->{richt_ab}.$row->{
        wttyp_ab}."%";
    blockhandler(%job);
   }
  }
 }
}


sub blockhandler {
 my %params = @_;

 # This subroutine is called to transform transferring
     information from $params{starttrip} to $params{endtrip}
      at stop area $params{from_stop} to $params{to_stop}.
     The transfer protection is valid from $params{
     from_starttime} to $params{from_endtime} for the origin
      trip, and from $params{to_starttime} to $params{
```

```
    to_endtime} for the destination trip. If $params{block}
     is "Y", the transfer is achieved by staying on the
    vehicle.


my $sth = $dbh->prepare('SELECT trips.trip_id AS trip_id,
    arrival_time, stop_id, block_id from trips join
    stop_times on trips.trip_id = stop_times.trip_id where
    trips.trip_id like ? ESCAPE "\" and arrival_time >= ?
    and arrival_time <= ? and stop_id LIKE ?');
$sth->execute($params{starttrip}, $params{from_starttime},
    $params{from_endtime}, $params{from_stop}."%");

my %block_identifier;
my %triptransfer;

while (my $arrival_triprow = $sth->fetchrow_hashref()) {

 my $current_arrival_time = $arrival_triprow->{arrival_time
     };
 my $current_arrival_trip = $arrival_triprow->{trip_id};
 my $current_arrival_stop = $arrival_triprow->{stop_id};

 # Handling transfers by staying on the vehicle
 if ($params{block} eq "Y") {
  # Does the inbound trip already have a block ID? If yes,
      we'll use that later on!
  if (defined $arrival_triprow->{block_id}) {
   $block_identifier{$current_arrival_trip} =
       $arrival_triprow->{block_id};
  }
  # If not, we will just use the current trip as a block
      identifier
  else {
   $block_identifier{$current_arrival_trip} =
       $current_arrival_trip;
  }
 }

 # Let's find matching departure trips for this arrival
     trip! Look at all departures between the inbound trip'
     s arrival time and the end of the transfer time frame.
 my $sth = $dbh->prepare('select trip_id, arrival_time,
     stop_id from stop_times
 where trip_id like ? ESCAPE "\" and arrival_time >= ? and
     arrival_time <= ? and stop_id LIKE ?
```

```
  order by arrival_time asc limit 1;
  ');
  $sth->execute($params{endtrip}, $current_arrival_time,
      $params{to_endtime}, $params{from_stop}."%");
  while (my $departure_triprow = $sth->fetchrow_hashref()) {

   my $current_departure_trip = $departure_triprow->{trip_id
       };
   my $current_departure_stop = $departure_triprow->{stop_id
       };

   # Transfer by staying on the vehicle
   if ($messyparams{block} eq "Y") {
    $block_identifier{$current_departure_trip} =
        $block_identifier{$current_arrival_trip};
   }
   # Else: Write a transfer
   elsif ($params{block} eq "N") {
    my $transfersth = $dbh->prepare('INSERT INTO TRANSFERS (
        from_stop_id, to_stop_id, transfer_type,
        from_trip_id, to_trip_id) VALUES (?, ?, ?, ?, ?)');
    $transfersth->execute($current_arrival_stop,
        $current_departure_stop, 1, $current_departure_trip,
         $current_arrival_trip);
   }
  }

 }

 # Finally, if the current request was for block transfers,
     use the temporary hash to write everything to the GTFS
     database!
 if ($params{block} eq "Y") {
  for (keys %block_identifier) {
  my $updatesth = $dbh->prepare('UPDATE trips SET block_id =
       ? WHERE trip_id = ?');
  $updatesth->execute($block_identifier{$_}, $_);
  }
 }

 $dbh->commit();
}
```

A.3.4 *Transforming Stop information from DIVA to GTFS*

Listing 24: Transforming DIVA stops into GTFS

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use DBI;


my $line;

# Database connection subroutine omitted here, see previous
    examples.


my %CS2CS_params = (
  NBWT => '+init=epsg:31467 +to +init=epsg:4326'
  # further CRS could be taken care of here
  );

my $stop_id ="";
my $stop_name = "";
my $stop_lat = "";
my $stop_lon = "";
my $zone_id = "";

# Handling stop areas

my $sth = $divadbh->prepare('SELECT S.hstnr AS stop_id, S.
    hstname AS stop_name, group_concat(tz.tzonen,"") AS
    zone_id, HK.x AS stop_lat, (-1 * (HK.y - 6160000)) AS
    stop_lon, HK.plan AS plan
FROM Stop AS S LEFT OUTER JOIN Stop_hst_koord as HK ON S.
    _AutoKey_=HK._FK__AutoKey_ AND S.input=HK.input LEFT
    OUTER JOIN Stop_tzonen as tz ON S._AutoKey_=tz.
    _FK__AutoKey_ AND S.input=tz.input
WHERE S._AutoKey_ IN (SELECT SHS._FK__AutoKey_ FROM
    Stop_hst_steig AS SHS WHERE S.input = SHS.input)
GROUP BY stop_id, HK.x');
$sth->execute();

while (my $row = $sth->fetchrow_hashref()) {

  $stop_id = $row->{stop_id};
  if (defined $row->{stop_name}) { $stop_name = $row->{
      stop_name}; }
```

```
  if (defined $row->{zone_id}) { $zone_id = $row->{zone_id};
      }

  if (defined $row->{stop_lat} and defined $row->{stop_lon})
      {
   $stop_lat = $row->{stop_lat};
   $stop_lon = $row->{stop_lon};

   my @coords1=split(/\s+/, `echo $stop_lat $stop_lon |
      cs2cs -f "%.8f" $CS2CS_params{$row->{plan}}`);

   $stop_lon = $coords1[0];
   $stop_lat = $coords1[1];

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
      stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($stop_id,undef,$stop_name,$stop_lat,
      $stop_lon,$zone_id,"1",undef,undef);

  } else {
   # In some instances, stops did not have coordinates
      attached for them; this calls for manual re-
      inspection
   $stop_name = $stop_name . "FIXME!";

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
      stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($stop_id,undef,$stop_name,undef,undef
      ,$zone_id,"1",undef,undef);
  }
}


# Handling stop places.

$sth = $divadbh->prepare('SELECT S.hstnr AS stop_id, S.
    hstname AS stop_name, group_concat(tz.tzonen,"") as
    zone_id, SHS.steig AS steig, SPK.x AS stop_lat, (- 1 * (
    SPK.y - 6160000)) AS stop_lon, SPK.plan AS plan
FROM Stop AS S LEFT OUTER JOIN Stop_tzonen as tz ON S.
    _AutoKey_=tz._FK__AutoKey_ AND S.input=tz.input LEFT
    OUTER JOIN Stop_hst_steig AS SHS on S._AutoKey_ = SHS.
    _FK__AutoKey_ AND S.input=SHS.input LEFT OUTER JOIN
    StopPlatformKoord AS SPK ON SHS._AutoKey_ = SPK.
    _FK__AutoKey_ AND SHS.input=SPK.input
WHERE SHS.steig NOT LIKE "Eing%"
```

```
GROUP BY stop_id, steig, SPK.x');
$sth->execute();

while (my $row = $sth->fetchrow_hashref()) {

  $stop_id = $row->{stop_id} . $row->{steig};
  if (defined $row->{stop_name}) { $stop_name = $row->{
      stop_name}; }
  if (defined $row->{zone_id}) { $zone_id = $row->{zone_id};
       }

  if (defined $row->{stop_lat} and defined $row->{stop_lon})
       {
   $stop_lat = $row->{stop_lat};
   $stop_lon = $row->{stop_lon};

   my @coords2=split(/\s+/, `echo $stop_lat $stop_lon |
       cs2cs -f "%.8f" $CS2CS_params{$row->{plan}}`);

   $stop_lon = $coords2[0];
   $stop_lat = $coords2[1];

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
       stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($stop_id,undef,$stop_name,$stop_lat,
       $stop_lon,$zone_id,"0",$row->{stop_id},undef);

  } else {
   $stop_name = $stop_name . "FIXME!";

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
       stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($stop_id,undef,$stop_name,undef,undef
       ,$zone_id,"0",$row->{stop_id},undef);
  }

}


# Handling solo stops

$sth = $divadbh->prepare('SELECT S.hstnr AS stop_id, S.
    hstname AS stop_name, group_concat(tz.tzonen,"") as
    zone_id, HK.x AS stop_lat, (-1 * (HK.y - 6160000)) AS
    stop_lon, HK.plan AS plan
```

```
FROM Stop AS S LEFT OUTER JOIN Stop_tzonen as tz ON S.
    _AutoKey_=tz._FK__AutoKey_ AND S.input=tz.input LEFT
    OUTER JOIN Stop_hst_koord as HK ON S._AutoKey_=HK.
    _FK__AutoKey_ AND HK.plan="NBWT" AND S.input=HK.input
WHERE S._AutoKey_ NOT IN (SELECT SHS._FK__AutoKey_ FROM
    Stop_hst_steig AS SHS WHERE S.input = SHS.input)
GROUP BY stop_id, HK.x');
$sth->execute();

while (my $row = $sth->fetchrow_hashref()) {

  if (defined $row->{stop_lat} and defined $row->{stop_lon})
      {
   $stop_lat = $row->{stop_lat};
   $stop_lon = $row->{stop_lon};

   my @coords1=split(/\s+/, `echo $stop_lat $stop_lon |
       cs2cs -f "%.8f" $CS2CS_params{$row->{plan}}`);

   $stop_lon = $coords1[0];
   $stop_lat = $coords1[1];

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
       stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($row->{stop_id},undef,$row->{
       stop_name},$stop_lat,$stop_lon,$row->{zone_id},"0",
       undef,undef);

  } else {
   $stop_name = $row->{stop_id} . "FIXME";

   my $insertsth = $dbh->prepare('INSERT OR REPLACE INTO
       stops VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
   $insertsth->execute($row->{stop_id},undef,$stop_name,
       undef,undef,$row->{zone_id},"0",undef,undef);

  }
}


$dbh->commit;
$divadbh->commit;
```

BIBLIOGRAPHY

[1] Regionalisierungsgesetz. (Cited on page 2.)

[2] Gesetz über Urheberrecht und verwandte Schutzrechte. (Cited on page 78.)

[3] *Geographic data files* ISO Standard 14825. (Cited on page 24.)

[4] *Reference data model for public transport* EN Standard 12896:2006, 2005. (Cited on pages 17 and 23.)

[5] Personenbeförderungsgesetz. (Cited on pages 2, 3, and 77.)

[6] Identification of fixed objects in public transport (IFOPT), March 2013. URL http://www.dft.gov.uk/naptan/ifopt/. (Cited on page 24.)

[7] *Service interface for real time information*. CEN Standard OO278181. URL http://user47094.vs.easily.co.uk/siri/. (Cited on page 24.)

[8] 9th DIMACS Implementation Challenge. Shortest path, 2005. URL http://www.dis.uniroma1.it/challenge9/data/tiger/. (Cited on page 6.)

[9] Allgemeines Eisenbahngesetz. (Cited on page 77.)

[10] Aaron Antrim and Sean Barbeau. The many uses of GTFS data—opening the door to transit and multimodal applications. *Location-Aware Information Systems Laboratory at the University of South Florida*, 2013. (Cited on page 4.)

[11] Sibylle Barth. Nahverkehr in kommunaler Verantwortung: Der öffentliche Personennahverkehr nach der Regionalisierung. *Schriftenreihe für Verkehr und Technik*, (90), 2000. (Cited on page 2.)

[12] Hannah Bast. Car or public transport—two worlds. In *Efficient Algorithms*. Springer, 2009. (Cited on page 6.)

[13] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *18th Annual European Symposium on Algorithms*, 2010. (Cited on page 6.)

[14] Michael Bell and Ian Catling. The EC DRIVE programme halfway through. *Computing & Control Engineering Journal*, 1(6): 247–253, 1990. (Cited on page 22.)

[15] Zs Berki. The implementation of public transport data models in Hungary. *Acta Technica Jaurinensis*, 4(2), 2011. (Cited on page 24.)

[16] Bruno Bert, Kasia Bouree, and Lutz Staub. Transmodel, reference data model for public transport (European prestandard). 1996. (Cited on page 23.)

[17] Bliksem Labs. Rrrr rapid real-time routing, 2014. URL https://github.com/bliksemlabs/rrrr. (Cited on page 6.)

[18] Kasia Bourée, Bruno Bert, Pierre Pietri, and Bert Vervoort. Transmodel, June 2001. URL http://www.transmodel.org/en/cadre1.html. (Cited on pages 23 and 24.)

[19] Patrick Brosi. Real-time movement visualization of public transit data. Master's Thesis, 2014. (Cited on page 6.)

[20] US Census Bureau. TIGER/Line files, 2013. URL http://www.census.gov/geo/maps-data/data/tiger-line.html. (Cited on page 6.)

[21] Avishai Ceder. *Public transit planning and operation: theory, modeling and practice*. Elsevier, Butterworth-Heinemann, 2007. (Cited on pages 9, 10, and 96.)

[22] IG Collaboratory. Offene Staatskunst – Bessere Politik durch Open Government. *Internet & Gesellschaft Co:llaboratory, Berlin*, 2010. (Cited on page 2.)

[23] Pieter Colpaert. Route Planning Using Linked Open Data. In *The Semantic Web: Trends and Challenges*, pages 827–833. Springer, 2014. (Cited on page 6.)

[24] European Commission. Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, March 1996. (Cited on page 78.)

[25] European Commission. Regulation (EC) no 1371/2007 of the European Parliament and of the Council of 23 October 2007 on rail passengers' rights and obligations, October 2007. (Cited on page 77.)

[26] Jehiah Czebotar. GTFS data exchange. URL http://www.gtfs-data-exchange.com/. (Cited on page 33.)

[27] Ian Davis. Transit: A vocabulary for describing transit systems and routes. URL http://vocab.org/transit/terms/.html. (Cited on page 39.)

[28] Daniel Delling, Thomas Pajor, and Renato Fonseca Werneck. Round-based public transit routing. In *Sixth Annual Symposium on Combinatorial Search*, 2013. (Cited on page 6.)

[29] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. *Intriguingly simple and fast transit routing*, pages 43–54. Experimental Algorithms. Springer, 2013. (Cited on page 6.)

[30] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. (Cited on page 5.)

[31] Katrin Dziekan and Karl Kottenhoff. Dynamic at-stop real-time information displays for public transport: Effects on customers. In *Transportation Research Part A: Policy and Practice*, 41(6):489–501, 2007. (Cited on page 15.)

[32] Peter J. Elkins. Service management systems for public transport—the German approach. In *Vehicle Location and Fleet Management Systems, IEE Colloquium on*, pages 4/1–410. IET, 1993. (Cited on page 12.)

[33] PB Ellson and RGP Tebb. *Benefits and costs of providing additional information about urban public transport services*, 1981. (Cited on page 5.)

[34] B. Finn and K. Holmes. Drive 2 programme—area group 7 report. Public transport management and information. In *Towards an intelligent transport system. Proceedings of the first world congress on applications of transport telematics and intelligent vehicle-highway systems*, volume 6, November 30–December 3 1994. (Cited on pages 23 and 24.)

[35] Open Knowledge Foundation. Open Definition, November 2009. URL http://opendefinition.org/od/. (Cited on page 1.)

[36] Open Knowledge Foundation. The Open Transport Vocabulary, 2014. URL https://github.com/opentransport/vocabulary. (Cited on page 39.)

[37] Sunlight Foundation. Ten principles for opening up government information, August 2010. URL http://sunlightfoundation.com/policy/documents/ten-open-data-principles/. (Cited on page 1.)

[38] Avichal Garg. Public transit via google, December 2005. URL http://googleblog.blogspot.de/2005/12/public-transit-via-google.html. (Cited on page 31.)

[39] Christian P. Geiger and Jörn von Lucke. Open government data. In *Conference for E-Democracy and Open Government*, page 183, 2011. (Cited on pages 2 and 89.)

[40] Marco Felice Gennaro. Eigentums- und Nutzungsrechte im öffentlichen Verkehr. 2010. (Cited on pages 77, 78, and 86.)

[41] R. Goetz. Verbesserung der Leistungsfähigkeit von Systemen des öffentlichen Personennahverkehrs durch das Betriebsführungs- und Informationssystem BISON. *Straßen und Verkehr 2000 – Internationale Straßen- und Verkehrskonferenz Berlin*, September 1988. (Cited on page 18.)

[42] R. Goetz and R. Kirwa. Standardisiertes Datenmodell für den ÖPNV. *Heureka '90 – Optimierung in Verkehr und Transport*, April 1990. (Cited on pages 17 and 18.)

[43] Google. GTFS best practices—extended GTFS route types. URL https://support.google.com/transitpartners/answer/3520902. (Cited on pages 36 and 68.)

[44] Google. General transit feed specification reference. URL https://developers.google.com/transit/gtfs/reference. (Cited on pages 34 and 35.)

[45] Open Transport Working Group. stations.io, 2014. URL http://stations.io/. (Cited on page 38.)

[46] Shannon Guymon. Biking directions added to Google Maps, 2010. URL http://googleblog.blogspot.de/2010/03/biking-directions-added-to-google-maps.html. (Cited on page 33.)

[47] HaCon. Hafas – Die perfekte Verbindung zum Kunden, 2014. URL http://www.hacon.de/hafas. (Cited on page 24.)

[48] Chris Harrelson. Happy trails with Google Transit, 2006. URL http://googleblog.blogspot.de/2006/09/happy-trails-with-google-transit.html. (Cited on page 33.)

[49] Mark Headd. New thinking in how governments deliver services. In: *Beyond Transparency*, pages 277–287. Code for America Press, 2013. (Cited on pages 1 and 2.)

[50] SBB Infrastruktur. Download der öffentlichen Fahrplansammlung der Schweiz, 2014. URL http://www.fahrplanfelder.ch/fahrplandaten/. (Cited on page 25.)

[51] H. Kaufhold. Bison: Management and information system for mass transit authorities. *Glasers Annalen ZEV*, 110(12), 1986. (Cited on page 17.)

[52] Donald F. Kettl. *The transformation of governance: Public administration for twenty-first century America*. JHU Press, 2002. (Cited on page 1.)

[53] Nick Kizoom and Peter Miller. A Transmodel based XML schema for the Google transit feed specification—with a GTFS/Transmodel comparison. 2008. (Cited on pages 24 and 36.)

[54] Thomas Koch. GTFS feed for the Netherlands. Google Groups discussion thread, 01 2014. URL https://groups.google.com/d/msg/transit-developers/MbGRNM9keJ8/Z9ExR65YZHsJ. (Cited on page 87.)

[55] Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. What frustrates screen reader users on the web: A study of 100 blind users. *International Journal of Human-Computer Interaction*, 22(3):247–269, 2007. (Cited on page 4.)

[56] Felix Mata, Andres Jaramillo, and Christophe Claramunt. A mobile navigation and orientation system for blind users in a metrobus environment. In *Web and Wireless Geographical Information Systems*, pages 94–108. Springer, 2011. (Cited on page 3.)

[57] Bibiana McHugh. Pioneering open data standards: The GTFS story. In: *Beyond Transparency*, pages 125–135. Code for America Press, 2013. (Cited on pages 31, 33, and 86.)

[58] MentzDV. About us, 2013. URL http://www.mentzdv.de/englisch/company/about-us/. (Cited on page 27.)

[59] MentzDV. References, 2013. URL http://www.mentzdv.de/englisch/company/references/. (Cited on page 27.)

[60] Ted Miller and Tom Neumayr. App store sales top $10 billion in 2013, 2014. URL http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html. (Cited on page 1.)

[61] Facharbeitsgruppe Mobilitätskonzept. Mobilitätskonzept für Menschen mit Behinderung. 2009. (Cited on page 4.)

[62] Sebastian Mygo. *Barrierefreiheit im öffentlichen Nahverkehr: Eine Voraussetzung fuer die gesellschaftliche Teilhabe*. AV Akademikerverlag, 2012. (Cited on page 3.)

[63] Bengt Müller. Visualisierung von Fahrplandaten in Kartenanwendungen. *Der Nahverkehr*, (1):19–21, 2014. (Cited on pages 37 and 74.)

[64] DB Netze. Übersicht der Betriebsstellen und deren Abkürzungen aus der Richtlinie 100, February 2014. URL http://fahrweg.dbnetze.com/file/2361656/data/betriebsstellen.pdf. (Cited on page 38.)

[65] n.n. Bahn will offenen Rechner-Verbund schaffen. *Computerwoche*, 11/1988. URL http://www.computerwoche.de/a/bahn-will-offenen-rechner-verbund-schaffen,1157261. (Cited on page 24.)

[66] HM Department of Transportation. TransXChange. URL https://www.gov.uk/government/collections/transxchange. (Cited on page 24.)

[67] OpenStreetMap. Copyright and license. URL http://www.openstreetmap.org/copyright/en. (Cited on page 57.)

[68] Tim O'Reilly. Government as a platform. *Innovations*, 6(1):13–40, 2011. (Cited on page 1.)

[69] OVapi. Dutch GTFS repository. URL http://gtfs.ovapi.nl/. (Cited on page 87.)

[70] Google Transit Partners. Google transit extensions to GTFS. URL https://support.google.com/transitpartners/answer/2450962. (Cited on pages 36 and 61.)

[71] John Pucher and Stefan Kurth. Verkehrsverbund: the success of regional public transport in Germany, Austria and Switzerland. *Transport Policy*, 2(4):279–291, 1995. (Cited on pages 2 and 27.)

[72] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999. (Cited on page 1.)

[73] Ole Reißmann. Routenplaner im Web: Google sichert sich den Nahverkehr, September 2012. URL http://www.spiegel.de/netzwelt/web/google-bekommt-fahrplaene-fuer-den-nahverkehr-a-856802.html. (Cited on page 7.)

[74] Helen Roach. Public transport data modelling. In *Public Transport Information and Management Systems*, pages 6/1–6/3. IET, 1993. (Cited on page 9.)

[75] Helen Roach. The example of Eurobus/Transmodel in integrating applications for public transport. In *Vehicle Navigation and Information Systems Conference*, pages 497–502. IEEE, 1994. (Cited on page 9.)

[76] Wade Roush. Welcome to Google transit: How (and why) the search giant is remapping public transportation. *Community Transportation*, 2012. (Cited on pages 31 and 33.)

[77] Peter Sanders and Dominik Schultes. Engineering fast route planning algorithms, pages 23–36. In: *WEA'07 Proceedings of the 6th international conference on Experimental algorithms*. Springer, 2007. (Cited on pages 5 and 6.)

[78] Andreas Schildbach. Öffi. URL http://oeffi.schildbach.de/. (Cited on page 2.)

[79] Y. Shafranovich. RfC 4180. Common format and mime type for comma-separated values (CSV) files. URL http://tools.ietf.org/html/rfc4180. (Cited on pages 19 and 35.)

[80] Inês Soares and Paulo Matos Martins. Public transport standardization: A contribution to the state of the art review. Rio de Janeiro, July 15-18 2013. (Cited on page 23.)

[81] Brian Stelter. A Pulitzer winner gets Apple's reconsideration. The New York Times, 2010. URL http://www.nytimes.com/2010/04/17/books/17cartoonist.html. (Cited on page 1.)

[82] QGIS Development Team. QGIS, 2013. URL http://qgis.org/. (Cited on page 59.)

[83] SQLite Development Team. SQLite. URL http://sqlite.org/. (Cited on page 61.)

[84] WMK Tizani. A review of trip planning systems. 1992. (Cited on pages 4 and 22.)

[85] Verband Deutscher Verkehrsunternehmen. VDV-451: Dateiformat für die Datenübertragung zwischen ÖPNV-Anwendungen, 1999. URL http://www.vdv.de/oepnv-datenmodell.aspx. (Cited on page 18.)

[86] Verband Deutscher Verkehrsunternehmen. VDV-452: Standardschnittstelle Liniennetz, 2013. URL http://www.vdv.de/oepnv-datenmodell.aspx. (Cited on pages 20 and 22.)

[87] Kari Edison Watkins, Brian Ferris, Alan Borning, G. Scott Rutherford, and David Layton. Where is my bus? impact of mobile real-time information on the perceived and actual wait time of transit riders. *Transportation Research Part A: Policy and Practice*, 45(8):839–848, 2011. (Cited on page 15.)

[88] Stefan Wehrmeyer. Mapnificent. URL http://www.mapnificent.net. (Cited on page 4.)

[89] Stefan Wehrmeyer. HAFAS raw data format to GTFS conversion. Mailinglist Discussion, 05 2013. URL https://lists.okfn.org/pipermail/open-transport/2013-May/000221.html. (Cited on page 25.)

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 22, 2014 (`classicthesis` version 1.0).

DECLARATION

I hereby certify that this diploma thesis is my original work and has been written by me in its entirety. I have faithfully and properly cited all sources used in the thesis.

Ulm, 2014-05-23

Stefan Kaufmann