ulm university universität
uulm

Model-Driven Software Development with ACTIVECHARTS - A Case Study

Dominik Gessenharter

Ulmer Informatik-Berichte

Nr. 2011-08
Dezember 2011

Model-Driven Software Development with ACTIVECHARTS — A Case Study

DOMINIK GESSENHARTER
Institute of Software Engineering and Compiler Construction
Ulm University
Ulm, Germany
dominik.gessenharter@uni-ulm.de
(dominik@gessenharter.com)

Technical Report
December 2011

Abstract — **Modeling static structure by using UML[1] class diagrams is well supported by many tools and a de facto standard in modern software development. But generating code is still in a rudimentary stage. Many tools comprise extensible code generators, but the initial implementations of tool vendors are on a very low level.**

Although the claim that code generation is beneficial with respect to e. g. code quality and development time is generally accepted, state-of-the-art tool support is still far away from the possible.

But what is possible? To answer this question, we conducted a case study using a highly sophisticated code generator for the development of a sample project. With regard to the project's static structure and its features it is a small project. However, with respect to the applied modeling concepts it is a very complex system. Thus, we achieve to outline benefits of code generation for highly abstract modeling concepts in the context of an easily manageable project.

This report is geared toward an audience with advanced knowledge of UML modeling concepts and their semantics. It focuses neither on technical aspects of how to generate code nor on details of the generated code itself, rather than on the benefit of usably supporting modeling and code generation by development tools.

[1] Unified Modeling Language, Version 2.2 [22]

CONTENTS

1 INTRODUCTION

The main ambition of Model-Driven Software Development (MDSD) is to focus the development process on modeling. Defining a model covering a system's structure and behavior gives rise to automatic generation of the full runnable code [14]. OMG's[2] MDA[3] proposes model transformations and refinements across multiple stages before source code is generated [18]. The idea is to start from a very abstract model and to add details on every transformation step until a level of detail is reached that allows for generating the object code.

For being effective, MDSD requires technically mature tools supporting both, modeling as well as generating code from models. Whereas tool support for modeling has reached a quite good standard in recent years, code generation techniques have not kept pace. State-of-the-art code generators still only cover a subset of all UML elements.

This situation is non-satisfying: models are the result of analysis and design phases and hence should meet all requirements of the modeled system. But what are detailed models, analysis on a model or model checking useful for, if code is not automatically derived from it but hand written — a process that is cumbersome and error prone? Not a single characteristic of the input model is guaranteed to be present in hand written code. However, code produced by a generator — if the correctness of the generator is assured by either extensive testing or verification — definitely covers all details of the input model. Thus, any proved characteristic of the model will be featured by the code as well.

The main problem of UML is that although being widely-used in industry, it is not based on a proper theory and sufficient formalization, thus resulting in weak tool support as well as restricted possibilities to do analysis with models [3]. We can see this when looking at multiplicities: although being defined for attributes and association ends in the static structure of a model, their bounds may be violated by actions adding or removing values to structural features. Since UML is very popular and the de facto standard in modeling software systems [4], we aim for generating code from UML models considering all model details — in case of multiplicities even more rigorously than defined by UML itself.

This additional strength in the semantics is desired in our approach, since there is a smooth transition from modeling to coding behaviors, as detailed in the following section.

Document Conventions

The *italic* font is used to indicate a UML metaclass (e. g. *Activity*), the *italic sans serif* font is used to indicate elements contained in a model (e. g. *Person*) and the sans serif font is used to indicate elements of source code (e. g. Class). *Italic* font is also used for any other kind of emphasizing.

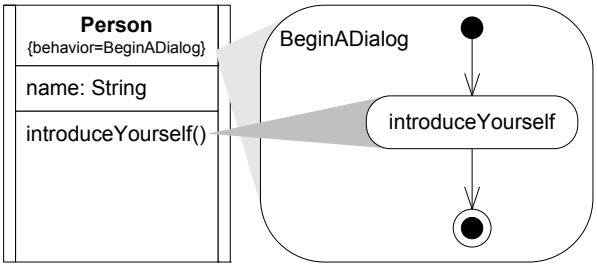
2. Object Management Group, http://www.omg.org
3. Model Driven Architecture, http://www.omg.org/mda

1.1 Our Approach at a Glance

With ACTIVECHARTS [20] we introduced an approach that combines modeling using UML2 and coding in a new way. The basic idea is to completely model static structures by means of class diagrams, whereas dynamic aspects of the system are partially modeled as activity diagrams and to some extent are expressed in code.

Initially, ACTIVECHARTS has been based on an activity interpreter that executes activities according to the token flow semantics defined by UML. An activity is associated to an active class by declaring it in the head compartment of the class (see Fig. 1). Its execution is automatically started on class instantiation by generating a constructor that calls the activity interpreter. Such a constructor is not shown in Fig. 1 as it is a quite technical detail and not intended to be edited by the developer.

CallOperationAction builds the connection from models to code: each *CallOperationAction* contained in an activity is associated to an operation, which might be associated to a behavior (by its attribute *method*). If so, this behavior is executed when the action executes. Otherwise, a method of the same name with conforming parameters must be present in the class implementation of the owning classifier that represents the context of the *CallOperationAction*, as shown in the listing contained in Fig. 1. By generating code for static structures, implementations of *CallOperationActions* get access to features of the context classifier. In Fig. 1, the attribute *name* of class *Person* is accessible within the method body of introduceYourself by this.name.




```
class Person{
  String name;
  public void introduceYourself(){
    // place user code here: e.g. print(this.name);
  }
}
```

Figure 1. Class *Person*, Activity *BeginADialog*, and the generated code for this model (constructor is omitted).

1.2 Using User Code with ACTIVECHARTS

In our approach, activities are used to model behavior to an arbitrary degree. We do not assume that it is feasible to cover all behavioral details of a system by models. For example, sort or search algorithms have been elaborated and implementations exist in many languages. It is much

more convenient to embed such implementations in code than to translate them into activities.

For this reason, we seamlessly combine user code with models. The code, e. g. a search or sort algorithm, makes up the body of a method which is represented as a *CallOperationAction* in an activity. By that, details like when to invoke the method, where to get inputs from and which action to provide outputs to are all captured by activity diagrams.

If applied adequately, models are clear and readable as well as the code, which comprises methods providing solutions to special, self-contained tasks by implementing possibly commonly known, highly sophisticated and efficiently designed algorithms.

1.3 Accessing Generated Code in ACTIVECHARTS

So far, we have shown why combining code and models might be useful and how user code is embedded in a behavioral model in ACTIVECHARTS. Now, we have a closer look at how to access generated code from user code.

Although activities are processed by an interpreter, code generation is part of our approach with respect to static structures. Beyond generating classes with members for attributes and association ends, methods are generated to access members as well as to manage associations with full consideration of multiplicity bounds. Depending on the upper bound of a feature, methods are generated for setting its value or adding a new value, for reading its value or its set of values, and for un-setting its value or removing one (or all) value(s).

The names for these methods are derived from the names of attributes and association ends by prepending the appropriate prefix *set*, *add*, *unset*, *remove*, or *get*.

In user code, structural features may be accessed by invoking these generated methods, which names are implicitly given by the prefix and the feature's name. If accessing a feature violates multiplicity bounds, exceptions are thrown. This effectively outruns the UML semantics whose specification is very weak in terms of considering multiplicity bounds.

1.4 Code Generation with ACTIVECHARTS

Besides interpreting activities, ACTIVECHARTS is also able to generate code from activities. This feature only slightly affects the code generation for static structures, since constructors must be adapted. The embedding of user code or access to generated code by user code is not affected at all. Regardless of whether interpreting an activity or running the code generated from it, each method implementing a *CallOperationAction* is invoked when the implemented action is to be executed. The invoked method may cause side effects, i. e. change the state of the context object by updating attribute values or sets of values.

Since the code generator for activities supports *StructuralFeatureAction*, access to the static structure is not only possible by invoking generated methods but also on the level of modeling. However, in either case, such side effects cannot result in an invalid system state because structural features are not directly accessed rather than generated methods are used. These methods throw an exception when being invoked by user code as well as from generated activity code if violating multiplicity bounds.

Handling exceptions depends on the execution technique: In user code, an exception can be caught. In activities, *StructuralFeatureActions* are replaced by *CallOperationActions* whose associated operations are the generated methods for accessing the concerning structural features. Since these operations may throw an exception, a *CallOperationAction* replacing an original *StructuralFeatureAction* may pass an exception on its termination. This exception can be handled in an activity by means of an *ExceptionHandler*, but not in user code.

1.5 Model-Driven Development with ACTIVECHARTS

Since the extent to which behavior is modeled is subject to the users arbitrary choice, the extent to which developing software with ACTIVECHARTS is model-driven also varies. The scope of this report is to show by means of a case study, that ACTIVECHARTS is an approach that may ease software development. Although our approach is not necessarily coupled with any specific methodology or development process, we outline how a typical course of a project using ACTIVECHARTS might look like. In order to not confuse readers with any kind of details concerning the development process itself, we use the simplest one — the waterfall model.

1.6 Structure of this Report

We start with a description of the sample project in Sect. 2 for which we give requirements in Sect. 2.1 and Sect. 2.2. An idea of how the system is operated is sketched by means of two use cases in Sect. 2.3, followed by a view on deriving activities thereof in Sect. 2.4.

In Sect. 3, we provide some insight into the generated code. In two separate parts, we first focus on code generated from models of the static structure in Sect. 3.1 before we briefly glance over the capabilities of code generation from activities in Sect. 3.2.

Since in ACTIVECHARTS models typically contain actions which have to be implemented manually, we dedicate Sect. 4 to the completion of a project by showing how to write the necessary code. In particular, we address how to consider GUI aspects (Sect. 4.1) and we present some algorithms which are essential for the business logic of the sample project (Sect. 4.2).

In Sect. 5, we provide some numbers to substantiate our claim of ACTIVECHARTS being a valuable approach for MDSD. We give a survey of related approaches in Sect. 6, before we finally end with our conclusions in Sect. 7.

2 TRAVELPLANER — A SAMPLE PROJECT

The presented sample project is designed to demonstrate the application of advanced modeling techniques on the basis of a self-explanatory system. It is called *TravelPlaner* and holds the timetables of public service vehicles of an imaginary city, initially based on the public transport of Munich. The purpose of the system is to find connections between stops provided as input, possibly additionally constrained by a desired time of departure or arrival.

2.1 Requirements — Structural Aspects

2.1.1 Public Transportation Vehicles

Apart from road vehicles, rail-bound vehicles are in use for the public transportation system. Buses are road vehicles as well as trams, which are running on rails placed within or beside lanes. The track gauge of trams is different to that of trains and undergrounds, which both are rail vehicles.

Vehicles offer seats and standing room for passengers. A reservation of a seat is only possible if traveling by train. Consequently, the capacity of other vehicles is not necessarily to be considered. This eases both, the model and the implementation of the system, in particular because the capacity of vehicles may vary depending on its year of construction or interior fittings.

Figure 2 shows the class diagram of a model covering the requirements of vehicles. Since the number of installed seats is only needed to be specified for trains, the class *Seat* is solely associated to the class *Train*.

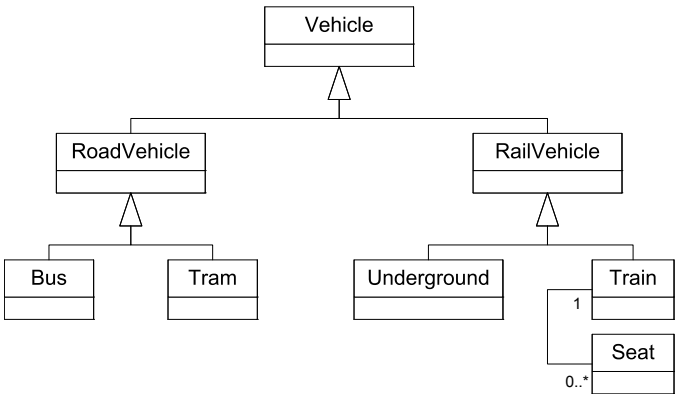


Figure 2. UML class diagram of the taxonomy of public transportation vehicles.

2.1.2 Public Transportation Network

A location where a vehicle stops for boarding and deboarding is called a *Stop*. Vehicles depart from a stop and drive to the next stop located on their route. The way from one stop to another stop is called a *Leg*. Each leg connects two stops without passing another stop.

As vehicles on their way from one stop to another may pass through other stops, a single leg or multiple legs make up a segment.

A stop is not further detailed. If many vehicles arrive at a stop at the same time and due to this the stop offers several holding positions, it is still considered to be an atomic element of the public transportation network. An exception to this is *Station*, which is a stop for rail vehicles and consists of at least one and at most 30 tracks. These requirements are modeled as shown in Fig. 3.

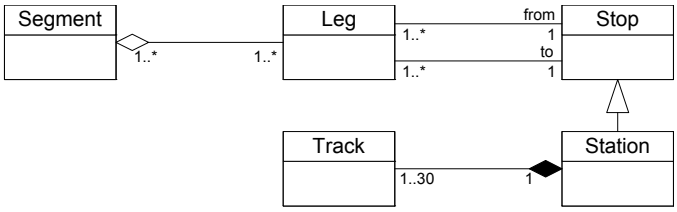


Figure 3. UML class diagram of the public transportation network.

2.1.3 Vehicle Operation in Public Transportation

In order to carry passengers from one stop to another, vehicles ride along segments. If a segment consists of only one single leg, no stop is passed without a halt, whereas a segment consisting of multiple legs represents a drive which passes through stops.

A *Ride* of a vehicle therefore is an association between a vehicle and the segment it rides along. Times of departure and arrival are attributes of each ride causing the *Ride* to be modeled as an *AssociationClass* as shown in Fig. 4.

A ride connecting a rail vehicle with a segment represents a *RailRide*, which is associated with *Tracks* of a station. One track represents the platform on which a train or underground departs, the other track represents the platform of the station where the train or underground arrives.

The class *TrainRide*, which is a specialization of *RailRide*, is not necessarily needed, but it has been introduced to support a clear structure for the reservation of seats, which is only possible in trains.

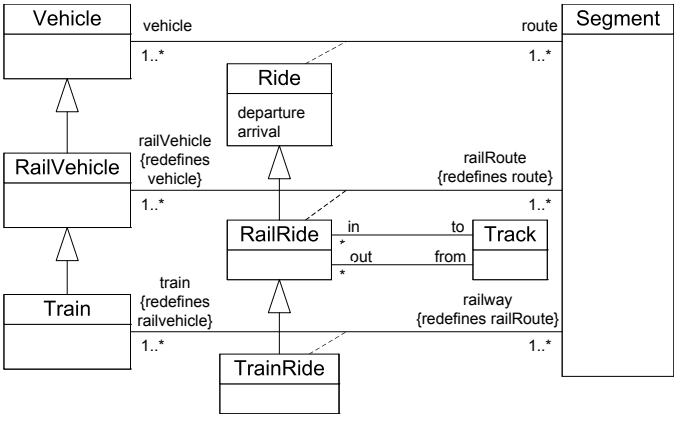


Figure 4. UML class diagram of the vehicle operation in the public transportation system.

2.1.4 Public Transportation Scheduling

The fact that each ride has a point of departure and a destination, both instances of *Stop*, as well as the two attributes departure time and arrival time facilitates the computation of time tables for each stop.

A time table for arrivals at a stop may be built by navigating from a stop to all incoming legs. From there, it is possible to navigate to all segments containing this leg. Of all these segments, only those are relevant in which the specific leg is the last leg of the segment. All rides connecting such a segment with a vehicle must be added to the time table of arrivals.

A technical detail prevents the application of this algorithm to build time tables at runtime: from a structural point of view, an association — as well as an association class — is not accessible from an instance participating in a link of an association at runtime. An association provides access to the instances at it's opposite ends, not to the link — or the link object in case of an association class — itself.

Therefore, time tables for each stop must be explicitly supported. For each stop, one *TimeTable* shall contain all arriving rides, another one all departing rides. A model for scheduling is given in Fig. 5.

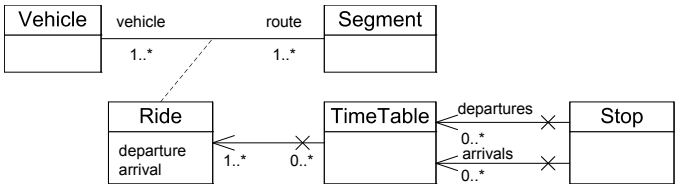


Figure 5. UML class diagram of scheduling.

2.1.5 Public Transportation Ticketing

Of course, using the public transportation system by passengers requires them to purchase a ticket. A ticket is valid for one or more rides as shown in Fig. 6.

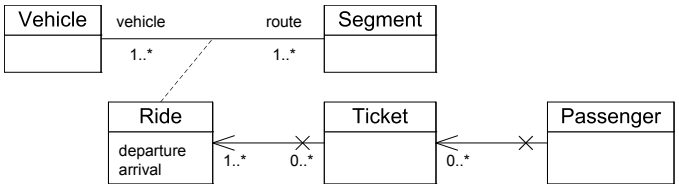


Figure 6. UML class diagram of ticketing.

2.1.6 Seat Reservations in Public Transportation

When travelling by train, passengers may reserve a seat. A seat reservation is valid together with a ticket. A ternary association between the classes *Reservation*, *Seat* and *TrainRide* can be used to model a seat reservation as shown in Fig. 7. The multiplicity of the end *reservation* must be [0..1], those of the remaining ends must be [0..*]. Furthermore, all ends must be marked as unique. Thereby we assure that for a specific ride, a seat is contained in at most one reservation and it is impossible to have two reservations of a seat on the same ride.

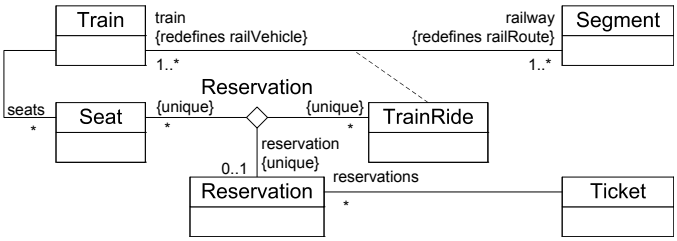


Figure 7. UML class diagram of seat reservations.

However, the model shown in Fig. 7 is too restrictive: Two reservations being associated with the same ride and the same seat should be possible if they are valid on different dates. Fixing this problem is possible by replacing the ternary association by a ternary association class with an additional attribute representing the validity date. All link objects of such an association class must be different by either its validity date or by the referenced objects.

Alternatively, the validity date could be contained as an attribute of the *Reservation* and the multiplicity of the reservation end could be changed to [0..*]. In this case, it must be assured that for each pair of a seat and a train ride, the date attribute of each associated reservation has a unique value.

Another solution to this problem is to explicitly model a *Day* class and include it by the association. The multiplicity of the association end connected to *Day* must be [0..1]. However, as each day for which a reservation exists must be represented by its own dedicated instance, this approach is quite costly with respect to resources.

In order to keep the sample project on a basic level, we do not consider the date of a trip. Assuming that all reservations relate to the same date, the model of Fig. 7 is sufficient.

2.1.7 Complete Static Structure

The complete static structure of *TravelPlaner* is shown in a single class diagram in Fig. 8. Association ends with no indication of navigability ($\times$ or $\rightarrow$) are considered to be navigable ends. A concrete public transportation network which we implemented for *TravelPlaner* is shown in Fig. 9.

In this report, we omit technical details of our model transformation for static structures [9], [10]. We also omit the presentation of the resulting model as well as we do not focus on details of the thereof generated code. For convenience, these details are hidden from the developer and consequently not further discussed in this report, which is concerned with the feasibility of our approach from a developer's point of view rather than with the magic behind the scenes.

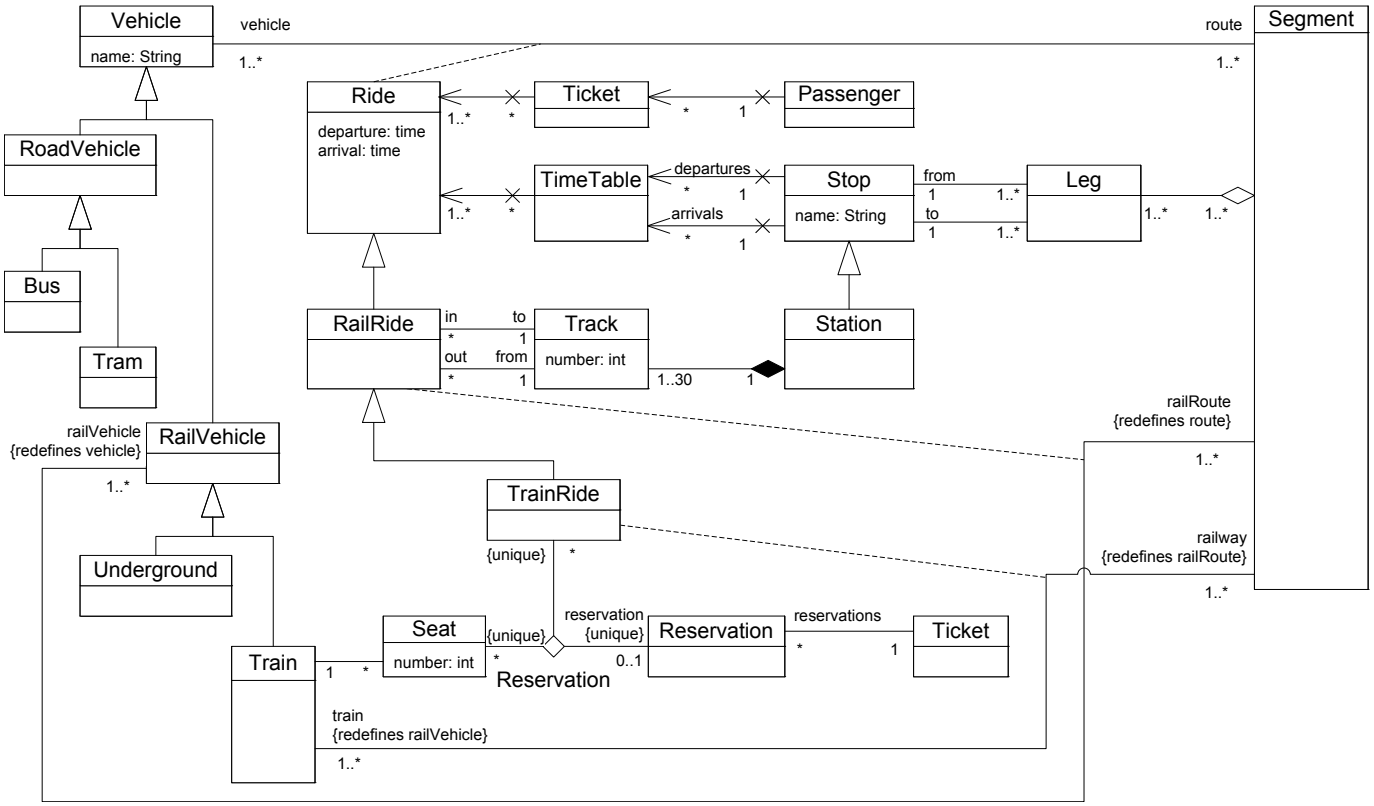


Figure 8. UML class diagram showing the complete static structure of *TravelPlaner*.

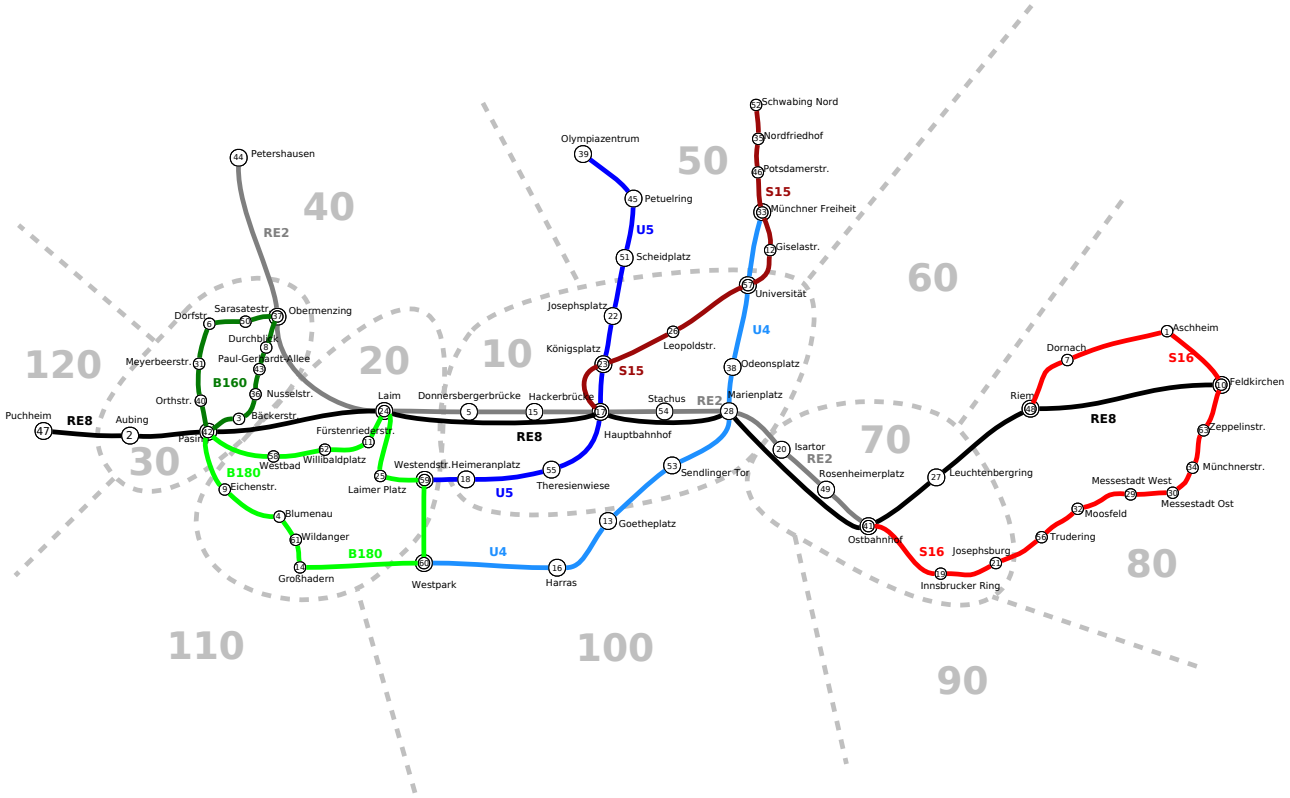


Figure 9. Concrete instance of a transportation network and its carriers.

2.2 Requirements — Behavioral Aspects

Although many behavioral features might be specified for a system like *TravelPlaner*, we concentrate on a single task — the sale of tickets.

Tickets are sold at terminals located at stops and stations, via Internet, and by a service hotline. A software module accessible by terminals and via Internet receives the input from the customer. When calling the hotline, the data are provided to the ticket agent by the customer. The ticket agent provides the data to the system. Requests from hotline service terminals, from terminals at stops and stations as well as requests received from the Internet are all represented as requests from a general terminal. The software module running on a terminal is called the GUI. The GUI is the part of the software that provides access to the system for a customer. The *System* communicates with terminals and processes and answers travel and seat reservation requests.

2.2.1 User Input

In the following, we describe the inputs that users provide to the system and depict suitable GUI snippets.

Departure and Arrival Data
A customer has to choose an origin and a time of departure as well as a destination stop and arrival time (Fig. 10).

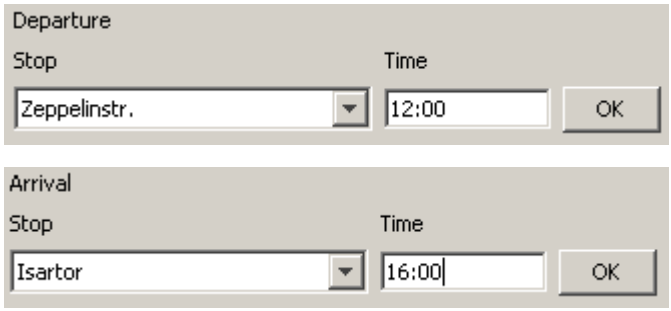


Figure 10. GUI panels for input of departure and destination data.

Selecting the Method of Payment
One of three options for paying is to be selected: paying cash, debiting a credit card or direct debit (EC). For using direct debit, the customer must be registered and his name must be selected (see Fig. 11).

Figure 11. GUI panel for selecting customer data and the method of payment.

Selecting a Connection
All connections between the origin and destination fitting the constraints given by departure and arrival time are presented. The user selects the connection for which he likes to purchase a ticket (see Fig. 12).

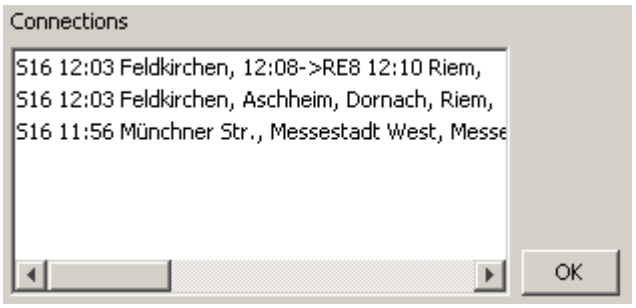


Figure 12. GUI panel for selecting a connection.

Seat Reservation
A customer may reserve one or more seats since he might be accompanied by children or — in case of being disabled — by another assisting person for whom no additional ticket is required. The maximum number of available seats is displayed (see Fig. 13).

Figure 13. GUI panel for reserving a number of seats.

Confirmation
Before actually purchasing a ticket, the user has to confirm the data of the requested travel. Instead of confirming, the user may cancel the request.

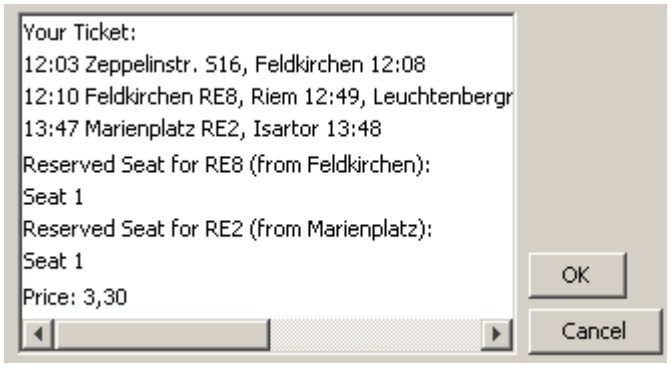


Figure 14. GUI panel for user confirmation.

2.2.2 System Actions

In this section, we sketch system actions by describing their purpose, the input they consume, and what output they produce. This is done on a quite abstract level rather than discussing details and algorithmic solutions to the tasks performed by system actions.

Checking the Method of Payment
Everyone using the public transport system may pay cash. Paying with credit card requires that valid card data are provided to the system. Paying by direct debit implies a prior registration of the customer and his grant for a direct debiting authorization.

Finding Connections
The system has to find all connections fitting the departure and arrival constraints. Inputs are origin, destination, time of departure, and time of arrival. The result is a list of lists of rides. Each list of rides represents a single connection. If no connection exists, the list of lists is empty. If changing vehicles during the trip is necessary, not all segments that make up the route are associated to the same vehicle, hence not all rides in the list representing this connection share the same value for the association end *vehicle*.

Checking the Permissibility of Reservations
If a requested connection contains at least one train ride, a number of seats may be reserved. The maximum number of reservable seats is the minimum of available seats of all concerned train rides.

Creating Reservations
If seats are to be reserved for a list of rides, a reservation object is created. Links of the association *Reservation* between *TrainRide*, *Seat*, and *Reservation* are created, each referencing the created reservation instance, a seat instance, and a train ride instance.

Deleting Reservations
Reservations are created before the purchase of the ticket is confirmed by the customer. Thus, it is guaranteed that a reservation option offered to the customer is still available after the confirmation step. However, instead of confirming, the customer may abort his request. In this case, the previously created reservation instance and the created links between instances of reservation, ride, and seat have to be destroyed.

2.3 Use Cases

In the following, we explain the process of purchasing a ticket by means of two use cases:
1. The user enters the origin and departure time, followed by the destination and arrival time. With this information, all connections — if any — can be found. All of them are presented and the user selects one of them. Now the system calculates the maximum number of seats available for reservation. If a reservation is possible, the user may enter the number of seats to be reserved. The reservation is processed by the system. Meanwhile, the user selects a method of payment, which is checked by the system. After processing the reservation request and checking the payment, all ticket data and costs are displayed. If the customer confirms the data and pays in cash, the ticket is printed. If not paying cash, the terminal connects to the bank for debiting the ticket price and afterward prints the ticket.

2. A user also might start by first entering the payment information, followed by the destination stop and arrival time before specifying when and where to start from. If connections fitting the time constraints are found, one of them is to be selected. After possibly entering the number of seats to be reserved, the ticket information is displayed. If — for whatever reason — the user does not confirm to purchase the ticket, no debiting is performed and any cash payment is returned. Furthermore, reservations — if any — are removed.

2.4 Deriving an Activity from Use Cases

When modeling the behavior of a system as an activity, the first step is sequencing actions. Before searching for connections, user input for specifying the origin and destination as well as departure and arrival times is required. All these pieces of information may be provided in any order since searching for connections cannot start before all values are present. Therefore, the GUI panels of Fig. 10 may both be displayed at the same time or one after another. Two concurrently executed actions, each of which showing one of the panels of Fig. 10, may be used to provide the user input to the search action as shown in Fig. 15.

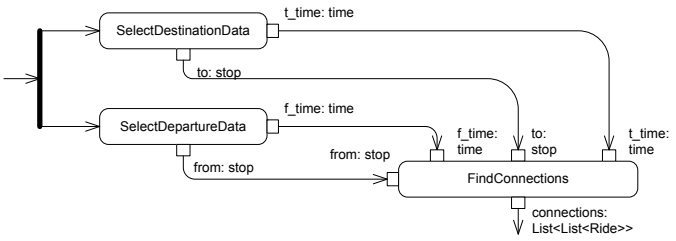


Figure 15. UML activity diagram: Concurrent actions for retrieving user input.

After searching for connections, one of them is selected and the permissibility of reservations is checked. If reservations are possible, the panel for specifying the number of desired seat reservations is to be displayed, receiving the maximum number of available seats as an input and providing the entered number as an output. If no reservation is possible, i. e. the maximum number of available seats is 0, this value is directly passed to the following action as shown in Fig. 16.

The complete activity is shown in Fig. 17. All actions contained in the activity are located in one of two swimlanes. Thus, ‚each action is associated to its context GUI or System. Actions needed for user inputs and system outputs including ticket printing are located in the GUI swimlane, and therefore, are executed on the terminal the user uses or a web client. The actions contained in the *System* swimlane are performed by the system itself, as described in Sect. 2.2.2.

The activity shown in Fig. 17 consists of three concurrently started flows. The first one executes the GUI action *Abort* which displays a cancel button. If this button is

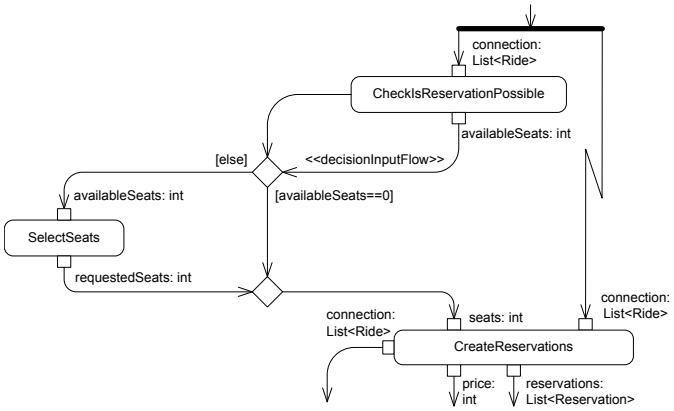


Figure 16. UML activity diagram: Decision of whether or not to allow input for a seat reservation.

pressed, the activity ends immediately and the terminal on which the activity executes will restart the execution.

Another flow initially executes actions for getting user inputs as shown in Fig. 15. Similarly, information about the method of payment are retrieved by a third flow executing *GetPaymentInfo*. Inputs provided by the latter two flows are concurrently checked by searching for connections and by checking the demanded method of payment.

While no connections are found, the input is not valid and actions demanding for user input are executed again. Note the *dummy* action, which is required for an activity flow conversion: the outgoing flow of *FindConnections* is an object flow, whereas the incoming flows of *SelectDepartureData* and *SelectDestinationData* must be control flows.

If the output *accepted* of *CheckPayment* is false, *GetPaymentInfo* is executed again. *CheckPayment* is located in an *InterruptibleActivityRegion* since it has the additional output *PaymentMethod,* which must be discarded if *accepted* is false. Only if the output *accepted* is true, outputs of *CheckPayment* are preserved until being consumed as inputs by *BuyTicket*.

If connections have been displayed, the connection selected by the user is returned to the system and reservations may be made under the conditions discussed above (see Fig. 16).

Note that executing *CreateReservations* results in the abortion of action *Abort* (by interrupting the concerning thread), since one of its input flows interrupts the activity region surrounding *Abort*. This causes the abort button on the GUI to disappear, so that the user can no longer terminate the activity. This is necessary, since from now on, terminating the activity requires some cleanup performed by *DeleteReservations*.

After reservations have been created, outputs of *CheckPayment* and *CreateReservations* together build the inputs for action *BuyTicket* which is executed next. It has two outputs, one of which is the ticket, the other one is a boolean result indicating whether the action has been confirmed or cancelled. In case of cancellation, the created reservations provided as an input to action *DeleteReservations* are deleted. Otherwise, *ProcessPayment* is executed indicating its success by its output *result*. In case of having executed successfully, the ticket is printed. Otherwise, no ticket is printed, reservations are deleted and the activity execution ends.

GUI actions as well as system actions are not detailed in the model of this case study. Their content is part of the implementation of the system. This is feasible, because searching for connections is closely related to the search for paths as covered by Dijkstra's algorithm [5]. The remaining system actions are less complex and easily implementable, as can be seen in Sect. 4.2.

Providing a GUI is best achieved by using one of the plenty available designer tools. The code that is to be inserted to complete the GUI comprises only few lines and is explained in Sect. 4.1.

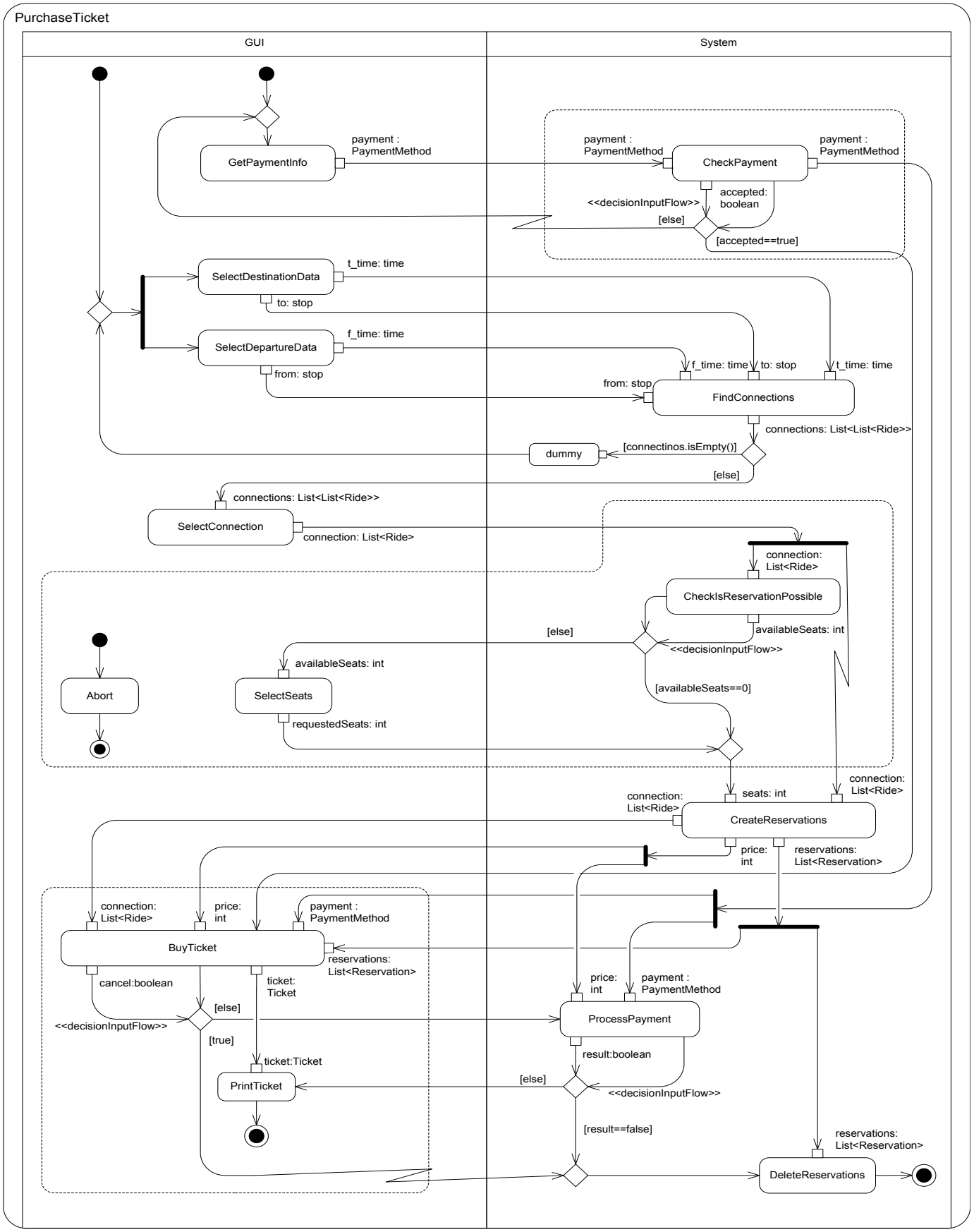


Figure 17. UML activity diagram: Complete behavioral model for the process of purchasing a ticket.

3 GENERATING CODE FROM MODELS

Before generating code, ACTIVECHARTS performs a model transformation. In this transformation, implementation classes for associations as well as some more advanced structural refinements are applied to make the code generation process, which is based on the transformed model, straightforward. Details to this transformation are presented in our previous work [9], which also includes a more technical view on it [10].

We published first results of our code generation approach in [11]. Afterward, *TravelPlaner* as well as the model transformation and code generation have become subject to minor changes, which finally result in slightly deviating numbers of generated code lines compared to the previously published ones.

3.1 Generating Static Structures

Apart from implementation classes needed for some technical reasons and abstract superclasses to contain generated code for each class of a model, empty classes are generated offering a location for user defined code. After this step, all classes with attributes and associations are available as source code and thus, if a powerful IDE is used, implementation of a system may be sped-up using code completion or similar features providing information about the structure and interfaces of existing classes.

Listing 1 contains the generated code for those classes shown in Fig. 2. These code classes are intended to contain generated code only, subclasses which may contain user written code are omitted. The class Train contains signatures of methods needed to manage the association between *Train* and *Seat*.

```
1  public class Vehicle{
2  }
3
4  public class RoadVehicle extends Vehicle{
5  }
6
7  public class RailVehicle extends Vehicle{
8  }
9
10 public class Bus extends RoadVehicle{
11 }
12
13 public class Tram extends RoadVehicle{
14 }
15
16 public class Underground extends RailVehicle{
17 }
18
19 public class Train extends RailVehicle{
20   public void addSeat(Seat seats){ ... }
21   public void removeSeat(Seat seats){ ... }
22   public Seat[] getSeat(){ ... }
23 }
```

Listing 1. Generated code for some of the classes of *TravelPlaner* representing basic concepts of modeling. For readability, technical details of the code are omitted and only publicly accessible features are presented.

Listing 2 contains some generated code implementing those classes participating in a higher order association as shown in Fig. 7. Again, subclasses for user code are omitted. Signatures for methods managing a higher order association are included. Furthermore, the association class *Ride* with its subclasses are roughly presented. Note that association ends can not be modified but only queried (ll 49–50). Methods for the redefined association ends of *RailRide* and *TrainRide* are not included in the listing but could easily be implemented as a delegation to modifier methods managing the association ends of *Ride*.

```
1  public class Seat{
2    public void addReservation(TrainRide t,
3           Reservation r)
4      { ... }
5    public void removeReservation(TrainRide t,
6           Reservation r)
7      { ... }
8    public TrainRide[] getTrainRide(Reservation r)
9      { ... }
10   public TrainRide[] getTrainRide(){ ... }
11   public Reservation[] getReservation(TrainRide t)
12     { ... }
13   public Reservation[] getReservation(){ ... }
14 }
15
16 public class Reservation{
17   public void addReservation(Seat s, TrainRide t)
18     { ... }
19   public void removeReservation(Seat s, TrainRide t)
20     { ... }
21   public Seat[] getSeat(TrainRide t){ ... }
22   public Seat[] getSeat(){ ... }
23   public TrainRide[] getTrainRide(Seat s){ ... }
24   public TrainRide[] getTrainRide(){ ... }
25 }
26
27 public class TrainRide extends RailRide{
28   public void addReservation(Reservation r, Seat s)
29     { ... }
30   public void removeReservation(Reservation r,
31          Seat s)
32     { ... }
33   public Seat[] getSeat(Reservation r){ ... }
34   public Seat[] getSeat(){ ... }
35   public Reservation[] getReservation(Seat s){ ... }
36   public Reservation[] getReservation(){ ... }
37 }
38
39 public class RailRide extends Ride{
40   public void setFrom(Track t){ ... }
41   public void unsetFrom(){ ... }
42   public Track getFrom(){ ... }
43   public void setTo(Track t){ ... }
44   public void unsetTo(){ ... }
45   public Track getTo(){ ... }
46 }
47
48 public class Ride{
49   public Vehicle getVehicle(){ ... }
50   public Segment getSegment(){ ... }
51 }
52
53 public class Segment{ ... }
```

Listing 2. Generated code for a higher order association and association classes (including specialization) of *TravelPlaner* representing advanced modeling concepts. Technical details are omitted for better readability.

3.2 Generating Dynamic Behavior

Code generation for behaviors is optional, since ACTIVECHARTS provides an interpreter for activities as well. In this report, we make use of the code generation for activities as it is roughly laid out in [12] and detailed in [13].

Code generated for activities does not notably affect the code representing static structures or user code, unless an activity should be started by user code. The technique of executing activities is transparent to other parts of a project — however, it obviously has impact on the deployment of an application, which must contain the interpreter if required.

Our current implementation makes activities an inner class of its context. Therefore, coupling code generation of static structures and activities is favorable: it facilitates the insertion of code for activity execution into classes while they are generated and thus spares persisting and parsing code files between both steps. In this regard, the transition from models to code currently consists of one single step.

Afterward, the coding of algorithms for actions is required to finish the implementation.

4 IMPLEMENTATION

TravelPlaner is intended to have a graphical user interface (GUI). Since modeling a GUI is not explicitly part of ACTIVECHARTS, the implementation of *TravelPlaner* requires a manually built GUI. Such a GUI may be composed of panels as shown in Sect. 2.2.1. In the following section, we first refer to issues related to implementing a proper GUI based on the actions running in its context. Thereafter we provide some insights to the implementation of those actions which are executed in the context of the system as introduced in Sect. 2.2.2.

4.1 Implementing the GUI

Implementing the GUI consists of two steps: providing the graphics and coupling graphics with input and output actions.

Assembling a GUI by arranging elements on the screen and by generating code that produces such an assembly of elements at runtime is supported by many GUI design tools, one of which is NetBeans[4]. The code generated by NetBeans must be enriched by methods coordinating the visibility of various parts of the GUI. Such methods are invoked when actions of the GUI swimlane are executed. Therefore, after the invocation of these methods, the control flow must not return to the caller before the user confirms his input.

A method implementing a GUI action must perform three steps: first, the GUI elements are shown (ll. 7 and 8 of Listing 3), then, the thread is suspended until user input is confirmed (ll. 10–16), and finally, the GUI elements are hidden and user input is returned (ll. 17–19).

```
1  class GUI {
2    private Thread dest;
3    private Stop to;
4    private int t_time;
5
6    public StopInt SelectDestinationData() {
7      jPnlStart.setVisible(true);
8      jBtnStartOK.setEnabled(true);
9      dest = Thread.currentThread();
10     synchronized (dest) {
11       try {
12         dest.wait();
13       }
14       catch (InterruptedException e) {
15         ...
16       }
17       jBtnStartOK.setEnabled(false);
18       jPnlStart.setVisible(false);
19       return new StopInt(to, t_time);
20     }
21   }
22 ...
```

Listing 3. Implementation of a GUI action.

The variable needed to suspend the thread by invoking its wait method is declared in line 2 of Listing 3. The two return values are wrapped in class StopInt (line 19), whose implementation is given in Listing 4. The variables passed as parameters for creating the StopInt instance must be defined in the GUI class. Values are assigned to these variables within the implementation of the listener added to the button which is pressed by the user to commit his input, as shown in Listing 5. For actually waking up the thread that waits for user input, the method input (see Listing 6) is invoked. Note that the code of this method could be in-lined into the anonymous listener class of each button for input submission.

```
public class StopInt {
  private Stop fStop;
  private int fInt;

  public StopInt(Stop stop, int i) {
    fStop = stop;
    fInt = i;
  }
  public Stop getStop() {
    return fStop;
  }
  public int getInt() {
    return fInt;
  }
}
```

Listing 4. Implementation of a wrapper class containing the values of two output pins.

```
1  private void addListeners() {
2    jBtnStartOK.addActionListener(
3        new java.awt.event.ActionListener() {
4      public void actionPerformed(
5          java.awt.event.ActionEvent evt) {
6        to = (Stop)jCbxTo.getSelectedItem();
7        int time = Integer.parseInt(
8          jTxtt_time.getText().replace(":",""));
9        t_time = (time / 100 * 60) + time % 100;
10       input(dest);
11     }
12   });
13 }
```

Listing 5. Implementation of a listener for converting user input and writing input data to class variables.

```
1  private void input(Thread t) {
2    if (t != null) {
3      synchronized (t) { t.notify(); }
4    }
5  }
```

Listing 6. Implementation of waking up a thread.

Note that code of wrapper classes, such as StopInt in Listing 4, is completely generated by our tool. The code of Listing 5 is partially generated by the GUI design tool (ll. 1–5 and 11–13). The completion by the user consists of assigning input values to variables (ll. 6 and 9), input conversion (ll. 7–9), and invocation of the input method.

Summing up, the code added by the user comprises variable declarations for threads to be suspended, values

4. www.netbeans.org

to be passed as return values, assignment of input values to those variables (including conversions or checks), and showing or hiding GUI elements.

Listings 3, 4, 5 and 6 together build the implementation for action *SelectDestinationData*. The actions *SelectDepartureData* and *GetPaymentInfo* can be implemented analogously.

The implementation of *SelectConnection* must contain some additional code to display the input as a list of options from which the customer may select a single item. Listing 7 shows that code.

In line 1, a hashtable is created to map each connection instance to its corresponding item in the list displayed by the GUI (line 13). In line 12, a string representation for each connection is created and all these strings are passed to the list element in line 16.

Listing 8 shows the code required to return the connection to which the selected list item is mapped. The variables connection and selectConnection are declared in Listing 7, lines 2 and 3.

The implementation of action *SelectSeats* is similar to the one of *SelectConnection* and not detailed here.

The implementation of *BuyTicket* is the most complex one, but we will only give a rough idea of it. Inputs of this action are a connection, the ticket price, the method of payment, and a list of reservations. Information about the connection and reservations are exclusively needed to display all ticket data for confirmation by the customer. The price is needed for the same purpose, as well as — in case of paying cash — for being able to decide whether the value of inserted coins equals or exceeds the ticket price. Non-cash payments have already been checked by the action *CheckPayment*. If the ticket is paid for and the user confirms the transaction, the return value of *cancel* is *false* and a ticket instance is passed to the action *PrintTicket* (for which we give no implementation, too). If the transaction is aborted, the return value of *cancel* is true and the return value of *ticket* is a null value.

```
1  Hashtable<String, List<Ride>>() rides;
2  List<Ride> connection;
3  Thread selectConnection;
4
5  public List<Ride> SelectConnection(
6        List<List<Ride>> llr) {
7
8     rides = new Hashtable<String, List<Ride>>();
9     int i = 0;
10
11    for (List<Ride> lr : llr) {
12       strings[i] = lr.toString();
13       rides.put(strings[i], lr);
14       i++;
15    }
16    jList1.setListData(strings);
17    jPnlSelectConnection.setVisible(true);
18
19    Thread t = Thread.currentThread();
20    synchronized (t) {
21       selectConnection = t;
22       try {
23          t.wait();
24       } catch (InterruptedException e) {
25          ...
26       }
27       jPnlSelectConnection.setVisible(false);
28       return connection;
29    }
30 }
```

Listing 7. Additional code for displaying action input values by GUI elements.

```
1  ...
2  jBtnSelectConnection.addActionListener(
3        new java.awt.event.ActionListener() {
4           public void actionPerformed(
5                 java.awt.event.ActionEvent evt) {
6              connection = rides.get(
7                    jList1.getSelectedValue());
8              input(selectConnection);
9           }
10 });
11 ...
```

Listing 8. Code for returning the selected connection.

4.2 Implementing System Actions

The implementation of system actions is easier than that of GUI actions since there is no additional code required for suspending threads or covering aspects not included in the model, like GUI elements.

System actions can be implemented as single methods or a set of methods from which some are invoked by others. The most complex action is that to find connections fitting a user's request. The Dijkstra-Algorithm may be adapted to suitably implement it. This algorithm is designed to find the shortest way between two nodes in a graph. In our case, the graph is the transportation network with its segments representing the edges and stops representing the nodes. The distance between two nodes might be the total travel time (ttt), the number of changes of the carrier vehicle, the ticket price or a combination of some or all of these or other aspects.

Since the customer usually wants to select one connection of a couple of options, the implementation should not only find the shortest connection — whatever the *shortest* in a concrete implementation means. However, presenting all connections that fit the given time constraints and are free of cycles probably is too much, as can be seen in a simple example:

EXAMPLE 1

Consider the request of traveling from *Pasing* to *Laim* departing not earlier than 10 a. m. and arriving no later than 11 a. m. (see Fig. 9 and Appendix B). Then, possible connections are:

B 180:

Pasing	Westbad	Willibaldplatz	Fürstenriederstr.	Laim	ttt
10:14	10:19	10:23	10:26	10:28	00:14
10:14	10:19	10:23	10:26	10:58	00:44
10:14	10:19	10:23	10:56	10:58	00:44
10:14	10:19	10:53	10:56	10:58	00:44
10:14	10:49	10:53	10:56	10:58	00:44
10:14	10:49	10:53	10:56	10:58	00:44
10:44	10:49	10:53	10:56	10:58	00:14

RE 8:

Pasing	Laim	ttt
10:34	10:40	00:06

Example 1 shows that presenting only the shortest connection with respect to travel time would return the connection of RE 8 lasting 6 minutes from origin to destination. Showing the shortest connections of each vehicle would add one of the two connections of B180 lasting 14 minutes from origin to destination.

But what criteria can be used to determine which of the two connections of B180 to present as an option? And how should a selection policy look like for connections where more than one vehicle is used?

A simple solution which is implemented in *TravelPlaner* is to find the shortest connection with regard to travel time for each departure at the origin. Applied to Example 1, the result of *FindConnections* consists of the following three connections:
B 180 departing on 10:14 a. m. arriving on 10:28 a. m.
RE 8 departing on 10:34 a. m. arriving on 10:40 a. m.
B 180 departing on 10:44 a. m. arriving on 10:58 a. m.

An implementation for finding connections as described above is given in Listing 9 and 10. An example for its application is given in the following:

EXAMPLE 2

Consider traveling from *Westendstr.* to *Universität* departing not earlier than 10:45 a. m. and arriving no later than 11:50 a. m. (see Fig. 9 and Appendix B). Then, possible connections are:

departure	via	arr.
U5 10:49	**RE2** 10:54[1] | **U4** 10:59[2]	11:02
U5 10:55	**S15** 11:00[3]	11:05
U5 11:01	**S15** 11:12[3]	11:17
B180 11:03	**U4** 11:12[4]	11:20
U5 11:07	**S15** 11:12[3]	11:17
U5 11:13	**S15** 11:24[3]	11:29
U5 11:19	**S15** 11:24[3]	11:29
U5 11:25	**S15** 11:36[3]	11:41
U5 11:31	**S15** 11:36[3]	11:41
B180 11:33	**U4** 11:42[4]	11:50
U5 11:37	**RE8** 11:43[1] | **U4** 11:47[2]	11:50

[1] Hauptbahnhof, [2] Marienplatz, [3] Königsplatz, [4] Westpark

```
public List<List<Ride>> FindConnections(Stop origin,
    Stop target, int departure, int arrival){
  List<Stop> visited = new LinkedList<Stop>();
  List<Ride> connection = new LinkedList<Ride>();
  List<List<Ride>> connections =
      new LinkedList<List<Ride>>();
  visited.add(origin);
  for (Ride r: origin.getSchedule().getDeparture()){
    connection = null;
    if ((r.getDeparture() >= departure)
        && (r.getArrival() <= arrival)){
      if (r.getTo() == target)
        connection = new LinkedList<Ride>();
      else
        connection = findFastest(
          r.getTo(), target, r.getDeparture(),
          arrival, visited, arrival);
    }
    if (connection != null){
      connection.add(0,r);
      connections.add(connection);
    }
  }
  return connections;
}
```

Listing 9. A simple algorithm for finding connections.

```
1  public List<Fahrt> findFastest(Stop origin, Stop target, int departure,
2        int arrival, List<Stop> visited, int firstdeparture) {
3
4     Hashtable<Stop, Integer> arrivalTimes = new Hashtable<Stop, Integer>();
5     Hashtable<Stop, Ride> via = new Hashtable<Stop, Ride>();
6     List<Stop> stops = new LinkedList<Stop>();
7     List<Ride> ll = new LinkedList<Ride>();
8
9     stops.add(origin);
10    Stop stop = origin;
11
12    arrivalTimes.put(stop, 0);
13    while (stop != null) {
14       if (stop != target) {
15          // set departure time to the value when we reach this node
16          if (via.containsKey(stop)) {
17             departure = via.get(stop).getArrival();
18          }
19          for (Ride r : start.getSchedule().getDeparture()) {
20
21             if (r.getDeparture() >= departure && r.getArrival() <= arrival){
22                stop destination = r.getTo();
23                if (visited.contains(destination))
24                   continue;
25
26                int traveltime = r.getArrival() - firstDeparture;
27                if (!arrivalTimes.containsKey(destination)) {
28                   stops.add(destination);
29                   arrivalTimes.put(destination, traveltime);
30                   via.put(destination, r);
31                } else {
32                   if (arrivalTimes.get(destination) > traveltime) {
33                      arrivalTimes.put(destination, traveltime);
34                      if (!stops.contains(destination))
35                         stops.add(destination);
36                      via.put(destination, r);
37                   }
38                }
39             }
40          }
41       }
42       stops.remove(0);
43       if (stops.isEmpty())
44          stop = null;
45       else
46          stop = stops.get(0);
47    }
48    stop dest = target;
49    List<Ride> connection = new LinkedList<Ride>();
50    while (dest != origin) {
51       if (via.get(dest) == null)
52          break;
53       else{
54          connection.add(0, via.get(dest));
55          dest = via.get(dest).getFrom();
56       }
57    }
58    return connection;
59 }
```

Listing 10. The Dijkstra Algorithm adapted to *TravelPlaner*.

Note that all departures of U5 — which leaves every 6 minutes — and both departures of B180, leaving every half an hour, are contained in the list of Example 2. However, to each departure only the fastest connection is shown.

For connections with U5 and S15, changing from U5 to S15 is possible at *Hauptbahnhof* and *Königsplatz*. The reason why only connections are presented where changing is proposed at *Königsplatz* can be seen in Listing 10 (ll. 27–30). When searching for connections, *Königsplatz* is reached earlier by riding there directly with U5 than by previously changing to S15 at *Hauptbahnhof*. Therefore, the latter option is discarded, although it would result in a connection with an equal travel time compared to the first option, which is included in the set of found connections. In this regard, the implementation is not optimal but does not invalidate our case study either, since additional optimizations could be applied.

After the user has selected a connection, it is determined whether or not a seat reservation is possible by the implementation of action *CheckIsReservationPossible*. At first, TrainRide instances are filtered out of all Ride instances of the given connection. After this, the minimum of available seats of those TrainRide instances is identified and returned, as shown in Listing 11.

The number of available seats is passed to a GUI action, if any. After its termination, this action returns the number of seat reservations requested by the customer. In order to be able to comply with the customer's request, reservations are processed immediately, even if the method of payment has not yet been selected, and therefore, the action *BuyTicket* is delayed until the missing information is provided to the system. If the method of payment is not accepted or the customer cancels the transaction, the reservations are deleted later.

The action *CreateReservations* as given in Listing 12 calculates the ticket price, creates the requested reservations and returns the price, a list of created reservations as well as the connection which has not been read or modified. The connection is not directly passed from action *SelectConnection* because it must be destroyed if the *abort* signal is received after the connection has been selected but before reservations are created. Handing over the connection by *CreateReservations* assures that the connection offered to *BuyTicket* is the same as the connection for which the price has been determined and reservations have been created, if any.

If reservations have been created but the transaction has not been confirmed, those reservations must be deleted. This is done by action *DeleteReservations* for which an implementation is given in Listing 13.

If paying cashless, the payment is processed by the system by either debiting a credit card or an account. An implementation of this is omitted here since it does not contribute to a better understanding of *TravelPlaner* or ACTIVECHARTS. If processing the payment does not succeed, reservations are deleted, in case of success, the ticket is printed. Cash payments are processed by the action *BuyTicket*, which does not terminate until the full amount is paid. *ProcessPayment* then always succeeds.

```
1  public int CheckIsReservationPossible(
2        List<Ride> connection) {
3     int free_seats = Integer.MAX_VALUE;
4     int available = 0;
5     for (Ride r: connection) {
6        available = 0;
7        Vehicle v = r.getVehicle();
8        if (r instanceof TrainRide) {
9           Train t = (Train)v;
10          TrainRide tr = (TrainRide) r;
11          Seat[] allSeats= t.getSeat();
12          for (Seat s: allSeats)
13             if (s.getReservation(tr).length == 0)
14                available++;
15       }
16       if (available < free_seats)
17          free_seats = available;
18    }
19    return free_seats;
20 }
```

Listing 11. Implementation of action *CheckIsReservationPossible*.

```
public ConnectionPriceReservation CreateReservations(
      int seats, List<Ride> connection){

   List<Reservation> reservations =
         new LinkedList<Reservation>();
   Reservation res = new Reservation();

   for (Ride r : connection) {
      if (r instanceof TrainRide) {
         int temp = seats;
         Train t = (Train) r.getVehicle();
         TrainRide tr = (TrainRide) r;
         Seat[] allSeats = t.getSeat();
         for (Seat s: allSeats) {
            if (s.getReservation(tr).length == 0) {
               res.addReservation(s,tr);
               reservations.add(res);
               temp--;
            }
            if (temp == 0) {
               break;
            }
         }
      }
   }
   // calculate price in cent
   int price = ...;
   return new ConnectionPriceReservation(
         connection, price, reservations);
}
```

Listing 12. Implementation of action *CreateReservations*.

```
public void DeleteReservations(
      List<Reservation> res) {
   for (Reservation r: res)
      for (Seat s: r.getSeat())
         for (TrainRide tr: r.getTrainRide(seat))
            r.removeSeatReservation(s,tr);
}
```

Listing 13. Implementation of action *DeleteReservations*.

5 EVALUATION

In this case study, ACTIVECHARTS was used to implement *TravelPlaner*. There is no other implementation of *TravelPlaner* to which the results of this case study could be compared. Accordingly, it is difficult to state numbers as evidence of advantages of MDSD. If numbers are related to each other, e. g. the quantity of lines of generated code and the quantity of lines of handwritten code, the percentage of code generation might be slightly overestimated since a completely hand written implementation might result in less lines of code for the entire project:

Let LOC_H be the number of lines of hand written code and LOC_G the number of lines of generated code. Then

$$\frac{LOC_G}{LOC_H + LOC_G} \times 100$$

is the percentage of generated code.

Now let LOC_X be the number of lines of code of a completely hand written implementation of the same project. Then if

$$LOC_H + LOC_G > LOC_X$$

holds, the percentage of code generation is overestimated.

Whether or not the assumption $LOC_H + LOC_G > LOC_X$ holds or not depends on how efficiently a code generator may handle special cases. It is generally believed that humans need less lines of code since they can intelligently adapt their implementation to the given problem, whereas implementations of tools normally are applicable to the general case with only limited capabilities of adjustment and tuning to the concrete, special case.

However, since we do not think that MDSD or code generation itself is questionable rather than its concrete implementation and tool support is arguable, we compare the results obtained by using ACTIVECHARTS with results achieved by using other MDSD tools.

The basis of our validation therefore is the model. It is the result of early project phases and is created on the basis of requirements and design decisions. The nature of a model driven approach is to use this model, which has been created during the developing time, at runtime, either by interpreting or by running generated code. ACTIVECHARTS uses a code generator that is able to generate the whole code to implement the static structure as well as to implement control and data flow of activities. Actions have to be implemented by the developer.

Using ACTIVECHARTS is beneficial because it generates code for advanced modeling concepts like higher order associations or association classes, even if generalization is applied to them. This may encourage the use of those structural elements in software engineering since working with them is quite comfortable, as can be seen in Listing 13.

By generating code for the higher order association between *Reservation*, *TrainRide*, and *Seat*, actions using this association can be succinctly implemented by using generated access methods. In Listing 13, only four lines of code are required to delete a set of links of association *Reservation*. The implementation of action *CheckIsReservationPossible* (see Listing 11), which reads the association and calculates the minimum number of available seats for a set of *Rides*, amounts to about 15 lines of source code (method signatures and lines containing only a closing bracket } are not counted). For creating links of that association, the implementation of action *CreateReservations* (see Listing 12) consists of 20 lines of code.

The given code for the implementation of action *FindConnections* (see Listings 9 and 10) consists of only 65 lines of code. Even if a higher sophisticated algorithm is desired, the implementation probably will be manageable compared to the large number of code lines required to implement *Ride* and its subclasses, *Segment*, *Leg*, and *Stop* with all the associations relating these classes. This is a point where — in our view — it clearly can be seen that both modeling and coding have pros and cons and it is up to the tools to combine both in a powerful way.

The complex structure of *TravelPlaner*, which to implement requires many lines of code, can be modeled with few classes and relationships. In contrast, algorithms working on an implementation of a structure often can be written in few lines of code thus being better readable than a complex model of it like an activity. This can be seen in Fig. 18, where a single method of about 20 lines of code contributing to implement *FindConnections* is modeled as an activity. We consider it to be no more readable than code and by no means easier to develop. Furthermore, the diagram does not completely model the method since the action *prepend* and operators $<$ and $\leq$ are not standard UML elements and therefore need their own — trivial but compulsory — implementations.

GUIs are not explicitly supported by ACTIVECHARTS. There are many tools available for this purpose and including GUIs in ACTIVECHARTS somehow resembles reinventing the wheel. However, *TravelPlaner* shows that, with minimal effort, a GUI generated by another tool can be extended to provide actions for user input and output. Listing 3 and Listing 5 evidence that approximately 15 lines of code are enough to manage the visibility of GUI elements and to suspend a running thread until input data are available.

Depending on the complexity of a GUI and the actions to implement, interplay of many GUI elements as well as intricate tests and conversions of input may increase code size considerably. However, the basic principle of including a GUI and engaging GUI actions is independent thereof and neither complicated nor difficult at all.

5.1 Generated Code

The implementation of the static structure of *TravelPlaner* according to the class diagram of Fig. 8 amounts

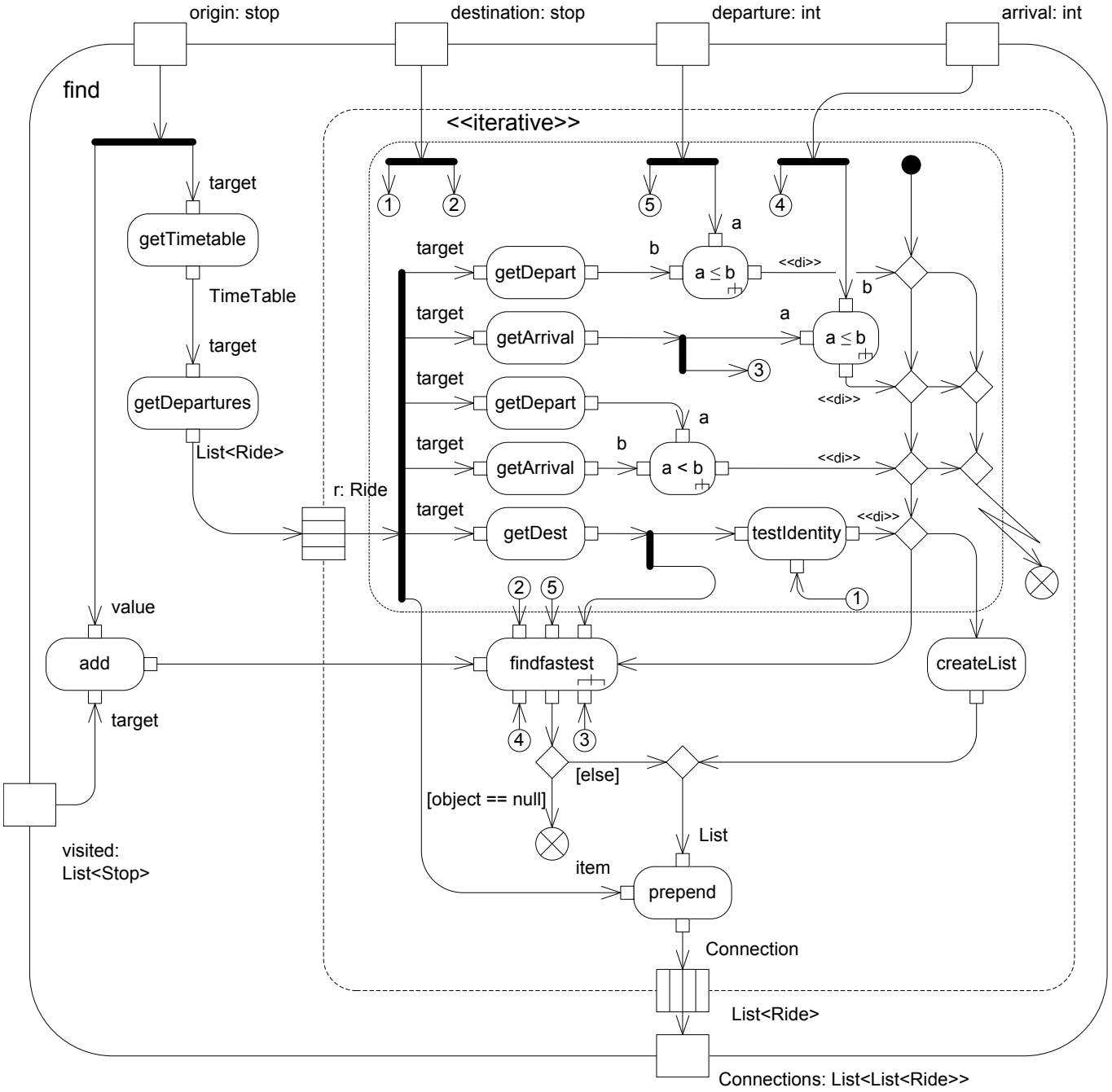


Figure 18. Activity modeling the method List<List<Ride>> FindConnections(Stop origin, Stop target, int departure, int arrival) of Listing 9. For better readability, the stereotype <<decisionInputFlow>> is abbreviated as <<di>>.

to 1727 lines of code. The implementation of activity *PurchaseTicket* of Fig. 17 amounts to 777 lines of code, that of the GUI amounts to 893 lines of code.

Concerning structure, this part of software is fully generated from the input model even though this implementation is not necessarily optimal with respect to code size. However, we do not believe in possible savings of more than 10% if implemented by hand.

Activities probably might be implemented with fewer lines of code by humans. The benefit in this field is not the low number of lines of code the generator produces, rather than the fact that the whole activity is translated into code. The only thing that has to be added manually is the implementations of actions if they are not formally detailed in the model, e. g. by a called behavior or an action specification using some kind of action language.

If the modeler succeeds in covering the entire behavior by models with all actions implementing small, self-contained tasks, errors are very unlikely. The implemented algorithms then may be extensively tested or their correctness might even be proved. Passing outputs of actions to inputs of other actions as well as

the sequencing of action executions is covered by the generated code. An evaluation of the code generator itself is ongoing work in the context of a PhD thesis.

ACTIVECHARTS does not cover modeling GUIs, but code generated by other tools can easily be adapted.

5.2 Hand-Written Code

In Listing 9, a part of the implementation of *FindConnections* is listed. The code produced by ACTIVECHARTS to implement the activity of Fig. 18 — which actually is a model of that algorithm — surely would be considerably larger and less efficient. Similarly, when using an interpreter, modeling the action *FindConnections* results in large overhead compared to implementing it in code. By providing the following numbers comparing the amount of lines of generated code to that of hand-written code — which of course are dependent on the concrete project, the models and the implementations of actions — we want to give an idea of what percentage of code generation definitely is achievable. As mentioned, these are rough numbers, that, to be more expressive, have to be compared to a completely hand-written project implementation.

Table 1 lists all system actions and the number of lines of code (LOC) needed for their implementation. Analogously, Table 2 lists all GUI actions. There, LOC is divided into the implementation of GUI updates such as showing or hiding elements and into the implementation of action listeners of GUI elements.

Even though aspects such as checking and processing payments or printing tickets have not been fully implemented, it can clearly be seen that code generators can create the bulk of a system's implementation.

In the case of *TravelPlaner* 3397 lines of code are generated whereas 266 lines of code are hand-written, which is approximately 8.5%. Since a detailed implementation of some actions (*CheckPayment*, *ProcessPayment*, *BuyTicket*, and *PrintTicket*) have been excluded in our case study, additional code must be considered. Even if we assume that this might double the number of lines of hand-written code, the percentage of generated code still is about 85%. If we additionally assume possible savings achievable for implementing the static structure of 10% and another 30% of savings due to a more enhanced implementation of activities, the size of generated code decreases to 2099 lines. By those adjustments, we try to eliminate overestimation of code generation. Accordingly, a supposable lower bound for the percentage of code generation is about 75% of the entire applications source code.

Action	LOC
CheckPayment *	1
FindConnections	77
CheckIsReservationPossible	18
CreateReservations	26
DeleteReservations	5
ProcessPayment *	1
dummy	0
	128

* not completely implemented, returns a default value

Table 1
System actions and size of their implementation (LOC)

Action	LOC GUI updates	LOC Action Listener
GetPaymentInfo	16	3
SelectDestinationData	16	5
SelectDepartureData	16	5
SelectConnection *	25	2
SelectSeats	16	2
BuyTicket **	16	–
PrintTicket **	16	–
Abort	13	1
	121	**17**

* requires customization of toString (approx. 20 lines of Code)
** not completely implemented, returns a default value

Table 2
GUI actions and size of their implementation (LOC)

Using the basic implementations of other tools results in about 150 to 500 lines of generated code for static structure. This is less than 30% of the output of our tool. Code generation for activities fails since the applied modeling elements, in particular *InterruptibleActivityRegion*, are not supported.

Of course, we must admit that other projects may be less suited for our approach, but since *TravelPlaner* has not been especially designed for our approach, it is reasonable to state the following two claims:

- Our approach facilitates the generation of large parts of a software from input models, although the exact proportion may vary depending on the concrete project.
- The applicability of our approach is not the same for all projects. Some are better suited, others may disqualify themselves, e. g. if the system cannot be well modeled using UML.

6 Discussion & Related Work

Systems which are subject to a case study often tend to be unrealistic. They are inadequate in size and complexity and their implementation does not suffer the negative impacts of time to market pressure, budget limitations or other considerable — but generally neglected — influences. Most of these aspects are disregarded in this case study as well, except for complexity of the system.

Although *TravelPlaner* is small, its structure as well as its behavior is not trivial. Similar case studies are rarely seen and focus on other characteristics of systems or generated code. Usman and Nadeem [21] report about a case study generally related to the one presented here, but underlying models are very simple and far away from the complexity we deal with. We make use of advanced modeling concepts which may significantly contribute to a clear and manageable model, in particular if the system under development is more complex and comprehensive than *TravelPlaner*. The purpose of this case study is not to attest this effect to highly abstract modeling concepts rather than to take these concepts as granted and to examine options of profitably using them.

Therefore, our ambition is to present a system that is easily understandable without the need for additional, domain specific knowledge. The models describing this system are small, but contain the modeling elements whose application with regard to code generation is of particular interest to us. In this regard, *TravelPlaner* is a suitable sample project that facilitates to concentrate on the research question to be answered: to what extend can a model contribute to the final implementation if it is translated into code, and can highly abstract modeling concepts be included and automatically be processed?

To answer this question, it is necessary to identify the semantic meaning of all modeling concepts applied in the source model. The semantics of structural modeling, particularly with regard to associations, is addressed in many publications. Commonly used implementations of associations consist of a pair of references [1], [6], [7], [8], although this approach is practical for some cases of binary associations only. In general, implementing associations requires more complex code [9].

As a consequence, models in which advanced modelling concepts are used, e. g. higher order associations, compositions, association classes and maybe even specialization of association classes, are difficult to implement. Considering this difficulty, implementation by hand is cumbersome and error prone. Good tool support, as is provided by our prototypical implementation of ActiveCharts, is therefore a crucial point for MDSD.

With *TravelPlaner*, we give an example that shows how advanced modelling concepts can — if supported by development tools — ease modeling without facing the developer with complex and hard-to-read code. To this extent, *TravelPlaner* is evidence of the applicability and feasibility of highly sophisticated code patterns [10] for structural modeling.

The semantics of behaviors, too, has been researched, e. g. by formalizing it using different formalisms. Closely related to our work is Crane [4]. However, there is a contradiction in the semantics of *Property* and *Action* that is neither discussed by Crane nor by other researchers: On the one hand, the semantics of upper and lower bounds of a property is defined to constrain the size of the set of values a property may hold. On the other hand, the semantics of actions that add/remove values to/from such a set of values (i. e. actions specializing *StructuralFeatureAction*) is not specified to strictly consider these bounds. The conflict between the execution state of an activity demanding to execute an action and a system state that would become invalid by executing this action is not discussed by Crane, since associations in general are not supported. Consideration of multiplicity bounds of attributes and the semantics of *AddStructuralFeatureValueAction* and *RemoveStructuralFeatureValueAction* is not included, either.

We face these aspects by generating modifier methods, which strictly consider multiplicity bounds for association ends and class attributes. To this extent, our claim is to always consider structural validity of a system at runtime. Furthermore, by generating methods for accessing class attributes and associations, we support all actions which are specializations of *StructuralFeatureAction* for accessing properties in a UML model. Even if behavior is not modeled but implemented in code, our approach is still feasible since structural features are accessible by directly invoking the appropriate access methods.

A wider support of behavioral modeling is achieved by supporting activities [13]. With *TravelPlaner* we also cover this aspect of a system and include the implementation of control and object flows, concurrency and *InterruptibleActivityRegion*. Thus, we do not focus only on basic UML concepts included in *Semantics of a Foundational Subset for Executable UML Models* (fUML)[19]. We also are not limited to a certain context in which behavioral modeling is tailored to very specific needs like it is in UML based Web-Engineering [16], [15]. Furthermore, we consider complex flow situations which are not covered by the translation of activities to the Esterel programming language proposed by Bhattacharjee and Shyamasundar [2].

With regard to behavioral modeling, Executable UML [17] addresses how to express behavior using the UML action semantics and an action language. Although checking multiplicity bounds for association ends and for attributes is explicitly stated to be an important step while modeling, Executable UML relies on the action semantics that does not consider multiplicities as detailed as it is defined for structural features. Furthermore, behavioral modeling or specifying actions by the use of an action language is still required.

To our best knowledge, it has not yet been aimed for an integration of the semantics described in the language unit *Classes* of UML and the semantics of UML actions. We consider the semantic specification of both language

units by providing methods, which consider structural validity and serve as implementations for specializations of *StructuralFeatureAction*. When generating code, specializations of *StructuralFeatureActions* are mapped to these methods.

Our approach extensively uses *CallOperationActions* which are not associated with a behavior, i. e. they do not have a *method*. Therefore, handwritten code must be provided as an implementation for them. Nevertheless, a considerable part of source code is generable. This includes the complex and hard-to-implement concepts of structural and behavioral modeling.

The presented example also shows another benefit associated with MDSD: a better maintainability. Even though not substantiated by numbers, it is obvious that changing the *workflow* of purchasing a ticket can easily be done by re-arranging actions in the corresponding activity. Such changes do not affect the static structure. If inputs and outputs of actions remain unchanged, their implementations are not affected as well. It is very likely much more time consuming to identify the locations where source code must be modified in order to achieve another sequencing of input screens on a terminal, to implement and finally to test the modifications.

7 CONCLUSION

According to our experience, developers often avoid using modeling concepts that cannot be mapped easily to source code. In order to encourage developers to use advanced modeling concepts, code generation for these concepts must be offered and features of these concepts must be easily accessible.

In UML, access to association ends, to instances of association classes, or to values of class attributes is specified by actions. Accordingly, a tool must support actions and therefore also activities, which are composed of actions and flows.

Unfortunately, if only structural aspects are modeled, no actions will be applied and therefore, access to structural features is not defined. Generated methods can act as an interface to access static structure. In particular, even complex, advanced modeling concepts can easily be accessed by invoking such methods. However, this code should be hidden in order not to confuse the developer and the provided methods must be convenient.

Our tool proves that UML with its high level of abstraction is well suited for MDSD and tool support complying to the claims above is achievable. Such tool support replaces the cumbersome and error prone implementation by hand and thus makes the application of advanced modeling concepts more appealing and beneficial.

With *TravelPlaner*, we show that using complex and hard-to-implement modeling concepts may help to achieve compact and comprehensive models. Our numbers — achieved by implementing these models — indicate that large parts of *TravelPlaner* can be generated, even though behavior is only modeled to a certain extent, and implementations of actions are counted to the lines of hand-written (and therefore not generated) code.

Admitting that these numbers might be too optimistic, we discussed overestimation. But even with some adjustments, the resulting numbers are still very impressive.

The raise in level of abstraction — which is one of the greatest benefits of modeling languages — decreases if concepts of higher semantic levels are not employed in models. Therefore, our ambition is to include advanced concepts of structural and behavioral modeling in code generation with full consideration of their complex semantics. This case study clearly shows that at least for some kinds of projects this approach is feasible, and that models which exactly describe the system can contribute to a high proportion of code generation.

REFERENCES

[1] D. Akehurst, G. Howells, and K. McDonald-Maier. Implementing associations: Uml 2.0 to java 5. *Software and Systems Modeling*, 6:3–35, 2007.

[2] A. Bhattacharjee and R. Shyamasundar. Validated code generation for activity diagrams. In G. Chakraborty, editor, *Distributed Computing and Internet Technology*, volume 3816 of *Lecture Notes in Computer Science*, pages 508–521. Springer Berlin / Heidelberg, 2005.

[3] M. Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 33–42, New York, NY, USA, 2006. ACM.

[4] M. L. Crane. *Slicing UML's Three-layer Architecture: A Semantic Foundation for Behavioural Specification*. PhD thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2009.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] Fujaba Associations Specification, 2011. http://www.se.eecs.uni-kassel.de/~fujabawiki/index.php/Fujaba_Associations_Specification

[7] Fujaba Development Group. Fujaba Tool Suite 4.3.2, 2007.
http://www.fujaba.de/

[8] G. Génova, J. Llorens, and C. R. del Castillo. Mapping UML Associations into Java Code. *Journal of Object Technology*, 2(5):135–162, 2003.

[9] D. Gessenharter. Mapping the UML2 Semantics of Associations to a Java Code Generation Model. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 813–827, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] D. Gessenharter. Implementing UML Associations in Java: A Slim Code Pattern for a Complex Modeling Concept. In *Proceedings of the Workshop on Relationships and Associations in Object-Oriented Languages*, RAOOL '09, pages 17–24, New York, NY, USA, 2009. ACM.

[11] D. Gessenharter. Extending the UML Semantics for a Better Support of Model Driven Software Development. In *Software Engineering Research and Practice*, SERP '10, pages 45–51, 2010.

[12] D. Gessenharter. UML Activities at Runtime - Experiences of Using Interpreters and Running Generated Code. In *ER Workshops*, ER '10, pages 275–284, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] D. Gessenharter and M. Rauscher. . In *ECMFA'11: Proceedings of the 7th European Conference on Modelling Foundations and Applications*, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] D. Harel. From play-in scenarios to code: An achievable dream. *Computer*, 34:53–60, 2001.

[15] N. Koch and A. Kraus. The Expressive Power of UML-based Web Engineering, 2002.
http://www.pst.informatik.uni-muenchen.de/personen/kochn/IWWOST02-koch-kraus.PDF

[16] N. Koch, G. Zhang, and H. Baumeister. UML-Based Web Engineering: An Approach Based on Standards. *Web Engineering: Modelling and Implementing Web Applications*, pages 157–191, 2008. http://www.pst.ifi.lmu.de/veroeffentlichungen/uwe.pdf

[17] S. J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[18] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[19] Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML), v1.0, 2011. OMG Document Number: formal/2011-02-01.

[20] S. Sarstedt, J. Kohlmeyer, A. Raschke, and M. Schneiderhan. A New Approach to Combine Models and Code in Model Driven Development. In *Software Engineering Research and Practice*, SERP '05, pages 396–402, 2005.

[21] M. Usman and A. Nadeem. Automatic generation of java code from uml diagrams using ujector. *International Journal of Software Engineering and Its Applications*, 3(2):21–38, April 2009.

[22] Object Management Group, UML 2.2 Superstructure Specification, 2009. Document formal/2009-02-02

APPENDIX A
FULL-SIZE FIGURES

In this appendix, some figures that are difficult to read in the text of this report are given in full size and rotated, where needed.

This page is intentionally left blank.
Figures are contained on the following pages.

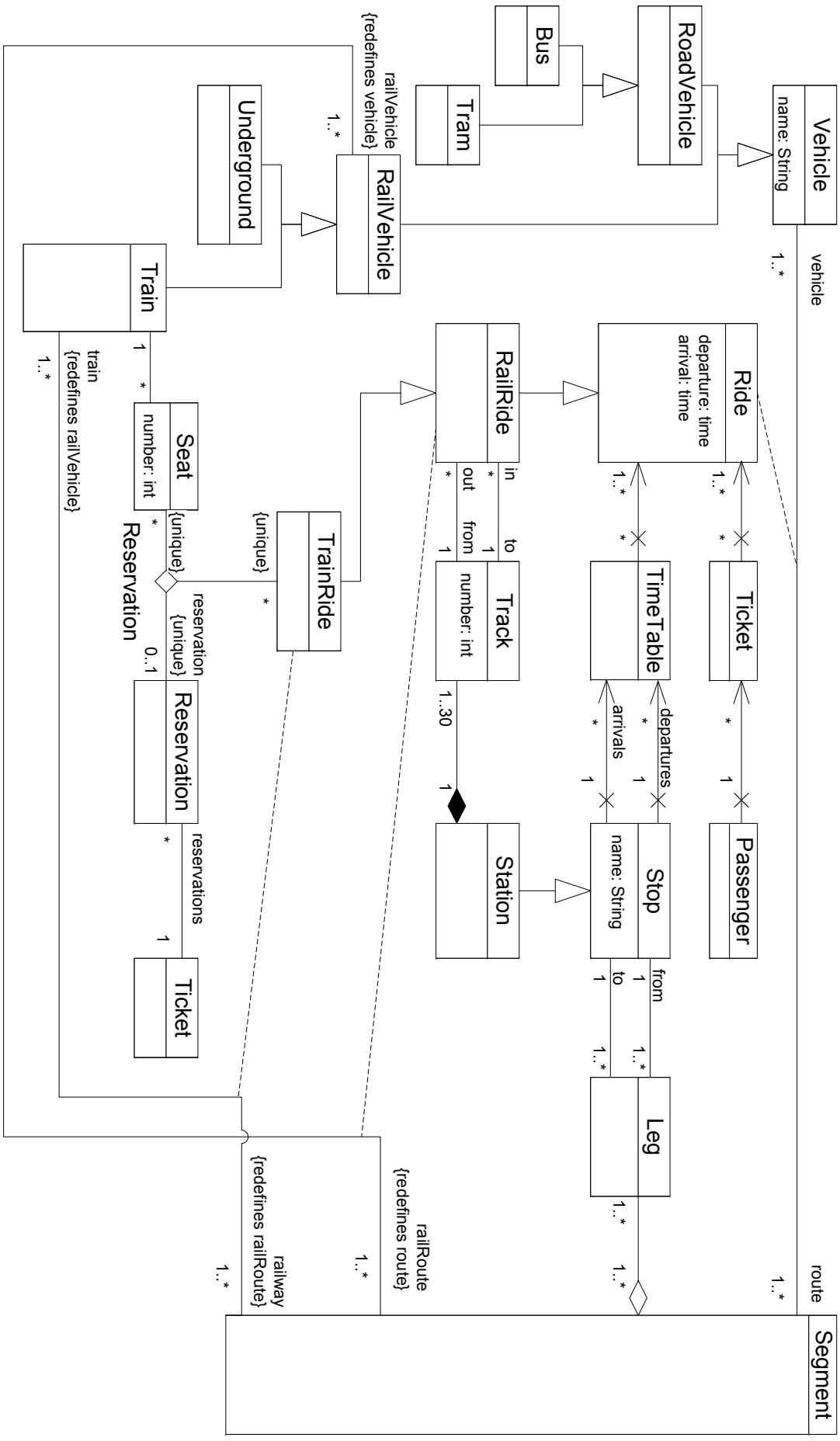


Figure 19. Complete static structure of the *TravelPlaner*.

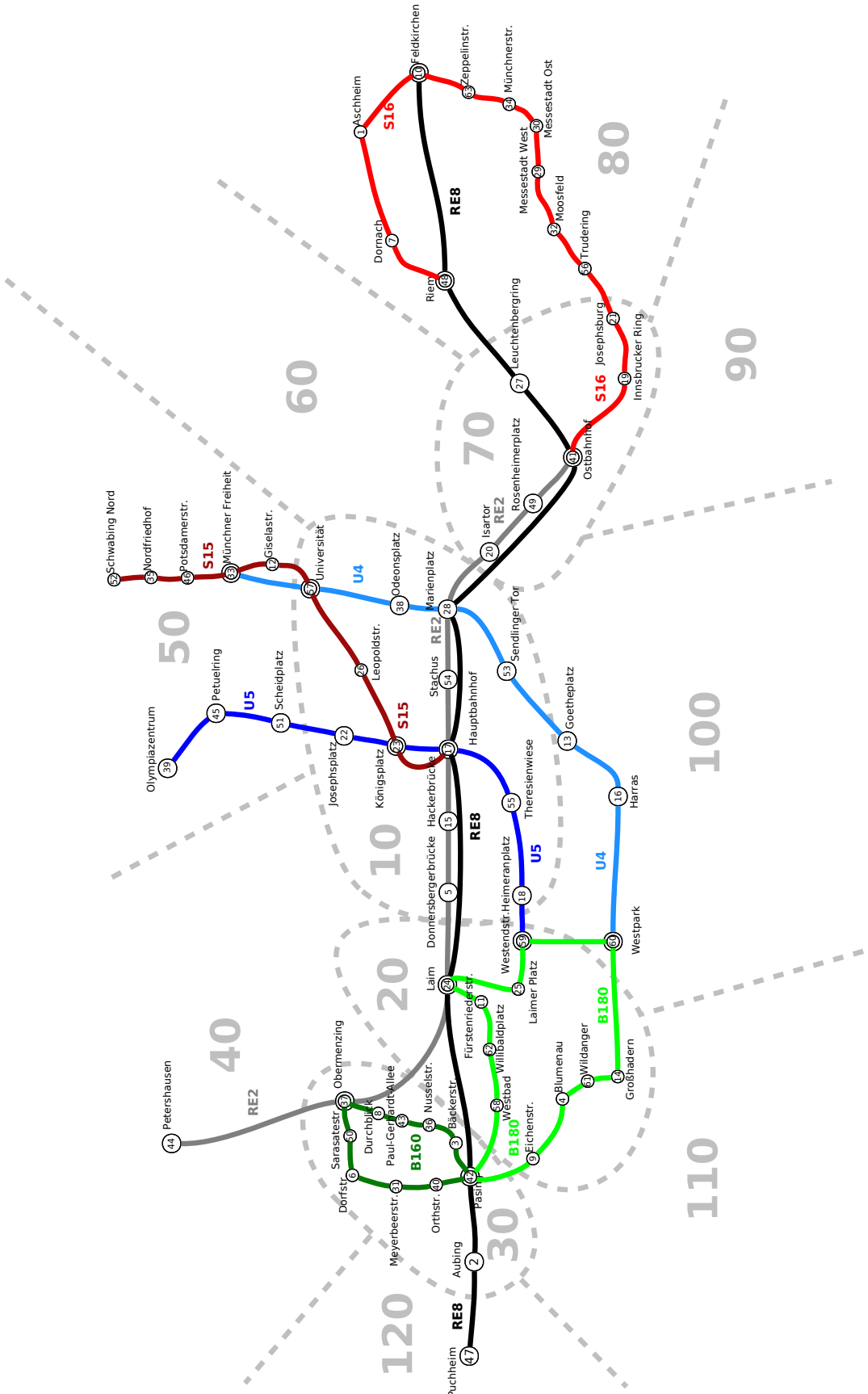


Figure 20. Concrete instance of a transportation network and its carriers

APPENDIX B
TIME TABLES

B.1 Busses

B160 — From Pasing to Pasing.

Pasing	Orthstr.	Meyerbeerstr.	Dorfstr.	Sarasatestr.	Obermenzing	Durchblick	Paul-Gerhardt-Allee	Nusselstr.	Bäckerstr.	Pasing
04:46	04:52	04:56	05:03	05:08	05:12	05:15	05:18	05:25	05:32	05:38
05:46	05:52	05:56	06:03	06:08	06:12	06:15	06:18	06:25	06:32	06:38
06:46	06:52	06:56	07:03	07:08	07:12	07:15	07:18	07:25	07:32	07:38
07:46	07:52	07:56	08:03	08:08	08:12	08:15	08:18	08:25	08:32	08:38
08:46	08:52	08:56	09:03	09:08	09:12	09:15	09:18	09:25	09:32	09:38
09:46	09:52	09:56	10:03	10:08	10:12	10:15	10:18	10:25	10:32	10:38
10:46	10:52	10:56	11:03	11:08	11:12	11:15	11:18	11:25	11:32	11:38
11:46	11:52	11:56	12:03	12:08	12:12	12:15	12:18	12:25	12:32	12:38
12:46	12:52	12:56	13:03	13:08	13:12	13:15	13:18	13:25	13:32	13:38
13:46	13:52	13:56	14:03	14:08	14:12	14:15	14:18	14:25	14:32	14:38
14:46	14:52	14:56	15:03	15:08	15:12	15:15	15:18	15:25	15:32	15:38
15:46	15:52	15:56	16:03	16:08	16:12	16:15	16:18	16:25	16:32	16:38
16:46	16:52	16:56	17:03	17:08	17:12	17:15	17:18	17:25	17:32	17:38
17:46	17:52	17:56	18:03	18:08	18:12	18:15	18:18	18:25	18:32	18:38
18:46	18:52	18:56	19:03	19:08	19:12	19:15	19:18	19:25	19:32	19:38
19:46	19:52	19:56	20:03	20:08	20:12	20:15	20:18	20:25	20:32	20:38
20:46	20:52	20:56	21:03	21:08	21:12	21:15	21:18	21:25	21:32	21:38
21:46	21:52	21:56	22:03	22:08	22:12	22:15	22:18	22:25	22:32	22:38
22:46	22:52	22:56	23:03	23:08	23:12	23:15	23:18	23:25	23:32	23:38
23:46	23:52	23:56	00:03	00:08	00:12	00:15	00:18	00:25	00:32	00:38

B180 — From Westpark to Westpark.

Westpark	Großhadern	Wildanger	Blumenau	Eichenstr.	Pasing	Westbad	Willibaldplatz	Fürstenriederstr.	Laim	Laimer Platz	Westendstr.	Westpark
04:47	04:59	05:02	05:05	05:10	05:14	05:19	05:23	05:26	05:28	05:31	05:33	05:40
05:17	05:29	05:32	05:35	05:40	05:44	05:49	05:53	05:56	05:58	06:01	06:03	06:10
05:47	05:59	06:02	06:05	06:10	06:14	06:19	06:23	06:26	06:28	06:31	06:33	06:40
06:17	06:29	06:32	06:35	06:40	06:44	06:49	06:53	06:56	06:58	07:01	07:03	07:10
06:47	06:59	07:02	07:05	07:10	07:14	07:19	07:23	07:26	07:28	07:31	07:33	07:40
07:17	07:29	07:32	07:35	07:40	07:44	07:49	07:53	07:56	07:58	08:01	08:03	08:10
07:47	07:59	08:02	08:05	08:10	08:14	08:19	08:23	08:26	08:28	08:31	08:33	08:40
08:17	08:29	08:32	08:35	08:40	08:44	08:49	08:53	08:56	08:58	09:01	09:03	09:10
08:47	08:59	09:02	09:05	09:10	09:14	09:19	09:23	09:26	09:28	09:31	09:33	09:40
09:17	09:29	09:32	09:35	09:40	09:44	09:49	09:53	09:56	09:58	10:01	10:03	10:10
09:47	09:59	10:02	10:05	10:10	10:14	10:19	10:23	10:26	10:28	10:31	10:33	10:40
10:17	10:29	10:32	10:35	10:40	10:44	10:49	10:53	10:56	10:58	11:01	11:03	11:10
10:47	10:59	11:02	11:05	11:10	11:14	11:19	11:23	11:26	11:28	11:31	11:33	11:40
11:17	11:29	11:32	11:35	11:40	11:44	11:49	11:53	11:56	11:58	12:01	12:03	12:10
11:47	11:59	12:02	12:05	12:10	12:14	12:19	12:23	12:26	12:28	12:31	12:33	12:40
12:17	12:29	12:32	12:35	12:40	12:44	12:49	12:53	12:56	12:58	13:01	13:03	13:10
12:47	12:59	13:02	13:05	13:10	13:14	13:19	13:23	13:26	13:28	13:31	13:33	13:40
13:17	13:29	13:32	13:35	13:40	13:44	13:49	13:53	13:56	13:58	14:01	14:03	14:10
13:47	13:59	14:02	14:05	14:10	14:14	14:19	14:23	14:26	14:28	14:31	14:33	14:40
14:17	14:29	14:32	14:35	14:40	14:44	14:49	14:53	14:56	14:58	15:01	15:03	15:10
14:47	14:59	15:02	15:05	15:10	15:14	15:19	15:23	15:26	15:28	15:31	15:33	15:40
15:17	15:29	15:32	15:35	15:40	15:44	15:49	15:53	15:56	15:58	16:01	16:03	16:10
15:47	15:59	16:02	16:05	16:10	16:14	16:19	16:23	16:26	16:28	16:31	16:33	16:40
16:17	16:29	16:32	16:35	16:40	16:44	16:49	16:53	16:56	16:58	17:01	17:03	17:10
16:47	16:59	17:02	17:05	17:10	17:14	17:19	17:23	17:26	17:28	17:31	17:33	17:40
17:17	17:29	17:32	17:35	17:40	17:44	17:49	17:53	17:56	17:58	18:01	18:03	18:10
17:47	17:59	18:02	18:05	18:10	18:14	18:19	18:23	18:26	18:28	18:31	18:33	18:40
18:17	18:29	18:32	18:35	18:40	18:44	18:49	18:53	18:56	18:58	19:01	19:03	19:10
18:47	18:59	19:02	19:05	19:10	19:14	19:19	19:23	19:26	19:28	19:31	19:33	19:40
19:17	19:29	19:32	19:35	19:40	19:44	19:49	19:53	19:56	19:58	20:01	20:03	20:10
19:47	19:59	20:02	20:05	20:10	20:14	20:19	20:23	20:26	20:28	20:31	20:33	20:40
20:17	20:29	20:32	20:35	20:40	20:44	20:49	20:53	20:56	20:58	21:01	21:03	21:10
20:47	20:59	21:02	21:05	21:10	21:14	21:19	21:23	21:26	21:28	21:31	21:33	21:40
21:17	21:29	21:32	21:35	21:40	21:44	21:49	21:53	21:56	21:58	22:01	22:03	22:10
21:47	21:59	22:02	22:05	22:10	22:14	22:19	22:23	22:26	22:28	22:31	22:33	22:40
22:17	22:29	22:32	22:35	22:40	22:44	22:49	22:53	22:56	22:58	23:01	23:03	23:10
22:47	22:59	23:02	23:05	23:10	23:14	23:19	23:23	23:26	23:28	23:31	23:33	23:40
23:17	23:29	23:32	23:35	23:40	23:44	23:49	23:53	23:56	23:58	00:01	00:03	00:10

B.2 Trams

S15 — Hauptbahnhof → Schwabing Nord

Hauptbahnhof	Königsplatz	Leopoldstr.	Universität	Giselastr.	Münchner Freiheit	Potsdamerstr.	Nordfriedhof	Schwabing Nord
04:26	04:28	04:31	04:33	04:35	04:37	04:39	04:41	04:43
04:46	04:48	04:51	04:53	04:55	04:57	04:59	05:01	05:03
05:06	05:08	05:11	05:13	05:15	05:17	05:19	05:21	05:23
05:26	05:28	05:31	05:33	05:35	05:37	05:39	05:41	05:43
05:46	05:48	05:51	05:53	05:55	05:57	05:59	06:01	06:03
05:58	06:00	06:03	06:05	06:07	06:09	06:11	06:13	06:15
06:10	06:12	06:15	06:17	06:19	06:21	06:23	06:25	06:27
06:22	06:24	06:27	06:29	06:31	06:33	06:35	06:37	06:39
06:34	06:36	06:39	06:41	06:43	06:45	06:47	06:49	06:51
06:46	06:48	06:51	06:53	06:55	06:57	06:59	07:01	07:03
06:58	07:00	07:03	07:05	07:07	07:09	07:11	07:13	07:15
07:10	07:12	07:15	07:17	07:19	07:21	07:23	07:25	07:27
07:22	07:24	07:27	07:29	07:31	07:33	07:35	07:37	07:39
07:34	07:36	07:39	07:41	07:43	07:45	07:47	07:49	07:51
07:46	07:48	07:51	07:53	07:55	07:57	07:59	08:01	08:03
07:58	08:00	08:03	08:05	08:07	08:09	08:11	08:13	08:15
08:10	08:12	08:15	08:17	08:19	08:21	08:23	08:25	08:27
08:22	08:24	08:27	08:29	08:31	08:33	08:35	08:37	08:39
08:34	08:36	08:39	08:41	08:43	08:45	08:47	08:49	08:51
08:46	08:48	08:51	08:53	08:55	08:57	08:59	09:01	09:03
08:58	09:00	09:03	09:05	09:07	09:09	09:11	09:13	09:15
09:10	09:12	09:15	09:17	09:19	09:21	09:23	09:25	09:27
09:22	09:24	09:27	09:29	09:31	09:33	09:35	09:37	09:39
09:34	09:36	09:39	09:41	09:43	09:45	09:47	09:49	09:51
09:46	09:48	09:51	09:53	09:55	09:57	09:59	10:01	10:03
09:58	10:00	10:03	10:05	10:07	10:09	10:11	10:13	10:15
10:10	10:12	10:15	10:17	10:19	10:21	10:23	10:25	10:27
10:22	10:24	10:27	10:29	10:31	10:33	10:35	10:37	10:39
10:34	10:36	10:39	10:41	10:43	10:45	10:47	10:49	10:51
10:46	10:48	10:51	10:53	10:55	10:57	10:59	11:01	11:03
10:58	11:00	11:03	11:05	11:07	11:09	11:11	11:13	11:15
11:10	11:12	11:15	11:17	11:19	11:21	11:23	11:25	11:27
11:22	11:24	11:27	11:29	11:31	11:33	11:35	11:37	11:39
11:34	11:36	11:39	11:41	11:43	11:45	11:47	11:49	11:51
11:46	11:48	11:51	11:53	11:55	11:57	11:59	12:01	12:03
11:58	12:00	12:03	12:05	12:07	12:09	12:11	12:13	12:15
12:10	12:12	12:15	12:17	12:19	12:21	12:23	12:25	12:27
12:22	12:24	12:27	12:29	12:31	12:33	12:35	12:37	12:39
12:34	12:36	12:39	12:41	12:43	12:45	12:47	12:49	12:51
12:46	12:48	12:51	12:53	12:55	12:57	12:59	13:01	13:03
12:58	13:00	13:03	13:05	13:07	13:09	13:11	13:13	13:15
13:10	13:12	13:15	13:17	13:19	13:21	13:23	13:25	13:27
13:22	13:24	13:27	13:29	13:31	13:33	13:35	13:37	13:39
13:34	13:36	13:39	13:41	13:43	13:45	13:47	13:49	13:51
13:46	13:48	13:51	13:53	13:55	13:57	13:59	14:01	14:03
13:58	14:00	14:03	14:05	14:07	14:09	14:11	14:13	14:15
14:10	14:12	14:15	14:17	14:19	14:21	14:23	14:25	14:27
14:22	14:24	14:27	14:29	14:31	14:33	14:35	14:37	14:39
14:34	14:36	14:39	14:41	14:43	14:45	14:47	14:49	14:51
14:46	14:48	14:51	14:53	14:55	14:57	14:59	15:01	15:03

Hauptbahnhof	Königsplatz	Leopoldstr.	Universität	Giselastr.	Münchner Freiheit	Potsdamerstr.	Nordfriedhof	Schwabing Nord
14:58	15:00	15:03	15:05	15:07	15:09	15:11	15:13	15:15
15:10	15:12	15:15	15:17	15:19	15:21	15:23	15:25	15:27
15:22	15:24	15:27	15:29	15:31	15:33	15:35	15:37	15:39
15:34	15:36	15:39	15:41	15:43	15:45	15:47	15:49	15:51
15:46	15:48	15:51	15:53	15:55	15:57	15:59	16:01	16:03
15:58	16:00	16:03	16:05	16:07	16:09	16:11	16:13	16:15
16:10	16:12	16:15	16:17	16:19	16:21	16:23	16:25	16:27
16:22	16:24	16:27	16:29	16:31	16:33	16:35	16:37	16:39
16:34	16:36	16:39	16:41	16:43	16:45	16:47	16:49	16:51
16:46	16:48	16:51	16:53	16:55	16:57	16:59	17:01	17:03
16:58	17:00	17:03	17:05	17:07	17:09	17:11	17:13	17:15
17:10	17:12	17:15	17:17	17:19	17:21	17:23	17:25	17:27
17:22	17:24	17:27	17:29	17:31	17:33	17:35	17:37	17:39
17:34	17:36	17:39	17:41	17:43	17:45	17:47	17:49	17:51
17:46	17:48	17:51	17:53	17:55	17:57	17:59	18:01	18:03
17:58	18:00	18:03	18:05	18:07	18:09	18:11	18:13	18:15
18:10	18:12	18:15	18:17	18:19	18:21	18:23	18:25	18:27
18:22	18:24	18:27	18:29	18:31	18:33	18:35	18:37	18:39
18:34	18:36	18:39	18:41	18:43	18:45	18:47	18:49	18:51
18:46	18:48	18:51	18:53	18:55	18:57	18:59	19:01	19:03
18:58	19:00	19:03	19:05	19:07	19:09	19:11	19:13	19:15
19:10	19:12	19:15	19:17	19:19	19:21	19:23	19:25	19:27
19:22	19:24	19:27	19:29	19:31	19:33	19:35	19:37	19:39
19:34	19:36	19:39	19:41	19:43	19:45	19:47	19:49	19:51
19:46	19:48	19:51	19:53	19:55	19:57	19:59	20:01	20:03
20:06	20:08	20:11	20:13	20:15	20:17	20:19	20:21	20:23
20:26	20:28	20:31	20:33	20:35	20:37	20:39	20:41	20:43
20:46	20:48	20:51	20:53	20:55	20:57	20:59	21:01	21:03
21:06	21:08	21:11	21:13	21:15	21:17	21:19	21:21	21:23
21:26	21:28	21:31	21:33	21:35	21:37	21:39	21:41	21:43
21:46	21:48	21:51	21:53	21:55	21:57	21:59	22:01	22:03
22:06	22:08	22:11	22:13	22:15	22:17	22:19	22:21	22:23
22:26	22:28	22:31	22:33	22:35	22:37	22:39	22:41	22:43
22:46	22:48	22:51	22:53	22:55	22:57	22:59	23:01	23:03
23:06	23:08	23:11	23:13	23:15	23:17	23:19	23:21	23:23
23:26	23:28	23:31	23:33	23:35	23:37	23:39	23:41	23:43
23:46	23:48	23:51	23:53	23:55	23:57	23:59	00:01	00:03

S15 — Schwabing Nord → Hauptbahnhof

Schwabing Nord	Nordfriedhof	Potsdamerstr.	Münchner Freiheit	Giselastr.	Universität	Leopoldstr.	Königsplatz	Hauptbahnhof
04:46	04:48	04:50	04:52	04:54	04:56	04:58	05:01	05:03
05:06	05:08	05:10	05:12	05:14	05:16	05:18	05:21	05:23
05:26	05:28	05:30	05:32	05:34	05:36	05:38	05:41	05:43
05:46	05:48	05:50	05:52	05:54	05:56	05:58	06:01	06:03
05:58	06:00	06:02	06:04	06:06	06:08	06:10	06:13	06:15
06:10	06:12	06:14	06:16	06:18	06:20	06:22	06:25	06:27
06:22	06:24	06:26	06:28	06:30	06:32	06:34	06:37	06:39
06:34	06:36	06:38	06:40	06:42	06:44	06:46	06:49	06:51
06:46	06:48	06:50	06:52	06:54	06:56	06:58	07:01	07:03
06:58	07:00	07:02	07:04	07:06	07:08	07:10	07:13	07:15
07:10	07:12	07:14	07:16	07:18	07:20	07:22	07:25	07:27
07:22	07:24	07:26	07:28	07:30	07:32	07:34	07:37	07:39
07:34	07:36	07:38	07:40	07:42	07:44	07:46	07:49	07:51
07:46	07:48	07:50	07:52	07:54	07:56	07:58	08:01	08:03
07:58	08:00	08:02	08:04	08:06	08:08	08:10	08:13	08:15
08:10	08:12	08:14	08:16	08:18	08:20	08:22	08:25	08:27
08:22	08:24	08:26	08:28	08:30	08:32	08:34	08:37	08:39
08:34	08:36	08:38	08:40	08:42	08:44	08:46	08:49	08:51
08:46	08:48	08:50	08:52	08:54	08:56	08:58	09:01	09:03
08:58	09:00	09:02	09:04	09:06	09:08	09:10	09:13	09:15
09:10	09:12	09:14	09:16	09:18	09:20	09:22	09:25	09:27
09:22	09:24	09:26	09:28	09:30	09:32	09:34	09:37	09:39
09:34	09:36	09:38	09:40	09:42	09:44	09:46	09:49	09:51
09:46	09:48	09:50	09:52	09:54	09:56	09:58	10:01	10:03
09:58	10:00	10:02	10:04	10:06	10:08	10:10	10:13	10:15
10:10	10:12	10:14	10:16	10:18	10:20	10:22	10:25	10:27
10:22	10:24	10:26	10:28	10:30	10:32	10:34	10:37	10:39
10:34	10:36	10:38	10:40	10:42	10:44	10:46	10:49	10:51
10:46	10:48	10:50	10:52	10:54	10:56	10:58	11:01	11:03
10:58	11:00	11:02	11:04	11:06	11:08	11:10	11:13	11:15
11:10	11:12	11:14	11:16	11:18	11:20	11:22	11:25	11:27
11:22	11:24	11:26	11:28	11:30	11:32	11:34	11:37	11:39
11:34	11:36	11:38	11:40	11:42	11:44	11:46	11:49	11:51
11:46	11:48	11:50	11:52	11:54	11:56	11:58	12:01	12:03
11:58	12:00	12:02	12:04	12:06	12:08	12:10	12:13	12:15
12:10	12:12	12:14	12:16	12:18	12:20	12:22	12:25	12:27
12:22	12:24	12:26	12:28	12:30	12:32	12:34	12:37	12:39
12:34	12:36	12:38	12:40	12:42	12:44	12:46	12:49	12:51
12:46	12:48	12:50	12:52	12:54	12:56	12:58	13:01	13:03
12:58	13:00	13:02	13:04	13:06	13:08	13:10	13:13	13:15
13:10	13:12	13:14	13:16	13:18	13:20	13:22	13:25	13:27
13:22	13:24	13:26	13:28	13:30	13:32	13:34	13:37	13:39
13:34	13:36	13:38	13:40	13:42	13:44	13:46	13:49	13:51
13:46	13:48	13:50	13:52	13:54	13:56	13:58	14:01	14:03
13:58	14:00	14:02	14:04	14:06	14:08	14:10	14:13	14:15
14:10	14:12	14:14	14:16	14:18	14:20	14:22	14:25	14:27
14:22	14:24	14:26	14:28	14:30	14:32	14:34	14:37	14:39
14:34	14:36	14:38	14:40	14:42	14:44	14:46	14:49	14:51
14:46	14:48	14:50	14:52	14:54	14:56	14:58	15:01	15:03
14:58	15:00	15:02	15:04	15:06	15:08	15:10	15:13	15:15
15:10	15:12	15:14	15:16	15:18	15:20	15:22	15:25	15:27
15:22	15:24	15:26	15:28	15:30	15:32	15:34	15:37	15:39

Schwabing Nord	Nordfriedhof	Potsdamerstr.	Münchner Freiheit	Giselastr.	Universität	Leopoldstr.	Königsplatz	Hauptbahnhof
15:34	15:36	15:38	15:40	15:42	15:44	15:46	15:49	15:51
15:46	15:48	15:50	15:52	15:54	15:56	15:58	16:01	16:03
15:58	16:00	16:02	16:04	16:06	16:08	16:10	16:13	16:15
16:10	16:12	16:14	16:16	16:18	16:20	16:22	16:25	16:27
16:22	16:24	16:26	16:28	16:30	16:32	16:34	16:37	16:39
16:34	16:36	16:38	16:40	16:42	16:44	16:46	16:49	16:51
16:46	16:48	16:50	16:52	16:54	16:56	16:58	17:01	17:03
16:58	17:00	17:02	17:04	17:06	17:08	17:10	17:13	17:15
17:10	17:12	17:14	17:16	17:18	17:20	17:22	17:25	17:27
17:22	17:24	17:26	17:28	17:30	17:32	17:34	17:37	17:39
17:34	17:36	17:38	17:40	17:42	17:44	17:46	17:49	17:51
17:46	17:48	17:50	17:52	17:54	17:56	17:58	18:01	18:03
17:58	18:00	18:02	18:04	18:06	18:08	18:10	18:13	18:15
18:10	18:12	18:14	18:16	18:18	18:20	18:22	18:25	18:27
18:22	18:24	18:26	18:28	18:30	18:32	18:34	18:37	18:39
18:34	18:36	18:38	18:40	18:42	18:44	18:46	18:49	18:51
18:46	18:48	18:50	18:52	18:54	18:56	18:58	19:01	19:03
18:58	19:00	19:02	19:04	19:06	19:08	19:10	19:13	19:15
19:10	19:12	19:14	19:16	19:18	19:20	19:22	19:25	19:27
19:22	19:24	19:26	19:28	19:30	19:32	19:34	19:37	19:39
19:34	19:36	19:38	19:40	19:42	19:44	19:46	19:49	19:51
19:46	19:48	19:50	19:52	19:54	19:56	19:58	20:01	20:03
20:06	20:08	20:10	20:12	20:14	20:16	20:18	20:21	20:23
20:26	20:28	20:30	20:32	20:34	20:36	20:38	20:41	20:43
20:46	20:48	20:50	20:52	20:54	20:56	20:58	21:01	21:03
21:06	21:08	21:10	21:12	21:14	21:16	21:18	21:21	21:23
21:26	21:28	21:30	21:32	21:34	21:36	21:38	21:41	21:43
21:46	21:48	21:50	21:52	21:54	21:56	21:58	22:01	22:03
22:06	22:08	22:10	22:12	22:14	22:16	22:18	22:21	22:23
22:26	22:28	22:30	22:32	22:34	22:36	22:38	22:41	22:43
22:46	22:48	22:50	22:52	22:54	22:56	22:58	23:01	23:03
23:06	23:08	23:10	23:12	23:14	23:16	23:18	23:21	23:23
23:26	23:28	23:30	23:32	23:34	23:36	23:38	23:41	23:43
23:46	23:48	23:50	23:52	23:54	23:56	23:58	00:01	00:03

S16 — Riem → Ostbahnhof

Riem	Dornach	Aschheim	Feldkirchen	Zeppelinstr.	Münchner Str.	Messestadt West	Messestadt Ost	Moosfeld	Trudering	Josphsburg	Innsbrucker Ring	Ostbahnhof
04:01	04:07	04:14	04:21	04:26	04:31	04:35	04:36	04:38	04:41	04:44	04:49	04:56
04:31	04:37	04:44	04:51	04:56	05:01	05:05	05:06	05:08	05:11	05:14	05:19	05:26
05:01	05:07	05:14	05:21	05:26	05:31	05:35	05:36	05:38	05:41	05:44	05:49	05:56
05:31	05:37	05:44	05:51	05:56	06:01	06:05	06:06	06:08	06:11	06:14	06:19	06:26
06:01	06:07	06:14	06:21	06:26	06:31	06:35	06:36	06:38	06:41	06:44	06:49	06:56
06:31	06:37	06:44	06:51	06:56	07:01	07:05	07:06	07:08	07:11	07:14	07:19	07:26
07:01	07:07	07:14	07:21	07:26	07:31	07:35	07:36	07:38	07:41	07:44	07:49	07:56
07:31	07:37	07:44	07:51	07:56	08:01	08:05	08:06	08:08	08:11	08:14	08:19	08:26
08:01	08:07	08:14	08:21	08:26	08:31	08:35	08:36	08:38	08:41	08:44	08:49	08:56
08:31	08:37	08:44	08:51	08:56	09:01	09:05	09:06	09:08	09:11	09:14	09:19	09:26
09:01	09:07	09:14	09:21	09:26	09:31	09:35	09:36	09:38	09:41	09:44	09:49	09:56
09:31	09:37	09:44	09:51	09:56	10:01	10:05	10:06	10:08	10:11	10:14	10:19	10:26
10:01	10:07	10:14	10:21	10:26	10:31	10:35	10:36	10:38	10:41	10:44	10:49	10:56
10:31	10:37	10:44	10:51	10:56	11:01	11:05	11:06	11:08	11:11	11:14	11:19	11:26
11:01	11:07	11:14	11:21	11:26	11:31	11:35	11:36	11:38	11:41	11:44	11:49	11:56
11:31	11:37	11:44	11:51	11:56	12:01	12:05	12:06	12:08	12:11	12:14	12:19	12:26
12:01	12:07	12:14	12:21	12:26	12:31	12:35	12:36	12:38	12:41	12:44	12:49	12:56
12:31	12:37	12:44	12:51	12:56	13:01	13:05	13:06	13:08	13:11	13:14	13:19	13:26
13:01	13:07	13:14	13:21	13:26	13:31	13:35	13:36	13:38	13:41	13:44	13:49	13:56
13:31	13:37	13:44	13:51	13:56	14:01	14:05	14:06	14:08	14:11	14:14	14:19	14:26
14:01	14:07	14:14	14:21	14:26	14:31	14:35	14:36	14:38	14:41	14:44	14:49	14:56
14:31	14:37	14:44	14:51	14:56	15:01	15:05	15:06	15:08	15:11	15:14	15:19	15:26
15:01	15:07	15:14	15:21	15:26	15:31	15:35	15:36	15:38	15:41	15:44	15:49	15:56
15:31	15:37	15:44	15:51	15:56	16:01	16:05	16:06	16:08	16:11	16:14	16:19	16:26
16:01	16:07	16:14	16:21	16:26	16:31	16:35	16:36	16:38	16:41	16:44	16:49	16:56
16:31	16:37	16:44	16:51	16:56	17:01	17:05	17:06	17:08	17:11	17:14	17:19	17:26
17:01	17:07	17:14	17:21	17:26	17:31	17:35	17:36	17:38	17:41	17:44	17:49	17:56
17:31	17:37	17:44	17:51	17:56	18:01	18:05	18:06	18:08	18:11	18:14	18:19	18:26
18:01	18:07	18:14	18:21	18:26	18:31	18:35	18:36	18:38	18:41	18:44	18:49	18:56
18:31	18:37	18:44	18:51	18:56	19:01	19:05	19:06	19:08	19:11	19:14	19:19	19:26
19:01	19:07	19:14	19:21	19:26	19:31	19:35	19:36	19:38	19:41	19:44	19:49	19:56
19:31	19:37	19:44	19:51	19:56	20:01	20:05	20:06	20:08	20:11	20:14	20:19	20:26
20:01	20:07	20:14	20:21	20:26	20:31	20:35	20:36	20:38	20:41	20:44	20:49	20:56
20:31	20:37	20:44	20:51	20:56	21:01	21:05	21:06	21:08	21:11	21:14	21:19	21:26
21:01	21:07	21:14	21:21	21:26	21:31	21:35	21:36	21:38	21:41	21:44	21:49	21:56
21:31	21:37	21:44	21:51	21:56	22:01	22:05	22:06	22:08	22:11	22:14	22:19	22:26
22:01	22:07	22:14	22:21	22:26	22:31	22:35	22:36	22:38	22:41	22:44	22:49	22:56
22:31	22:37	22:44	22:51	22:56	23:01	23:05	23:06	23:08	23:11	23:14	23:19	23:26
23:01	23:07	23:14	23:21	23:26	23:31	23:35	23:36	23:38	23:41	23:44	23:49	23:56
23:31	23:37	23:44	23:51	23:56	00:01	00:05	00:06	00:08	00:11	00:14	00:19	00:26
00:01	00:07	00:14	00:21	00:26	00:31	00:35	00:36	00:38	00:41	00:44	00:49	00:56
00:31	00:37	00:44	00:51	00:56	01:01	01:05	01:06	01:08	01:11	01:14	01:19	01:26

S16 — Ostbahnhof → Riem

Ostbahnhof	Innsbrucker Ring	Josphsburg	Trudering	Moosfeld	Messestadt Ost	Messestadt West	Münchner Str.	Zeppelinstr.	Feldkirchen	Aschheim	Dornach	Riem
05:03	05:10	05:15	05:18	05:21	05:23	05:24	05:28	05:33	05:38	05:45	05:52	05:58
05:33	05:40	05:45	05:48	05:51	05:53	05:54	05:58	06:03	06:08	06:15	06:22	06:28
06:03	06:10	06:15	06:18	06:21	06:23	06:24	06:28	06:33	06:38	06:45	06:52	06:58
06:33	06:40	06:45	06:48	06:51	06:53	06:54	06:58	07:03	07:08	07:15	07:22	07:28
07:03	07:10	07:15	07:18	07:21	07:23	07:24	07:28	07:33	07:38	07:45	07:52	07:58
07:33	07:40	07:45	07:48	07:51	07:53	07:54	07:58	08:03	08:08	08:15	08:22	08:28
08:03	08:10	08:15	08:18	08:21	08:23	08:24	08:28	08:33	08:38	08:45	08:52	08:58
08:33	08:40	08:45	08:48	08:51	08:53	08:54	08:58	09:03	09:08	09:15	09:22	09:28
09:03	09:10	09:15	09:18	09:21	09:23	09:24	09:28	09:33	09:38	09:45	09:52	09:58
09:33	09:40	09:45	09:48	09:51	09:53	09:54	09:58	10:03	10:08	10:15	10:22	10:28
10:03	10:10	10:15	10:18	10:21	10:23	10:24	10:28	10:33	10:38	10:45	10:52	10:58
10:33	10:40	10:45	10:48	10:51	10:53	10:54	10:58	11:03	11:08	11:15	11:22	11:28
11:03	11:10	11:15	11:18	11:21	11:23	11:24	11:28	11:33	11:38	11:45	11:52	11:58
11:33	11:40	11:45	11:48	11:51	11:53	11:54	11:58	12:03	12:08	12:15	12:22	12:28
12:03	12:10	12:15	12:18	12:21	12:23	12:24	12:28	12:33	12:38	12:45	12:52	12:58
12:33	12:40	12:45	12:48	12:51	12:53	12:54	12:58	13:03	13:08	13:15	13:22	13:28
13:03	13:10	13:15	13:18	13:21	13:23	13:24	13:28	13:33	13:38	13:45	13:52	13:58
13:33	13:40	13:45	13:48	13:51	13:53	13:54	13:58	14:03	14:08	14:15	14:22	14:28
14:03	14:10	14:15	14:18	14:21	14:23	14:24	14:28	14:33	14:38	14:45	14:52	14:58
14:33	14:40	14:45	14:48	14:51	14:53	14:54	14:58	15:03	15:08	15:15	15:22	15:28
15:03	15:10	15:15	15:18	15:21	15:23	15:24	15:28	15:33	15:38	15:45	15:52	15:58
15:33	15:40	15:45	15:48	15:51	15:53	15:54	15:58	16:03	16:08	16:15	16:22	16:28
16:03	16:10	16:15	16:18	16:21	16:23	16:24	16:28	16:33	16:38	16:45	16:52	16:58
16:33	16:40	16:45	16:48	16:51	16:53	16:54	16:58	17:03	17:08	17:15	17:22	17:28
17:03	17:10	17:15	17:18	17:21	17:23	17:24	17:28	17:33	17:38	17:45	17:52	17:58
17:33	17:40	17:45	17:48	17:51	17:53	17:54	17:58	18:03	18:08	18:15	18:22	18:28
18:03	18:10	18:15	18:18	18:21	18:23	18:24	18:28	18:33	18:38	18:45	18:52	18:58
18:33	18:40	18:45	18:48	18:51	18:53	18:54	18:58	19:03	19:08	19:15	19:22	19:28
19:03	19:10	19:15	19:18	19:21	19:23	19:24	19:28	19:33	19:38	19:45	19:52	19:58
19:33	19:40	19:45	19:48	19:51	19:53	19:54	19:58	20:03	20:08	20:15	20:22	20:28
20:03	20:10	20:15	20:18	20:21	20:23	20:24	20:28	20:33	20:38	20:45	20:52	20:58
20:33	20:40	20:45	20:48	20:51	20:53	20:54	20:58	21:03	21:08	21:15	21:22	21:28
21:03	21:10	21:15	21:18	21:21	21:23	21:24	21:28	21:33	21:38	21:45	21:52	21:58
21:33	21:40	21:45	21:48	21:51	21:53	21:54	21:58	22:03	22:08	22:15	22:22	22:28
22:03	22:10	22:15	22:18	22:21	22:23	22:24	22:28	22:33	22:38	22:45	22:52	22:58
22:33	22:40	22:45	22:48	22:51	22:53	22:54	22:58	23:03	23:08	23:15	23:22	23:28
23:03	23:10	23:15	23:18	23:21	23:23	23:24	23:28	23:33	23:38	23:45	23:52	23:58
23:33	23:40	23:45	23:48	23:51	23:53	23:54	23:58	00:03	00:08	00:15	00:22	00:28
00:03	00:10	00:15	00:18	00:21	00:23	00:24	00:28	00:33	00:38	00:45	00:52	00:58

B.3 Undergrounds

U4 — Westpark → Münchner Freiheit

Westpark	Harras	Goetheplatz	Sendlinger Tor	Marienplatz	Odeonsplatz	Universität	Münchner Freiheit
04:24	04:26	04:27	04:28	04:29	04:30	04:32	04:34
04:34	04:36	04:37	04:38	04:39	04:40	04:42	04:44
04:44	04:46	04:47	04:48	04:49	04:50	04:52	04:54
04:54	04:56	04:57	04:58	04:59	05:00	05:02	05:04
05:04	05:06	05:07	05:08	05:09	05:10	05:12	05:14
05:14	05:16	05:17	05:18	05:19	05:20	05:22	05:24
05:24	05:26	05:27	05:28	05:29	05:30	05:32	05:34
05:34	05:36	05:37	05:38	05:39	05:40	05:42	05:44
05:44	05:46	05:47	05:48	05:49	05:50	05:52	05:54
05:54	05:56	05:57	05:58	05:59	06:00	06:02	06:04
06:00	06:02	06:03	06:04	06:05	06:06	06:08	06:10
06:06	06:08	06:09	06:10	06:11	06:12	06:14	06:16
06:12	06:14	06:15	06:16	06:17	06:18	06:20	06:22
06:18	06:20	06:21	06:22	06:23	06:24	06:26	06:28
06:24	06:26	06:27	06:28	06:29	06:30	06:32	06:34
06:30	06:32	06:33	06:34	06:35	06:36	06:38	06:40
06:36	06:38	06:39	06:40	06:41	06:42	06:44	06:46
06:42	06:44	06:45	06:46	06:47	06:48	06:50	06:52
06:48	06:50	06:51	06:52	06:53	06:54	06:56	06:58
06:54	06:56	06:57	06:58	06:59	07:00	07:02	07:04
07:00	07:02	07:03	07:04	07:05	07:06	07:08	07:10
07:06	07:08	07:09	07:10	07:11	07:12	07:14	07:16
07:12	07:14	07:15	07:16	07:17	07:18	07:20	07:22
07:18	07:20	07:21	07:22	07:23	07:24	07:26	07:28
07:24	07:26	07:27	07:28	07:29	07:30	07:32	07:34
07:30	07:32	07:33	07:34	07:35	07:36	07:38	07:40
07:36	07:38	07:39	07:40	07:41	07:42	07:44	07:46
07:42	07:44	07:45	07:46	07:47	07:48	07:50	07:52
07:48	07:50	07:51	07:52	07:53	07:54	07:56	07:58
07:54	07:56	07:57	07:58	07:59	08:00	08:02	08:04
08:00	08:02	08:03	08:04	08:05	08:06	08:08	08:10
08:06	08:08	08:09	08:10	08:11	08:12	08:14	08:16
08:12	08:14	08:15	08:16	08:17	08:18	08:20	08:22
08:18	08:20	08:21	08:22	08:23	08:24	08:26	08:28
08:24	08:26	08:27	08:28	08:29	08:30	08:32	08:34
08:30	08:32	08:33	08:34	08:35	08:36	08:38	08:40
08:36	08:38	08:39	08:40	08:41	08:42	08:44	08:46
08:42	08:44	08:45	08:46	08:47	08:48	08:50	08:52
08:48	08:50	08:51	08:52	08:53	08:54	08:56	08:58
08:54	08:56	08:57	08:58	08:59	09:00	09:02	09:04
09:00	09:02	09:03	09:04	09:05	09:06	09:08	09:10
09:06	09:08	09:09	09:10	09:11	09:12	09:14	09:16
09:12	09:14	09:15	09:16	09:17	09:18	09:20	09:22
09:18	09:20	09:21	09:22	09:23	09:24	09:26	09:28
09:24	09:26	09:27	09:28	09:29	09:30	09:32	09:34
09:30	09:32	09:33	09:34	09:35	09:36	09:38	09:40
09:36	09:38	09:39	09:40	09:41	09:42	09:44	09:46
09:42	09:44	09:45	09:46	09:47	09:48	09:50	09:52
09:48	09:50	09:51	09:52	09:53	09:54	09:56	09:58
09:54	09:56	09:57	09:58	09:59	10:00	10:02	10:04

Westpark	Harras	Goetheplatz	Sendlinger Tor	Marienplatz	Odeonsplatz	Universität	M¨nchner Freiheit
10:00	10:02	10:03	10:04	10:05	10:06	10:08	10:10
10:06	10:08	10:09	10:10	10:11	10:12	10:14	10:16
10:12	10:14	10:15	10:16	10:17	10:18	10:20	10:22
10:18	10:20	10:21	10:22	10:23	10:24	10:26	10:28
10:24	10:26	10:27	10:28	10:29	10:30	10:32	10:34
10:30	10:32	10:33	10:34	10:35	10:36	10:38	10:40
10:36	10:38	10:39	10:40	10:41	10:42	10:44	10:46
10:42	10:44	10:45	10:46	10:47	10:48	10:50	10:52
10:48	10:50	10:51	10:52	10:53	10:54	10:56	10:58
10:54	10:56	10:57	10:58	10:59	11:00	11:02	11:04
11:00	11:02	11:03	11:04	11:05	11:06	11:08	11:10
11:06	11:08	11:09	11:10	11:11	11:12	11:14	11:16
11:12	11:14	11:15	11:16	11:17	11:18	11:20	11:22
11:18	11:20	11:21	11:22	11:23	11:24	11:26	11:28
11:24	11:26	11:27	11:28	11:29	11:30	11:32	11:34
11:30	11:32	11:33	11:34	11:35	11:36	11:38	11:40
11:36	11:38	11:39	11:40	11:41	11:42	11:44	11:46
11:42	11:44	11:45	11:46	11:47	11:48	11:50	11:52
11:48	11:50	11:51	11:52	11:53	11:54	11:56	11:58
11:54	11:56	11:57	11:58	11:59	12:00	12:02	12:04
12:00	12:02	12:03	12:04	12:05	12:06	12:08	12:10
12:06	12:08	12:09	12:10	12:11	12:12	12:14	12:16
12:12	12:14	12:15	12:16	12:17	12:18	12:20	12:22
12:18	12:20	12:21	12:22	12:23	12:24	12:26	12:28
12:24	12:26	12:27	12:28	12:29	12:30	12:32	12:34
12:30	12:32	12:33	12:34	12:35	12:36	12:38	12:40
12:36	12:38	12:39	12:40	12:41	12:42	12:44	12:46
12:42	12:44	12:45	12:46	12:47	12:48	12:50	12:52
12:48	12:50	12:51	12:52	12:53	12:54	12:56	12:58
12:54	12:56	12:57	12:58	12:59	13:00	13:02	13:04
13:00	13:02	13:03	13:04	13:05	13:06	13:08	13:10
13:06	13:08	13:09	13:10	13:11	13:12	13:14	13:16
13:12	13:14	13:15	13:16	13:17	13:18	13:20	13:22
13:18	13:20	13:21	13:22	13:23	13:24	13:26	13:28
13:24	13:26	13:27	13:28	13:29	13:30	13:32	13:34
13:30	13:32	13:33	13:34	13:35	13:36	13:38	13:40
13:36	13:38	13:39	13:40	13:41	13:42	13:44	13:46
13:42	13:44	13:45	13:46	13:47	13:48	13:50	13:52
13:48	13:50	13:51	13:52	13:53	13:54	13:56	13:58
13:54	13:56	13:57	13:58	13:59	14:00	14:02	14:04
14:00	14:02	14:03	14:04	14:05	14:06	14:08	14:10
14:06	14:08	14:09	14:10	14:11	14:12	14:14	14:16
14:12	14:14	14:15	14:16	14:17	14:18	14:20	14:22
14:18	14:20	14:21	14:22	14:23	14:24	14:26	14:28
14:24	14:26	14:27	14:28	14:29	14:30	14:32	14:34
14:30	14:32	14:33	14:34	14:35	14:36	14:38	14:40
14:36	14:38	14:39	14:40	14:41	14:42	14:44	14:46
14:42	14:44	14:45	14:46	14:47	14:48	14:50	14:52
14:48	14:50	14:51	14:52	14:53	14:54	14:56	14:58
14:54	14:56	14:57	14:58	14:59	15:00	15:02	15:04
15:00	15:02	15:03	15:04	15:05	15:06	15:08	15:10
15:06	15:08	15:09	15:10	15:11	15:12	15:14	15:16
15:12	15:14	15:15	15:16	15:17	15:18	15:20	15:22
15:18	15:20	15:21	15:22	15:23	15:24	15:26	15:28

Westpark	Harras	Goetheplatz	Sendlinger Tor	Marienplatz	Odeonsplatz	Universität	M¨nchner Freiheit
15:24	15:26	15:27	15:28	15:29	15:30	15:32	15:34
15:30	15:32	15:33	15:34	15:35	15:36	15:38	15:40
15:36	15:38	15:39	15:40	15:41	15:42	15:44	15:46
15:42	15:44	15:45	15:46	15:47	15:48	15:50	15:52
15:48	15:50	15:51	15:52	15:53	15:54	15:56	15:58
15:54	15:56	15:57	15:58	15:59	16:00	16:02	16:04
16:00	16:02	16:03	16:04	16:05	16:06	16:08	16:10
16:06	16:08	16:09	16:10	16:11	16:12	16:14	16:16
16:12	16:14	16:15	16:16	16:17	16:18	16:20	16:22
16:18	16:20	16:21	16:22	16:23	16:24	16:26	16:28
16:24	16:26	16:27	16:28	16:29	16:30	16:32	16:34
16:30	16:32	16:33	16:34	16:35	16:36	16:38	16:40
16:36	16:38	16:39	16:40	16:41	16:42	16:44	16:46
16:42	16:44	16:45	16:46	16:47	16:48	16:50	16:52
16:48	16:50	16:51	16:52	16:53	16:54	16:56	16:58
16:54	16:56	16:57	16:58	16:59	17:00	17:02	17:04
17:00	17:02	17:03	17:04	17:05	17:06	17:08	17:10
17:06	17:08	17:09	17:10	17:11	17:12	17:14	17:16
17:12	17:14	17:15	17:16	17:17	17:18	17:20	17:22
17:18	17:20	17:21	17:22	17:23	17:24	17:26	17:28
17:24	17:26	17:27	17:28	17:29	17:30	17:32	17:34
17:30	17:32	17:33	17:34	17:35	17:36	17:38	17:40
17:36	17:38	17:39	17:40	17:41	17:42	17:44	17:46
17:42	17:44	17:45	17:46	17:47	17:48	17:50	17:52
17:48	17:50	17:51	17:52	17:53	17:54	17:56	17:58
17:54	17:56	17:57	17:58	17:59	18:00	18:02	18:04
18:00	18:02	18:03	18:04	18:05	18:06	18:08	18:10
18:06	18:08	18:09	18:10	18:11	18:12	18:14	18:16
18:12	18:14	18:15	18:16	18:17	18:18	18:20	18:22
18:18	18:20	18:21	18:22	18:23	18:24	18:26	18:28
18:24	18:26	18:27	18:28	18:29	18:30	18:32	18:34
18:30	18:32	18:33	18:34	18:35	18:36	18:38	18:40
18:36	18:38	18:39	18:40	18:41	18:42	18:44	18:46
18:42	18:44	18:45	18:46	18:47	18:48	18:50	18:52
18:48	18:50	18:51	18:52	18:53	18:54	18:56	18:58
18:54	18:56	18:57	18:58	18:59	19:00	19:02	19:04
19:00	19:02	19:03	19:04	19:05	19:06	19:08	19:10
19:06	19:08	19:09	19:10	19:11	19:12	19:14	19:16
19:12	19:14	19:15	19:16	19:17	19:18	19:20	19:22
19:18	19:20	19:21	19:22	19:23	19:24	19:26	19:28
19:24	19:26	19:27	19:28	19:29	19:30	19:32	19:34
19:30	19:32	19:33	19:34	19:35	19:36	19:38	19:40
19:36	19:38	19:39	19:40	19:41	19:42	19:44	19:46
19:42	19:44	19:45	19:46	19:47	19:48	19:50	19:52
19:48	19:50	19:51	19:52	19:53	19:54	19:56	19:58
19:54	19:56	19:57	19:58	19:59	20:00	20:02	20:04
20:04	20:06	20:07	20:08	20:09	20:10	20:12	20:14
20:14	20:16	20:17	20:18	20:19	20:20	20:22	20:24
20:24	20:26	20:27	20:28	20:29	20:30	20:32	20:34
20:34	20:36	20:37	20:38	20:39	20:40	20:42	20:44
20:44	20:46	20:47	20:48	20:49	20:50	20:52	20:54
20:54	20:56	20:57	20:58	20:59	21:00	21:02	21:04
21:04	21:06	21:07	21:08	21:09	21:10	21:12	21:14
21:14	21:16	21:17	21:18	21:19	21:20	21:22	21:24

Westpark	Harras	Goetheplatz	Sendlinger Tor	Marienplatz	Odeonsplatz	Universität	M¨nchner Freiheit
21:24	21:26	21:27	21:28	21:29	21:30	21:32	21:34
21:34	21:36	21:37	21:38	21:39	21:40	21:42	21:44
21:44	21:46	21:47	21:48	21:49	21:50	21:52	21:54
21:54	21:56	21:57	21:58	21:59	22:00	22:02	22:04
22:04	22:06	22:07	22:08	22:09	22:10	22:12	22:14
22:14	22:16	22:17	22:18	22:19	22:20	22:22	22:24
22:24	22:26	22:27	22:28	22:29	22:30	22:32	22:34
22:34	22:36	22:37	22:38	22:39	22:40	22:42	22:44
22:44	22:46	22:47	22:48	22:49	22:50	22:52	22:54
22:54	22:56	22:57	22:58	22:59	23:00	23:02	23:04
23:14	23:16	23:17	23:18	23:19	23:20	23:22	23:24
23:34	23:36	23:37	23:38	23:39	23:40	23:42	23:44
23:54	23:56	23:57	23:58	23:59	00:00	00:02	00:04
00:14	00:16	00:17	00:18	00:19	00:20	00:22	00:24
00:34	00:36	00:37	00:38	00:39	00:40	00:42	00:44
00:54	00:56	00:57	00:58	00:59	01:00	01:02	01:04
01:14	01:16	01:17	01:18	01:19	01:20	01:22	01:24
01:34	01:36	01:37	01:38	01:39	01:40	01:42	01:44
01:54	01:56	01:57	01:58	01:59	02:00	02:02	02:04

U4 — Münchner Freiheit → Westpark

Münchner Freiheit	Universität	Odeonsplatz	Marienplatz	Sendlinger Tor	Goetheplatz	Harras	Westpark
04:36	04:37	04:39	04:40	04:41	04:42	04:43	04:45
04:46	04:47	04:49	04:50	04:51	04:52	04:53	04:55
04:56	04:57	04:59	05:00	05:01	05:02	05:03	05:05
05:06	05:07	05:09	05:10	05:11	05:12	05:13	05:15
05:16	05:17	05:19	05:20	05:21	05:22	05:23	05:25
05:26	05:27	05:29	05:30	05:31	05:32	05:33	05:35
05:36	05:37	05:39	05:40	05:41	05:42	05:43	05:45
05:46	05:47	05:49	05:50	05:51	05:52	05:53	05:55
05:56	05:57	05:59	06:00	06:01	06:02	06:03	06:05
06:02	06:03	06:05	06:06	06:07	06:08	06:09	06:11
06:08	06:09	06:11	06:12	06:13	06:14	06:15	06:17
06:14	06:15	06:17	06:18	06:19	06:20	06:21	06:23
06:20	06:21	06:23	06:24	06:25	06:26	06:27	06:29
06:26	06:27	06:29	06:30	06:31	06:32	06:33	06:35
06:32	06:33	06:35	06:36	06:37	06:38	06:39	06:41
06:38	06:39	06:41	06:42	06:43	06:44	06:45	06:47
06:44	06:45	06:47	06:48	06:49	06:50	06:51	06:53
06:50	06:51	06:53	06:54	06:55	06:56	06:57	06:59
06:56	06:57	06:59	07:00	07:01	07:02	07:03	07:05
07:02	07:03	07:05	07:06	07:07	07:08	07:09	07:11
07:08	07:09	07:11	07:12	07:13	07:14	07:15	07:17
07:14	07:15	07:17	07:18	07:19	07:20	07:21	07:23
07:20	07:21	07:23	07:24	07:25	07:26	07:27	07:29
07:26	07:27	07:29	07:30	07:31	07:32	07:33	07:35
07:32	07:33	07:35	07:36	07:37	07:38	07:39	07:41
07:38	07:39	07:41	07:42	07:43	07:44	07:45	07:47
07:44	07:45	07:47	07:48	07:49	07:50	07:51	07:53
07:50	07:51	07:53	07:54	07:55	07:56	07:57	07:59
07:56	07:57	07:59	08:00	08:01	08:02	08:03	08:05
08:02	08:03	08:05	08:06	08:07	08:08	08:09	08:11
08:08	08:09	08:11	08:12	08:13	08:14	08:15	08:17
08:14	08:15	08:17	08:18	08:19	08:20	08:21	08:23
08:20	08:21	08:23	08:24	08:25	08:26	08:27	08:29
08:26	08:27	08:29	08:30	08:31	08:32	08:33	08:35
08:32	08:33	08:35	08:36	08:37	08:38	08:39	08:41
08:38	08:39	08:41	08:42	08:43	08:44	08:45	08:47
08:44	08:45	08:47	08:48	08:49	08:50	08:51	08:53
08:50	08:51	08:53	08:54	08:55	08:56	08:57	08:59
08:56	08:57	08:59	09:00	09:01	09:02	09:03	09:05
09:02	09:03	09:05	09:06	09:07	09:08	09:09	09:11
09:08	09:09	09:11	09:12	09:13	09:14	09:15	09:17
09:14	09:15	09:17	09:18	09:19	09:20	09:21	09:23
09:20	09:21	09:23	09:24	09:25	09:26	09:27	09:29
09:26	09:27	09:29	09:30	09:31	09:32	09:33	09:35
09:32	09:33	09:35	09:36	09:37	09:38	09:39	09:41
09:38	09:39	09:41	09:42	09:43	09:44	09:45	09:47
09:44	09:45	09:47	09:48	09:49	09:50	09:51	09:53
09:50	09:51	09:53	09:54	09:55	09:56	09:57	09:59
09:56	09:57	09:59	10:00	10:01	10:02	10:03	10:05
10:02	10:03	10:05	10:06	10:07	10:08	10:09	10:11
10:08	10:09	10:11	10:12	10:13	10:14	10:15	10:17
10:14	10:15	10:17	10:18	10:19	10:20	10:21	10:23

Münchner Freiheit	Universität	Odeonsplatz	Marienplatz	Sendlinger Tor	Goetheplatz	Harras	Westpark
10:20	10:21	10:23	10:24	10:25	10:26	10:27	10:29
10:26	10:27	10:29	10:30	10:31	10:32	10:33	10:35
10:32	10:33	10:35	10:36	10:37	10:38	10:39	10:41
10:38	10:39	10:41	10:42	10:43	10:44	10:45	10:47
10:44	10:45	10:47	10:48	10:49	10:50	10:51	10:53
10:50	10:51	10:53	10:54	10:55	10:56	10:57	10:59
10:56	10:57	10:59	11:00	11:01	11:02	11:03	11:05
11:02	11:03	11:05	11:06	11:07	11:08	11:09	11:11
11:08	11:09	11:11	11:12	11:13	11:14	11:15	11:17
11:14	11:15	11:17	11:18	11:19	11:20	11:21	11:23
11:20	11:21	11:23	11:24	11:25	11:26	11:27	11:29
11:26	11:27	11:29	11:30	11:31	11:32	11:33	11:35
11:32	11:33	11:35	11:36	11:37	11:38	11:39	11:41
11:38	11:39	11:41	11:42	11:43	11:44	11:45	11:47
11:44	11:45	11:47	11:48	11:49	11:50	11:51	11:53
11:50	11:51	11:53	11:54	11:55	11:56	11:57	11:59
11:56	11:57	11:59	12:00	12:01	12:02	12:03	12:05
12:02	12:03	12:05	12:06	12:07	12:08	12:09	12:11
12:08	12:09	12:11	12:12	12:13	12:14	12:15	12:17
12:14	12:15	12:17	12:18	12:19	12:20	12:21	12:23
12:20	12:21	12:23	12:24	12:25	12:26	12:27	12:29
12:26	12:27	12:29	12:30	12:31	12:32	12:33	12:35
12:32	12:33	12:35	12:36	12:37	12:38	12:39	12:41
12:38	12:39	12:41	12:42	12:43	12:44	12:45	12:47
12:44	12:45	12:47	12:48	12:49	12:50	12:51	12:53
12:50	12:51	12:53	12:54	12:55	12:56	12:57	12:59
12:56	12:57	12:59	13:00	13:01	13:02	13:03	13:05
13:02	13:03	13:05	13:06	13:07	13:08	13:09	13:11
13:08	13:09	13:11	13:12	13:13	13:14	13:15	13:17
13:14	13:15	13:17	13:18	13:19	13:20	13:21	13:23
13:20	13:21	13:23	13:24	13:25	13:26	13:27	13:29
13:26	13:27	13:29	13:30	13:31	13:32	13:33	13:35
13:32	13:33	13:35	13:36	13:37	13:38	13:39	13:41
13:38	13:39	13:41	13:42	13:43	13:44	13:45	13:47
13:44	13:45	13:47	13:48	13:49	13:50	13:51	13:53
13:50	13:51	13:53	13:54	13:55	13:56	13:57	13:59
13:56	13:57	13:59	14:00	14:01	14:02	14:03	14:05
14:02	14:03	14:05	14:06	14:07	14:08	14:09	14:11
14:08	14:09	14:11	14:12	14:13	14:14	14:15	14:17
14:14	14:15	14:17	14:18	14:19	14:20	14:21	14:23
14:20	14:21	14:23	14:24	14:25	14:26	14:27	14:29
14:26	14:27	14:29	14:30	14:31	14:32	14:33	14:35
14:32	14:33	14:35	14:36	14:37	14:38	14:39	14:41
14:38	14:39	14:41	14:42	14:43	14:44	14:45	14:47
14:44	14:45	14:47	14:48	14:49	14:50	14:51	14:53
14:50	14:51	14:53	14:54	14:55	14:56	14:57	14:59
14:56	14:57	14:59	15:00	15:01	15:02	15:03	15:05
15:02	15:03	15:05	15:06	15:07	15:08	15:09	15:11
15:08	15:09	15:11	15:12	15:13	15:14	15:15	15:17
15:14	15:15	15:17	15:18	15:19	15:20	15:21	15:23
15:20	15:21	15:23	15:24	15:25	15:26	15:27	15:29
15:26	15:27	15:29	15:30	15:31	15:32	15:33	15:35
15:32	15:33	15:35	15:36	15:37	15:38	15:39	15:41

Münchner Freiheit	Universität	Odeonsplatz	Marienplatz	Sendlinger Tor	Goetheplatz	Harras	Westpark
15:38	15:39	15:41	15:42	15:43	15:44	15:45	15:47
15:44	15:45	15:47	15:48	15:49	15:50	15:51	15:53
15:50	15:51	15:53	15:54	15:55	15:56	15:57	15:59
15:56	15:57	15:59	16:00	16:01	16:02	16:03	16:05
16:02	16:03	16:05	16:06	16:07	16:08	16:09	16:11
16:08	16:09	16:11	16:12	16:13	16:14	16:15	16:17
16:14	16:15	16:17	16:18	16:19	16:20	16:21	16:23
16:20	16:21	16:23	16:24	16:25	16:26	16:27	16:29
16:26	16:27	16:29	16:30	16:31	16:32	16:33	16:35
16:32	16:33	16:35	16:36	16:37	16:38	16:39	16:41
16:38	16:39	16:41	16:42	16:43	16:44	16:45	16:47
16:44	16:45	16:47	16:48	16:49	16:50	16:51	16:53
16:50	16:51	16:53	16:54	16:55	16:56	16:57	16:59
16:56	16:57	16:59	17:00	17:01	17:02	17:03	17:05
17:02	17:03	17:05	17:06	17:07	17:08	17:09	17:11
17:08	17:09	17:11	17:12	17:13	17:14	17:15	17:17
17:14	17:15	17:17	17:18	17:19	17:20	17:21	17:23
17:20	17:21	17:23	17:24	17:25	17:26	17:27	17:29
17:26	17:27	17:29	17:30	17:31	17:32	17:33	17:35
17:32	17:33	17:35	17:36	17:37	17:38	17:39	17:41
17:38	17:39	17:41	17:42	17:43	17:44	17:45	17:47
17:44	17:45	17:47	17:48	17:49	17:50	17:51	17:53
17:50	17:51	17:53	17:54	17:55	17:56	17:57	17:59
17:56	17:57	17:59	18:00	18:01	18:02	18:03	18:05
18:02	18:03	18:05	18:06	18:07	18:08	18:09	18:11
18:08	18:09	18:11	18:12	18:13	18:14	18:15	18:17
18:14	18:15	18:17	18:18	18:19	18:20	18:21	18:23
18:20	18:21	18:23	18:24	18:25	18:26	18:27	18:29
18:26	18:27	18:29	18:30	18:31	18:32	18:33	18:35
18:32	18:33	18:35	18:36	18:37	18:38	18:39	18:41
18:38	18:39	18:41	18:42	18:43	18:44	18:45	18:47
18:44	18:45	18:47	18:48	18:49	18:50	18:51	18:53
18:50	18:51	18:53	18:54	18:55	18:56	18:57	18:59
18:56	18:57	18:59	19:00	19:01	19:02	19:03	19:05
19:02	19:03	19:05	19:06	19:07	19:08	19:09	19:11
19:08	19:09	19:11	19:12	19:13	19:14	19:15	19:17
19:14	19:15	19:17	19:18	19:19	19:20	19:21	19:23
19:20	19:21	19:23	19:24	19:25	19:26	19:27	19:29
19:26	19:27	19:29	19:30	19:31	19:32	19:33	19:35
19:32	19:33	19:35	19:36	19:37	19:38	19:39	19:41
19:38	19:39	19:41	19:42	19:43	19:44	19:45	19:47
19:44	19:45	19:47	19:48	19:49	19:50	19:51	19:53
19:50	19:51	19:53	19:54	19:55	19:56	19:57	19:59
19:56	19:57	19:59	20:00	20:01	20:02	20:03	20:05
20:06	20:07	20:09	20:10	20:11	20:12	20:13	20:15
20:16	20:17	20:19	20:20	20:21	20:22	20:23	20:25
20:26	20:27	20:29	20:30	20:31	20:32	20:33	20:35
20:36	20:37	20:39	20:40	20:41	20:42	20:43	20:45
20:46	20:47	20:49	20:50	20:51	20:52	20:53	20:55
20:56	20:57	20:59	21:00	21:01	21:02	21:03	21:05
21:06	21:07	21:09	21:10	21:11	21:12	21:13	21:15
21:16	21:17	21:19	21:20	21:21	21:22	21:23	21:25
21:26	21:27	21:29	21:30	21:31	21:32	21:33	21:35

Münchner Freiheit	Universität	Odeonsplatz	Marienplatz	Sendlinger Tor	Goetheplatz	Harras	Westpark
21:36	21:37	21:39	21:40	21:41	21:42	21:43	21:45
21:46	21:47	21:49	21:50	21:51	21:52	21:53	21:55
21:56	21:57	21:59	22:00	22:01	22:02	22:03	22:05
22:06	22:07	22:09	22:10	22:11	22:12	22:13	22:15
22:16	22:17	22:19	22:20	22:21	22:22	22:23	22:25
22:26	22:27	22:29	22:30	22:31	22:32	22:33	22:35
22:36	22:37	22:39	22:40	22:41	22:42	22:43	22:45
22:46	22:47	22:49	22:50	22:51	22:52	22:53	22:55
22:56	22:57	22:59	23:00	23:01	23:02	23:03	23:05
23:16	23:17	23:19	23:20	23:21	23:22	23:23	23:25
23:36	23:37	23:39	23:40	23:41	23:42	23:43	23:45
23:56	23:57	23:59	00:00	00:01	00:02	00:03	00:05
00:16	00:17	00:19	00:20	00:21	00:22	00:23	00:25
00:36	00:37	00:39	00:40	00:41	00:42	00:43	00:45
00:56	00:57	00:59	01:00	01:01	01:02	01:03	01:05
01:16	01:17	01:19	01:20	01:21	01:22	01:23	01:25
01:36	01:37	01:39	01:40	01:41	01:42	01:43	01:45

U5 — Olympiazentrum → Westendstr.

Olympiazentrum	Petuelring	Scheidplatz	Josephsplatz	Königsplatz	Hauptbahnhof	Theresienwiese	Heimeranplatz	Westendstr.
04:35	04:36	04:37	04:38	04:39	04:40	04:41	04:42	04:43
04:45	04:46	04:47	04:48	04:49	04:50	04:51	04:52	04:53
04:55	04:56	04:57	04:58	04:59	05:00	05:01	05:02	05:03
05:05	05:06	05:07	05:08	05:09	05:10	05:11	05:12	05:13
05:15	05:16	05:17	05:18	05:19	05:20	05:21	05:22	05:23
05:25	05:26	05:27	05:28	05:29	05:30	05:31	05:32	05:33
05:35	05:36	05:37	05:38	05:39	05:40	05:41	05:42	05:43
05:45	05:46	05:47	05:48	05:49	05:50	05:51	05:52	05:53
05:55	05:56	05:57	05:58	05:59	06:00	06:01	06:02	06:03
06:01	06:02	06:03	06:04	06:05	06:06	06:07	06:08	06:09
06:07	06:08	06:09	06:10	06:11	06:12	06:13	06:14	06:15
06:13	06:14	06:15	06:16	06:17	06:18	06:19	06:20	06:21
06:19	06:20	06:21	06:22	06:23	06:24	06:25	06:26	06:27
06:25	06:26	06:27	06:28	06:29	06:30	06:31	06:32	06:33
06:31	06:32	06:33	06:34	06:35	06:36	06:37	06:38	06:39
06:37	06:38	06:39	06:40	06:41	06:42	06:43	06:44	06:45
06:43	06:44	06:45	06:46	06:47	06:48	06:49	06:50	06:51
06:49	06:50	06:51	06:52	06:53	06:54	06:55	06:56	06:57
06:55	06:56	06:57	06:58	06:59	07:00	07:01	07:02	07:03
07:01	07:02	07:03	07:04	07:05	07:06	07:07	07:08	07:09
07:07	07:08	07:09	07:10	07:11	07:12	07:13	07:14	07:15
07:13	07:14	07:15	07:16	07:17	07:18	07:19	07:20	07:21
07:19	07:20	07:21	07:22	07:23	07:24	07:25	07:26	07:27
07:25	07:26	07:27	07:28	07:29	07:30	07:31	07:32	07:33
07:31	07:32	07:33	07:34	07:35	07:36	07:37	07:38	07:39
07:37	07:38	07:39	07:40	07:41	07:42	07:43	07:44	07:45
07:43	07:44	07:45	07:46	07:47	07:48	07:49	07:50	07:51
07:49	07:50	07:51	07:52	07:53	07:54	07:55	07:56	07:57
07:55	07:56	07:57	07:58	07:59	08:00	08:01	08:02	08:03
08:01	08:02	08:03	08:04	08:05	08:06	08:07	08:08	08:09
08:07	08:08	08:09	08:10	08:11	08:12	08:13	08:14	08:15
08:13	08:14	08:15	08:16	08:17	08:18	08:19	08:20	08:21
08:19	08:20	08:21	08:22	08:23	08:24	08:25	08:26	08:27
08:25	08:26	08:27	08:28	08:29	08:30	08:31	08:32	08:33
08:31	08:32	08:33	08:34	08:35	08:36	08:37	08:38	08:39
08:37	08:38	08:39	08:40	08:41	08:42	08:43	08:44	08:45
08:43	08:44	08:45	08:46	08:47	08:48	08:49	08:50	08:51
08:49	08:50	08:51	08:52	08:53	08:54	08:55	08:56	08:57
08:55	08:56	08:57	08:58	08:59	09:00	09:01	09:02	09:03
09:01	09:02	09:03	09:04	09:05	09:06	09:07	09:08	09:09
09:07	09:08	09:09	09:10	09:11	09:12	09:13	09:14	09:15
09:13	09:14	09:15	09:16	09:17	09:18	09:19	09:20	09:21
09:19	09:20	09:21	09:22	09:23	09:24	09:25	09:26	09:27
09:25	09:26	09:27	09:28	09:29	09:30	09:31	09:32	09:33
09:31	09:32	09:33	09:34	09:35	09:36	09:37	09:38	09:39
09:37	09:38	09:39	09:40	09:41	09:42	09:43	09:44	09:45
09:43	09:44	09:45	09:46	09:47	09:48	09:49	09:50	09:51
09:49	09:50	09:51	09:52	09:53	09:54	09:55	09:56	09:57
09:55	09:56	09:57	09:58	09:59	10:00	10:01	10:02	10:03
10:01	10:02	10:03	10:04	10:05	10:06	10:07	10:08	10:09
10:07	10:08	10:09	10:10	10:11	10:12	10:13	10:14	10:15
10:13	10:14	10:15	10:16	10:17	10:18	10:19	10:20	10:21

Olympiazentrum	Petuelring	Scheidplatz	Josephsplatz	Königsplatz	Hauptbahnhof	Theresienwiese	Heimeranplatz	Westendstr.
10:19	10:20	10:21	10:22	10:23	10:24	10:25	10:26	10:27
10:25	10:26	10:27	10:28	10:29	10:30	10:31	10:32	10:33
10:31	10:32	10:33	10:34	10:35	10:36	10:37	10:38	10:39
10:37	10:38	10:39	10:40	10:41	10:42	10:43	10:44	10:45
10:43	10:44	10:45	10:46	10:47	10:48	10:49	10:50	10:51
10:49	10:50	10:51	10:52	10:53	10:54	10:55	10:56	10:57
10:55	10:56	10:57	10:58	10:59	11:00	11:01	11:02	11:03
11:01	11:02	11:03	11:04	11:05	11:06	11:07	11:08	11:09
11:07	11:08	11:09	11:10	11:11	11:12	11:13	11:14	11:15
11:13	11:14	11:15	11:16	11:17	11:18	11:19	11:20	11:21
11:19	11:20	11:21	11:22	11:23	11:24	11:25	11:26	11:27
11:25	11:26	11:27	11:28	11:29	11:30	11:31	11:32	11:33
11:31	11:32	11:33	11:34	11:35	11:36	11:37	11:38	11:39
11:37	11:38	11:39	11:40	11:41	11:42	11:43	11:44	11:45
11:43	11:44	11:45	11:46	11:47	11:48	11:49	11:50	11:51
11:49	11:50	11:51	11:52	11:53	11:54	11:55	11:56	11:57
11:55	11:56	11:57	11:58	11:59	12:00	12:01	12:02	12:03
12:01	12:02	12:03	12:04	12:05	12:06	12:07	12:08	12:09
12:07	12:08	12:09	12:10	12:11	12:12	12:13	12:14	12:15
12:13	12:14	12:15	12:16	12:17	12:18	12:19	12:20	12:21
12:19	12:20	12:21	12:22	12:23	12:24	12:25	12:26	12:27
12:25	12:26	12:27	12:28	12:29	12:30	12:31	12:32	12:33
12:31	12:32	12:33	12:34	12:35	12:36	12:37	12:38	12:39
12:37	12:38	12:39	12:40	12:41	12:42	12:43	12:44	12:45
12:43	12:44	12:45	12:46	12:47	12:48	12:49	12:50	12:51
12:49	12:50	12:51	12:52	12:53	12:54	12:55	12:56	12:57
12:55	12:56	12:57	12:58	12:59	13:00	13:01	13:02	13:03
13:01	13:02	13:03	13:04	13:05	13:06	13:07	13:08	13:09
13:07	13:08	13:09	13:10	13:11	13:12	13:13	13:14	13:15
13:13	13:14	13:15	13:16	13:17	13:18	13:19	13:20	13:21
13:19	13:20	13:21	13:22	13:23	13:24	13:25	13:26	13:27
13:25	13:26	13:27	13:28	13:29	13:30	13:31	13:32	13:33
13:31	13:32	13:33	13:34	13:35	13:36	13:37	13:38	13:39
13:37	13:38	13:39	13:40	13:41	13:42	13:43	13:44	13:45
13:43	13:44	13:45	13:46	13:47	13:48	13:49	13:50	13:51
13:49	13:50	13:51	13:52	13:53	13:54	13:55	13:56	13:57
13:55	13:56	13:57	13:58	13:59	14:00	14:01	14:02	14:03
14:01	14:02	14:03	14:04	14:05	14:06	14:07	14:08	14:09
14:07	14:08	14:09	14:10	14:11	14:12	14:13	14:14	14:15
14:13	14:14	14:15	14:16	14:17	14:18	14:19	14:20	14:21
14:19	14:20	14:21	14:22	14:23	14:24	14:25	14:26	14:27
14:25	14:26	14:27	14:28	14:29	14:30	14:31	14:32	14:33
14:31	14:32	14:33	14:34	14:35	14:36	14:37	14:38	14:39
14:37	14:38	14:39	14:40	14:41	14:42	14:43	14:44	14:45
14:43	14:44	14:45	14:46	14:47	14:48	14:49	14:50	14:51
14:49	14:50	14:51	14:52	14:53	14:54	14:55	14:56	14:57
14:55	14:56	14:57	14:58	14:59	15:00	15:01	15:02	15:03
15:01	15:02	15:03	15:04	15:05	15:06	15:07	15:08	15:09
15:07	15:08	15:09	15:10	15:11	15:12	15:13	15:14	15:15
15:13	15:14	15:15	15:16	15:17	15:18	15:19	15:20	15:21
15:19	15:20	15:21	15:22	15:23	15:24	15:25	15:26	15:27
15:25	15:26	15:27	15:28	15:29	15:30	15:31	15:32	15:33
15:31	15:32	15:33	15:34	15:35	15:36	15:37	15:38	15:39
15:37	15:38	15:39	15:40	15:41	15:42	15:43	15:44	15:45

Olympiazentrum	Petuelring	Scheidplatz	Josephsplatz	Königsplatz	Hauptbahnhof	Theresienwiese	Heimeranplatz	Westendstr.
15:43	15:44	15:45	15:46	15:47	15:48	15:49	15:50	15:51
15:49	15:50	15:51	15:52	15:53	15:54	15:55	15:56	15:57
15:55	15:56	15:57	15:58	15:59	16:00	16:01	16:02	16:03
16:01	16:02	16:03	16:04	16:05	16:06	16:07	16:08	16:09
16:07	16:08	16:09	16:10	16:11	16:12	16:13	16:14	16:15
16:13	16:14	16:15	16:16	16:17	16:18	16:19	16:20	16:21
16:19	16:20	16:21	16:22	16:23	16:24	16:25	16:26	16:27
16:25	16:26	16:27	16:28	16:29	16:30	16:31	16:32	16:33
16:31	16:32	16:33	16:34	16:35	16:36	16:37	16:38	16:39
16:37	16:38	16:39	16:40	16:41	16:42	16:43	16:44	16:45
16:43	16:44	16:45	16:46	16:47	16:48	16:49	16:50	16:51
16:49	16:50	16:51	16:52	16:53	16:54	16:55	16:56	16:57
16:55	16:56	16:57	16:58	16:59	17:00	17:01	17:02	17:03
17:01	17:02	17:03	17:04	17:05	17:06	17:07	17:08	17:09
17:07	17:08	17:09	17:10	17:11	17:12	17:13	17:14	17:15
17:13	17:14	17:15	17:16	17:17	17:18	17:19	17:20	17:21
17:19	17:20	17:21	17:22	17:23	17:24	17:25	17:26	17:27
17:25	17:26	17:27	17:28	17:29	17:30	17:31	17:32	17:33
17:31	17:32	17:33	17:34	17:35	17:36	17:37	17:38	17:39
17:37	17:38	17:39	17:40	17:41	17:42	17:43	17:44	17:45
17:43	17:44	17:45	17:46	17:47	17:48	17:49	17:50	17:51
17:49	17:50	17:51	17:52	17:53	17:54	17:55	17:56	17:57
17:55	17:56	17:57	17:58	17:59	18:00	18:01	18:02	18:03
18:01	18:02	18:03	18:04	18:05	18:06	18:07	18:08	18:09
18:07	18:08	18:09	18:10	18:11	18:12	18:13	18:14	18:15
18:13	18:14	18:15	18:16	18:17	18:18	18:19	18:20	18:21
18:19	18:20	18:21	18:22	18:23	18:24	18:25	18:26	18:27
18:25	18:26	18:27	18:28	18:29	18:30	18:31	18:32	18:33
18:31	18:32	18:33	18:34	18:35	18:36	18:37	18:38	18:39
18:37	18:38	18:39	18:40	18:41	18:42	18:43	18:44	18:45
18:43	18:44	18:45	18:46	18:47	18:48	18:49	18:50	18:51
18:49	18:50	18:51	18:52	18:53	18:54	18:55	18:56	18:57
18:55	18:56	18:57	18:58	18:59	19:00	19:01	19:02	19:03
19:01	19:02	19:03	19:04	19:05	19:06	19:07	19:08	19:09
19:07	19:08	19:09	19:10	19:11	19:12	19:13	19:14	19:15
19:13	19:14	19:15	19:16	19:17	19:18	19:19	19:20	19:21
19:19	19:20	19:21	19:22	19:23	19:24	19:25	19:26	19:27
19:25	19:26	19:27	19:28	19:29	19:30	19:31	19:32	19:33
19:31	19:32	19:33	19:34	19:35	19:36	19:37	19:38	19:39
19:37	19:38	19:39	19:40	19:41	19:42	19:43	19:44	19:45
19:43	19:44	19:45	19:46	19:47	19:48	19:49	19:50	19:51
19:49	19:50	19:51	19:52	19:53	19:54	19:55	19:56	19:57
19:55	19:56	19:57	19:58	19:59	20:00	20:01	20:02	20:03
20:05	20:06	20:07	20:08	20:09	20:10	20:11	20:12	20:13
20:15	20:16	20:17	20:18	20:19	20:20	20:21	20:22	20:23
20:25	20:26	20:27	20:28	20:29	20:30	20:31	20:32	20:33
20:35	20:36	20:37	20:38	20:39	20:40	20:41	20:42	20:43
20:45	20:46	20:47	20:48	20:49	20:50	20:51	20:52	20:53
20:55	20:56	20:57	20:58	20:59	21:00	21:01	21:02	21:03
21:05	21:06	21:07	21:08	21:09	21:10	21:11	21:12	21:13
21:15	21:16	21:17	21:18	21:19	21:20	21:21	21:22	21:23
21:25	21:26	21:27	21:28	21:29	21:30	21:31	21:32	21:33
21:35	21:36	21:37	21:38	21:39	21:40	21:41	21:42	21:43
21:45	21:46	21:47	21:48	21:49	21:50	21:51	21:52	21:53

Olympiazentrum	Petuelring	Scheidplatz	Josephsplatz	Königsplatz	Hauptbahnhof	Theresienwiese	Heimeranplatz	Westendstr.
21:55	21:56	21:57	21:58	21:59	22:00	22:01	22:02	22:03
22:05	22:06	22:07	22:08	22:09	22:10	22:11	22:12	22:13
22:15	22:16	22:17	22:18	22:19	22:20	22:21	22:22	22:23
22:25	22:26	22:27	22:28	22:29	22:30	22:31	22:32	22:33
22:35	22:36	22:37	22:38	22:39	22:40	22:41	22:42	22:43
22:45	22:46	22:47	22:48	22:49	22:50	22:51	22:52	22:53
22:55	22:56	22:57	22:58	22:59	23:00	23:01	23:02	23:03
23:15	23:16	23:17	23:18	23:19	23:20	23:21	23:22	23:23
23:35	23:36	23:37	23:38	23:39	23:40	23:41	23:42	23:43
23:55	23:56	23:57	23:58	23:59	00:00	00:01	00:02	00:03
00:15	00:16	00:17	00:18	00:19	00:20	00:21	00:22	00:23
00:35	00:36	00:37	00:38	00:39	00:40	00:41	00:42	00:43
00:55	00:56	00:57	00:58	00:59	01:00	01:01	01:02	01:03
01:15	01:16	01:17	01:18	01:19	01:20	01:21	01:22	01:23
01:35	01:36	01:37	01:38	01:39	01:40	01:41	01:42	01:43

U5 — Westendstr. → Olympiazentrum

Westendstr.	Heimeranplatz	Theresienwiese	Hauptbahnhof	Königsplatz	Josephsplatz	Scheidplatz	Petuelring	Olympiazentrum
04:45	04:46	04:47	04:48	04:49	04:50	04:51	04:52	04:53
04:55	04:56	04:57	04:58	04:59	05:00	05:01	05:02	05:03
05:05	05:06	05:07	05:08	05:09	05:10	05:11	05:12	05:13
05:15	05:16	05:17	05:18	05:19	05:20	05:21	05:22	05:23
05:25	05:26	05:27	05:28	05:29	05:30	05:31	05:32	05:33
05:35	05:36	05:37	05:38	05:39	05:40	05:41	05:42	05:43
05:45	05:46	05:47	05:48	05:49	05:50	05:51	05:52	05:53
05:55	05:56	05:57	05:58	05:59	06:00	06:01	06:02	06:03
06:01	06:02	06:03	06:04	06:05	06:06	06:07	06:08	06:09
06:07	06:08	06:09	06:10	06:11	06:12	06:13	06:14	06:15
06:13	06:14	06:15	06:16	06:17	06:18	06:19	06:20	06:21
06:19	06:20	06:21	06:22	06:23	06:24	06:25	06:26	06:27
06:25	06:26	06:27	06:28	06:29	06:30	06:31	06:32	06:33
06:31	06:32	06:33	06:34	06:35	06:36	06:37	06:38	06:39
06:37	06:38	06:39	06:40	06:41	06:42	06:43	06:44	06:45
06:43	06:44	06:45	06:46	06:47	06:48	06:49	06:50	06:51
06:49	06:50	06:51	06:52	06:53	06:54	06:55	06:56	06:57
06:55	06:56	06:57	06:58	06:59	07:00	07:01	07:02	07:03
07:01	07:02	07:03	07:04	07:05	07:06	07:07	07:08	07:09
07:07	07:08	07:09	07:10	07:11	07:12	07:13	07:14	07:15
07:13	07:14	07:15	07:16	07:17	07:18	07:19	07:20	07:21
07:19	07:20	07:21	07:22	07:23	07:24	07:25	07:26	07:27
07:25	07:26	07:27	07:28	07:29	07:30	07:31	07:32	07:33
07:31	07:32	07:33	07:34	07:35	07:36	07:37	07:38	07:39
07:37	07:38	07:39	07:40	07:41	07:42	07:43	07:44	07:45
07:43	07:44	07:45	07:46	07:47	07:48	07:49	07:50	07:51
07:49	07:50	07:51	07:52	07:53	07:54	07:55	07:56	07:57
07:55	07:56	07:57	07:58	07:59	08:00	08:01	08:02	08:03
08:01	08:02	08:03	08:04	08:05	08:06	08:07	08:08	08:09
08:07	08:08	08:09	08:10	08:11	08:12	08:13	08:14	08:15
08:13	08:14	08:15	08:16	08:17	08:18	08:19	08:20	08:21
08:19	08:20	08:21	08:22	08:23	08:24	08:25	08:26	08:27
08:25	08:26	08:27	08:28	08:29	08:30	08:31	08:32	08:33
08:31	08:32	08:33	08:34	08:35	08:36	08:37	08:38	08:39
08:37	08:38	08:39	08:40	08:41	08:42	08:43	08:44	08:45
08:43	08:44	08:45	08:46	08:47	08:48	08:49	08:50	08:51
08:49	08:50	08:51	08:52	08:53	08:54	08:55	08:56	08:57
08:55	08:56	08:57	08:58	08:59	09:00	09:01	09:02	09:03
09:01	09:02	09:03	09:04	09:05	09:06	09:07	09:08	09:09
09:07	09:08	09:09	09:10	09:11	09:12	09:13	09:14	09:15
09:13	09:14	09:15	09:16	09:17	09:18	09:19	09:20	09:21
09:19	09:20	09:21	09:22	09:23	09:24	09:25	09:26	09:27
09:25	09:26	09:27	09:28	09:29	09:30	09:31	09:32	09:33
09:31	09:32	09:33	09:34	09:35	09:36	09:37	09:38	09:39
09:37	09:38	09:39	09:40	09:41	09:42	09:43	09:44	09:45
09:43	09:44	09:45	09:46	09:47	09:48	09:49	09:50	09:51
09:49	09:50	09:51	09:52	09:53	09:54	09:55	09:56	09:57
09:55	09:56	09:57	09:58	09:59	10:00	10:01	10:02	10:03
10:01	10:02	10:03	10:04	10:05	10:06	10:07	10:08	10:09
10:07	10:08	10:09	10:10	10:11	10:12	10:13	10:14	10:15
10:13	10:14	10:15	10:16	10:17	10:18	10:19	10:20	10:21
10:19	10:20	10:21	10:22	10:23	10:24	10:25	10:26	10:27

Westendstr.	Heimeranplatz	Theresienwiese	Hauptbahnhof	Königsplatz	Josephsplatz	Scheidplatz	Petuelring	Olympiazentrum
10:25	10:26	10:27	10:28	10:29	10:30	10:31	10:32	10:33
10:31	10:32	10:33	10:34	10:35	10:36	10:37	10:38	10:39
10:37	10:38	10:39	10:40	10:41	10:42	10:43	10:44	10:45
10:43	10:44	10:45	10:46	10:47	10:48	10:49	10:50	10:51
10:49	10:50	10:51	10:52	10:53	10:54	10:55	10:56	10:57
10:55	10:56	10:57	10:58	10:59	11:00	11:01	11:02	11:03
11:01	11:02	11:03	11:04	11:05	11:06	11:07	11:08	11:09
11:07	11:08	11:09	11:10	11:11	11:12	11:13	11:14	11:15
11:13	11:14	11:15	11:16	11:17	11:18	11:19	11:20	11:21
11:19	11:20	11:21	11:22	11:23	11:24	11:25	11:26	11:27
11:25	11:26	11:27	11:28	11:29	11:30	11:31	11:32	11:33
11:31	11:32	11:33	11:34	11:35	11:36	11:37	11:38	11:39
11:37	11:38	11:39	11:40	11:41	11:42	11:43	11:44	11:45
11:43	11:44	11:45	11:46	11:47	11:48	11:49	11:50	11:51
11:49	11:50	11:51	11:52	11:53	11:54	11:55	11:56	11:57
11:55	11:56	11:57	11:58	11:59	12:00	12:01	12:02	12:03
12:01	12:02	12:03	12:04	12:05	12:06	12:07	12:08	12:09
12:07	12:08	12:09	12:10	12:11	12:12	12:13	12:14	12:15
12:13	12:14	12:15	12:16	12:17	12:18	12:19	12:20	12:21
12:19	12:20	12:21	12:22	12:23	12:24	12:25	12:26	12:27
12:25	12:26	12:27	12:28	12:29	12:30	12:31	12:32	12:33
12:31	12:32	12:33	12:34	12:35	12:36	12:37	12:38	12:39
12:37	12:38	12:39	12:40	12:41	12:42	12:43	12:44	12:45
12:43	12:44	12:45	12:46	12:47	12:48	12:49	12:50	12:51
12:49	12:50	12:51	12:52	12:53	12:54	12:55	12:56	12:57
12:55	12:56	12:57	12:58	12:59	13:00	13:01	13:02	13:03
13:01	13:02	13:03	13:04	13:05	13:06	13:07	13:08	13:09
13:07	13:08	13:09	13:10	13:11	13:12	13:13	13:14	13:15
13:13	13:14	13:15	13:16	13:17	13:18	13:19	13:20	13:21
13:19	13:20	13:21	13:22	13:23	13:24	13:25	13:26	13:27
13:25	13:26	13:27	13:28	13:29	13:30	13:31	13:32	13:33
13:31	13:32	13:33	13:34	13:35	13:36	13:37	13:38	13:39
13:37	13:38	13:39	13:40	13:41	13:42	13:43	13:44	13:45
13:43	13:44	13:45	13:46	13:47	13:48	13:49	13:50	13:51
13:49	13:50	13:51	13:52	13:53	13:54	13:55	13:56	13:57
13:55	13:56	13:57	13:58	13:59	14:00	14:01	14:02	14:03
14:01	14:02	14:03	14:04	14:05	14:06	14:07	14:08	14:09
14:07	14:08	14:09	14:10	14:11	14:12	14:13	14:14	14:15
14:13	14:14	14:15	14:16	14:17	14:18	14:19	14:20	14:21
14:19	14:20	14:21	14:22	14:23	14:24	14:25	14:26	14:27
14:25	14:26	14:27	14:28	14:29	14:30	14:31	14:32	14:33
14:31	14:32	14:33	14:34	14:35	14:36	14:37	14:38	14:39
14:37	14:38	14:39	14:40	14:41	14:42	14:43	14:44	14:45
14:43	14:44	14:45	14:46	14:47	14:48	14:49	14:50	14:51
14:49	14:50	14:51	14:52	14:53	14:54	14:55	14:56	14:57
14:55	14:56	14:57	14:58	14:59	15:00	15:01	15:02	15:03
15:01	15:02	15:03	15:04	15:05	15:06	15:07	15:08	15:09
15:07	15:08	15:09	15:10	15:11	15:12	15:13	15:14	15:15
15:13	15:14	15:15	15:16	15:17	15:18	15:19	15:20	15:21
15:19	15:20	15:21	15:22	15:23	15:24	15:25	15:26	15:27
15:25	15:26	15:27	15:28	15:29	15:30	15:31	15:32	15:33
15:31	15:32	15:33	15:34	15:35	15:36	15:37	15:38	15:39
15:37	15:38	15:39	15:40	15:41	15:42	15:43	15:44	15:45
15:43	15:44	15:45	15:46	15:47	15:48	15:49	15:50	15:51

Westendstr.	Heimeranplatz	Theresienwiese	Hauptbahnhof	Königsplatz	Josephsplatz	Scheidplatz	Petuelring	Olympiazentrum
15:49	15:50	15:51	15:52	15:53	15:54	15:55	15:56	15:57
15:55	15:56	15:57	15:58	15:59	16:00	16:01	16:02	16:03
16:01	16:02	16:03	16:04	16:05	16:06	16:07	16:08	16:09
16:07	16:08	16:09	16:10	16:11	16:12	16:13	16:14	16:15
16:13	16:14	16:15	16:16	16:17	16:18	16:19	16:20	16:21
16:19	16:20	16:21	16:22	16:23	16:24	16:25	16:26	16:27
16:25	16:26	16:27	16:28	16:29	16:30	16:31	16:32	16:33
16:31	16:32	16:33	16:34	16:35	16:36	16:37	16:38	16:39
16:37	16:38	16:39	16:40	16:41	16:42	16:43	16:44	16:45
16:43	16:44	16:45	16:46	16:47	16:48	16:49	16:50	16:51
16:49	16:50	16:51	16:52	16:53	16:54	16:55	16:56	16:57
16:55	16:56	16:57	16:58	16:59	17:00	17:01	17:02	17:03
17:01	17:02	17:03	17:04	17:05	17:06	17:07	17:08	17:09
17:07	17:08	17:09	17:10	17:11	17:12	17:13	17:14	17:15
17:13	17:14	17:15	17:16	17:17	17:18	17:19	17:20	17:21
17:19	17:20	17:21	17:22	17:23	17:24	17:25	17:26	17:27
17:25	17:26	17:27	17:28	17:29	17:30	17:31	17:32	17:33
17:31	17:32	17:33	17:34	17:35	17:36	17:37	17:38	17:39
17:37	17:38	17:39	17:40	17:41	17:42	17:43	17:44	17:45
17:43	17:44	17:45	17:46	17:47	17:48	17:49	17:50	17:51
17:49	17:50	17:51	17:52	17:53	17:54	17:55	17:56	17:57
17:55	17:56	17:57	17:58	17:59	18:00	18:01	18:02	18:03
18:01	18:02	18:03	18:04	18:05	18:06	18:07	18:08	18:09
18:07	18:08	18:09	18:10	18:11	18:12	18:13	18:14	18:15
18:13	18:14	18:15	18:16	18:17	18:18	18:19	18:20	18:21
18:19	18:20	18:21	18:22	18:23	18:24	18:25	18:26	18:27
18:25	18:26	18:27	18:28	18:29	18:30	18:31	18:32	18:33
18:31	18:32	18:33	18:34	18:35	18:36	18:37	18:38	18:39
18:37	18:38	18:39	18:40	18:41	18:42	18:43	18:44	18:45
18:43	18:44	18:45	18:46	18:47	18:48	18:49	18:50	18:51
18:49	18:50	18:51	18:52	18:53	18:54	18:55	18:56	18:57
18:55	18:56	18:57	18:58	18:59	19:00	19:01	19:02	19:03
19:01	19:02	19:03	19:04	19:05	19:06	19:07	19:08	19:09
19:07	19:08	19:09	19:10	19:11	19:12	19:13	19:14	19:15
19:13	19:14	19:15	19:16	19:17	19:18	19:19	19:20	19:21
19:19	19:20	19:21	19:22	19:23	19:24	19:25	19:26	19:27
19:25	19:26	19:27	19:28	19:29	19:30	19:31	19:32	19:33
19:31	19:32	19:33	19:34	19:35	19:36	19:37	19:38	19:39
19:37	19:38	19:39	19:40	19:41	19:42	19:43	19:44	19:45
19:43	19:44	19:45	19:46	19:47	19:48	19:49	19:50	19:51
19:49	19:50	19:51	19:52	19:53	19:54	19:55	19:56	19:57
19:55	19:56	19:57	19:58	19:59	20:00	20:01	20:02	20:03
20:05	20:06	20:07	20:08	20:09	20:10	20:11	20:12	20:13
20:15	20:16	20:17	20:18	20:19	20:20	20:21	20:22	20:23
20:25	20:26	20:27	20:28	20:29	20:30	20:31	20:32	20:33
20:35	20:36	20:37	20:38	20:39	20:40	20:41	20:42	20:43
20:45	20:46	20:47	20:48	20:49	20:50	20:51	20:52	20:53
20:55	20:56	20:57	20:58	20:59	21:00	21:01	21:02	21:03
21:05	21:06	21:07	21:08	21:09	21:10	21:11	21:12	21:13
21:15	21:16	21:17	21:18	21:19	21:20	21:21	21:22	21:23
21:25	21:26	21:27	21:28	21:29	21:30	21:31	21:32	21:33
21:35	21:36	21:37	21:38	21:39	21:40	21:41	21:42	21:43
21:45	21:46	21:47	21:48	21:49	21:50	21:51	21:52	21:53
21:55	21:56	21:57	21:58	21:59	22:00	22:01	22:02	22:03

Westendstr.	Heimeranplatz	Theresienwiese	Hauptbahnhof	Königsplatz	Josephsplatz	Scheidplatz	Petuelring	Olympiazentrum
22:05	22:06	22:07	22:08	22:09	22:10	22:11	22:12	22:13
22:15	22:16	22:17	22:18	22:19	22:20	22:21	22:22	22:23
22:25	22:26	22:27	22:28	22:29	22:30	22:31	22:32	22:33
22:35	22:36	22:37	22:38	22:39	22:40	22:41	22:42	22:43
22:45	22:46	22:47	22:48	22:49	22:50	22:51	22:52	22:53
22:55	22:56	22:57	22:58	22:59	23:00	23:01	23:02	23:03
23:15	23:16	23:17	23:18	23:19	23:20	23:21	23:22	23:23
23:35	23:36	23:37	23:38	23:39	23:40	23:41	23:42	23:43
23:55	23:56	23:57	23:58	23:59	00:00	00:01	00:02	00:03
00:15	00:16	00:17	00:18	00:19	00:20	00:21	00:22	00:23
00:35	00:36	00:37	00:38	00:39	00:40	00:41	00:42	00:43
00:55	00:56	00:57	00:58	00:59	01:00	01:01	01:02	01:03
01:15	01:16	01:17	01:18	01:19	01:20	01:21	01:22	01:23
01:35	01:36	01:37	01:38	01:39	01:40	01:41	01:42	01:43
01:55	01:56	01:57	01:58	01:59	02:00	02:01	02:02	02:03

B.4 Trains

RE 2 — Petershausen → Ostbahnhof

Petershausen	Obermenzing	Laim	Donnersbergerbrücke	Hackerbrücke	Hauptbahnhof	Stachus	Marienplatz	Isartor	Rosenheimerplatz	Ostbahnhof
04:32	04:47	04:50	04:52	04:53	04:54	04:56	04:58	05:01	05:06	05:09
05:05	05:23	05:27	05:30	05:32	05:33	05:35	05:37	05:38	05:40	05:42
06:32	06:47	06:50	06:52	06:53	06:54	06:56	06:58	07:01	07:06	07:09
07:05	07:23	07:27	07:30	07:32	07:33	07:35	07:37	07:38	07:40	07:42
08:32	08:47	08:50	08:52	08:53	08:54	08:56	08:58	09:01	09:06	09:09
09:05	09:23	09:27	09:30	09:32	09:33	09:35	09:37	09:38	09:40	09:42
10:32	10:47	10:50	10:52	10:53	10:54	10:56	10:58	11:01	11:06	11:09
11:05	11:23	11:27	11:30	11:32	11:33	11:35	11:37	11:38	11:40	11:42
12:32	12:47	12:50	12:52	12:53	12:54	12:56	12:58	13:01	13:06	13:09
13:05	13:23	13:27	13:30	13:32	13:33	13:35	13:37	13:38	13:40	13:42
14:32	14:47	14:50	14:52	14:53	14:54	14:56	14:58	15:01	15:06	15:09
15:05	15:23	15:27	15:30	15:32	15:33	15:35	15:37	15:38	15:40	15:42
16:32	16:47	16:50	16:52	16:53	16:54	16:56	16:58	17:01	17:06	17:09
17:05	17:23	17:27	17:30	17:32	17:33	17:35	17:37	17:38	17:40	17:42
18:32	18:47	18:50	18:52	18:53	18:54	18:56	18:58	19:01	19:06	19:09
19:05	19:23	19:27	19:30	19:32	19:33	19:35	19:37	19:38	19:40	19:42
20:32	20:47	20:50	20:52	20:53	20:54	20:56	20:58	21:01	21:06	21:09
21:05	21:23	21:27	21:30	21:32	21:33	21:35	21:37	21:38	21:40	21:42
22:32	22:47	22:50	22:52	22:53	22:54	22:56	22:58	23:01	23:06	23:09
23:05	23:23	23:27	23:30	23:32	23:33	23:35	23:37	23:38	23:40	23:42

RE 2 — Ostbahnhof → Petershausen

Ostbahnhof	Rosenheimerplatz	Isartor	Marienplatz	Stachus	Hauptbahnhof	Hackerbrücke	Donnersbergerbrücke	Laim	Obermenzing	Petershausen
05:17	05:19	05:21	05:22	05:24	05:25	05:26	05:29	05:33	05:37	05:56
06:13	06:16	06:18	06:19	06:20	06:22	06:23	06:26	06:27	06:31	06:59
07:17	07:19	07:21	07:22	07:24	07:25	07:26	07:29	07:33	07:37	07:56
08:13	08:16	08:18	08:19	08:20	08:22	08:23	08:26	08:27	08:31	08:59
09:17	09:19	09:21	09:22	09:24	09:25	09:26	09:29	09:33	09:37	09:56
10:13	10:16	10:18	10:19	10:20	10:22	10:23	10:26	10:27	10:31	10:59
11:17	11:19	11:21	11:22	11:24	11:25	11:26	11:29	11:33	11:37	11:56
12:13	12:16	12:18	12:19	12:20	12:22	12:23	12:26	12:27	12:31	12:59
13:17	13:19	13:21	13:22	13:24	13:25	13:26	13:29	13:33	13:37	13:56
14:13	14:16	14:18	14:19	14:20	14:22	14:23	14:26	14:27	14:31	14:59
15:17	15:19	15:21	15:22	15:24	15:25	15:26	15:29	15:33	15:37	15:56
16:13	16:16	16:18	16:19	16:20	16:22	16:23	16:26	16:27	16:31	16:59
17:17	17:19	17:21	17:22	17:24	17:25	17:26	17:29	17:33	17:37	17:56
18:13	18:16	18:18	18:19	18:20	18:22	18:23	18:26	18:27	18:31	18:59
19:17	19:19	19:21	19:22	19:24	19:25	19:26	19:29	19:33	19:37	19:56
20:13	20:16	20:18	20:19	20:20	20:22	20:23	20:26	20:27	20:31	20:59
21:17	21:19	21:21	21:22	21:24	21:25	21:26	21:29	21:33	21:37	21:56
22:13	22:16	22:18	22:19	22:20	22:22	22:23	22:26	22:27	22:31	22:59
23:17	23:19	23:21	23:22	23:24	23:25	23:26	23:29	23:33	23:37	23:56
00:13	00:16	00:18	00:19	00:20	00:22	00:23	00:26	00:27	00:31	00:59

RE 8 — Feldkirchen → Puchheim

Feldkirchen	Riem	Leuchtenbergring	Ostbahnhof	Marienplatz	Hauptbahnhof	Laim	Pasing	Aubing	Puchheim
04:11	04:21	04:27	04:32	04:35	04:37	04:40	04:46	04:58	05:12
05:09	05:19	05:25	05:30	05:33	05:35	05:38	05:44	05:56	06:10
06:11	06:21	06:27	06:32	06:35	06:37	06:40	06:46	06:58	07:12
07:09	07:19	07:25	07:30	07:33	07:35	07:38	07:44	07:56	08:10
08:11	08:21	08:27	08:32	08:35	08:37	08:40	08:46	08:58	09:12
09:09	09:19	09:25	09:30	09:33	09:35	09:38	09:44	09:56	10:10
10:11	10:21	10:27	10:32	10:35	10:37	10:40	10:46	10:58	11:12
11:09	11:19	11:25	11:30	11:33	11:35	11:38	11:44	11:56	12:10
12:11	12:21	12:27	12:32	12:35	12:37	12:40	12:46	12:58	13:12
13:09	13:19	13:25	13:30	13:33	13:35	13:38	13:44	13:56	14:10
14:11	14:21	14:27	14:32	14:35	14:37	14:40	14:46	14:58	15:12
15:09	15:19	15:25	15:30	15:33	15:35	15:38	15:44	15:56	16:10
16:11	16:21	16:27	16:32	16:35	16:37	16:40	16:46	16:58	17:12
17:09	17:19	17:25	17:30	17:33	17:35	17:38	17:44	17:56	18:10
18:11	18:21	18:27	18:32	18:35	18:37	18:40	18:46	18:58	19:12
19:09	19:19	19:25	19:30	19:33	19:35	19:38	19:44	19:56	20:10
20:11	20:21	20:27	20:32	20:35	20:37	20:40	20:46	20:58	21:12
21:09	21:19	21:25	21:30	21:33	21:35	21:38	21:44	21:56	22:10
22:11	22:21	22:27	22:32	22:35	22:37	22:40	22:46	22:58	23:12
23:09	23:19	23:25	23:30	23:33	23:35	23:38	23:44	23:56	00:10
00:11	00:21	00:27	00:32	00:35	00:37	00:40	00:46	00:58	01:12
01:09	01:19								

RE 8 — Puchheim → Feldkirchen

Puchheim	Aubing	Pasing	Laim	Hauptbahnhof	Marienplatz	Ostbahnhof	Leuchtenbergring	Riem	Feldkirchen
04:18	04:32	04:40	04:46	04:49	04:51	04:54	04:59	05:05	05:14
05:12	05:26	05:34	05:40	05:43	05:45	05:48	05:53	05:59	06:08
06:18	06:32	06:40	06:46	06:49	06:51	06:54	06:59	07:05	07:14
07:12	07:26	07:34	07:40	07:43	07:45	07:48	07:53	07:59	08:08
08:18	08:32	08:40	08:46	08:49	08:51	08:54	08:59	09:05	09:14
09:12	09:26	09:34	09:40	09:43	09:45	09:48	09:53	09:59	10:08
10:18	10:32	10:40	10:46	10:49	10:51	10:54	10:59	11:05	11:14
11:12	11:26	11:34	11:40	11:43	11:45	11:48	11:53	11:59	12:08
12:18	12:32	12:40	12:46	12:49	12:51	12:54	12:59	13:05	13:14
13:12	13:26	13:34	13:40	13:43	13:45	13:48	13:53	13:59	14:08
14:18	14:32	14:40	14:46	14:49	14:51	14:54	14:59	15:05	15:14
15:12	15:26	15:34	15:40	15:43	15:45	15:48	15:53	15:59	16:08
16:18	16:32	16:40	16:46	16:49	16:51	16:54	16:59	17:05	17:14
17:12	17:26	17:34	17:40	17:43	17:45	17:48	17:53	17:59	18:08
18:18	18:32	18:40	18:46	18:49	18:51	18:54	18:59	19:05	19:14
19:12	19:26	19:34	19:40	19:43	19:45	19:48	19:53	19:59	20:08
20:18	20:32	20:40	20:46	20:49	20:51	20:54	20:59	21:05	21:14
21:12	21:26	21:34	21:40	21:43	21:45	21:48	21:53	21:59	22:08
22:18	22:32	22:40	22:46	22:49	22:51	22:54	22:59	23:05	23:14
23:12	23:26	23:34	23:40	23:43	23:45	23:48	23:53	23:59	00:08
00:18	00:32	00:40	00:46	00:49	00:51	00:54	00:59	01:05	01:14

APPENDIX C
SOURCE CODE

Source models, transformed models an resulting generated source code as well as hand written completion of the project will be available on-line, however, this will not be before summer 2012. Refer to:

http://www.activecharts.de
http://www.uni-ulm.de/en/in/pm/research/projects/activecharts.html
http://www.gessenharter.com/activecharts

Please contact the author for any further assistance, examples, resources etc.:

dominik@gessenahrter.com

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Betz, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

2011-01 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Flexibilisierung Service-orientierter Architekturen

2011-02 *Johannes Hanika, Holger Dammertz, Hendrik Lensch*
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 *Stefanie Kaiser, Manfred Reichert*
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 *Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):*
Statistical Computing 2011 - Abstracts der 43. Arbeitstagung

2011-05 *Vera Künzle, Manfred Reichert*
PHILharmonicFlows: Research and Design Methodology

2011-06 *David Knuplesch, Manfred Reichert*
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 *Dausend, Marcel*
Towards a UML-Profile on Formal Semantics for Modelling Multimodal Interactive Systems

2011-08 *Gessenharter, Dominik*
Model-Driven Software Development with ACTIVECHARTS - A Case Study

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm