Adaptive Rule-Based Infrastructure Modelling

Bernhard Seybold
trafIT solutions gmbh, Heinrichstrasse 48, CH-8005 Zürich
seybold@trafit.ch

Alexander Kuckelberg
VIA Consulting & Development GmbH, Frankenberger Str. 30, DE-52066 Aachen
a.kuckelberg@via-con.de

Abstract
In this paper, we present an adaptive, rule-based modelling approach for railway infrastructure representation. While currently railway operation tools usually require a complete and consistent set of infrastructure objects, our rule-based approach widens these limitations. The approach follows a functional-oriented modelling approach, where the usage and characteristic of infrastructure is the key aspect in contrast to the representation of objects and references between them. The adaptive, rule-based approach offers several advantages, especially with respect to heterogeneity of granularity and completeness requirements. Also, it is well suited to express temporal changes of infrastructure and we show that the approach is efficient even for large rule systems.

Keywords
Railway Infrastructure, Modelling, Rule-Based, Logic Evaluation, Efficient Algorithm

1 Introduction

Different railway operation tools and applications use quite specific and more or less individual infrastructure models. Keywords like microscopic, mesoscopic or macroscopic infrastructure models are usually mentioned in this context with a commonly accepted idea about the characteristics of these granularity levels and how they are distinguished [5][6] [10]. The infrastructure model for a specific tool is usually chosen due to functional needs, legacy conditions or available specifications and standards. Microscopic simulation, timetable construction and analytic tools like e.g. *LUKS* [3] require detailed knowledge about topology, available train protection systems, gradients or stopping positions to compute travel and occupation times to the second. In contrast, macroscopic tools require more abstract data like stations, available routes or minimum headway times [4][8][12].

Generally, current infrastructure modelling approaches follow an object-based approach, which transforms the topology and characteristics of infrastructure into comparable data and object structures. This leads to a graph model, which normally represents the topology one-to-one. In a microscopic world, switches or crossings correspond to nodes and connecting tracks to edges, subdivided by signals, gradients or stopping positions as inner nodes if needed. Macroscopic views may consider stations, marshalling yards or control points as nodes and routes as connecting edged. Additional operational information can be assigned to nodes or edges.

However the infrastructure detail level is selected for a specific application field and tool, the data used has to match and has to be consistent within the selected data granularity level.

Microscopic tools require the detailed information like gradient, speed profile, interlocking equipment, topology and stopping position information to enable a detailed running and occupation time calculation. With these computations additional actions and algorithms like rescheduling, dispatching or capacity evaluation with analytical methods or simulation become possible. Macroscopic tools might abstract this information to operation points, links between them and/or headway times. The information is usually mapped to objects with attributes and relations between them.

The data model defines a rather static framework for the information. Partial definition or partial provision of information is typically not intended or possible. Moreover, looking into microscopic data related specifications like *railML* [13] or *XML-ISS* and *XML-KSS* [1] shows, that data require a high information completeness and consistency, because small changes usually have strong influence on computational results, e.g. a missing or misplaced liberation point. This makes it difficult and time-consuming to provide consistent and complete (base) data. And it is often a barrier when starting new projects. The step to feed systems with a sufficient minimal set of data often requires a lot of work that should and cannot remain unconsidered within project management.

In our approach, we replace the explicit representation of infrastructure and its topology by representation of its properties and functional characteristics by rules. Railway operation tasks are consequently transferred to rule evaluation; queries are answered by evaluating these rules.

As it has turned out, changing the infrastructure modelling approach implied other problems, especially performance problems when the number of rules rises. Therefore, in addition to the rule-based modelling itself, some specific optimization aspects are presented in the following as well.

The paper is structured as follows: in Chapter 2, we discuss variants of infrastructure modelling. Our rule system is introduced in Chapter 3, its evaluation is analysed and improved in Chapter 4, and its practicability shown in Chapter 5. A short discussion in Chapter 6 rounds up the paper.

2 Infrastructure Modelling

The infrastructure model we use follows a rule-based approach. In contrast to the commonly used object-oriented approaches, where a more or less complete data set consistent within its modelling level has to be provided, our approach allows to define properties and characteristics of infrastructure as fare as required (or available) instead of transferring the topology into object structures. Properties are represented by rules and stored within a repository. This repository can be queried about well-defined properties, for instance the headway time, the available tracks or the connection time between two trains. Each query has a certain format that needs to be obeyed.

Internally, the repository is set up as a rule-system. For each of the well-defined queries, there is an ordered list of rules that provide values for this particular case.

Rules consist of two parts: (1) a left-hand side built of selectors with which it can be determined whether the rule applies for the query and (2) a right-hand side of values that are the answer to the query. Values could be values of any type or even vectors of values.

To get a better impression of the rule-based approach a figure might clarify the ideas (Figure 1).

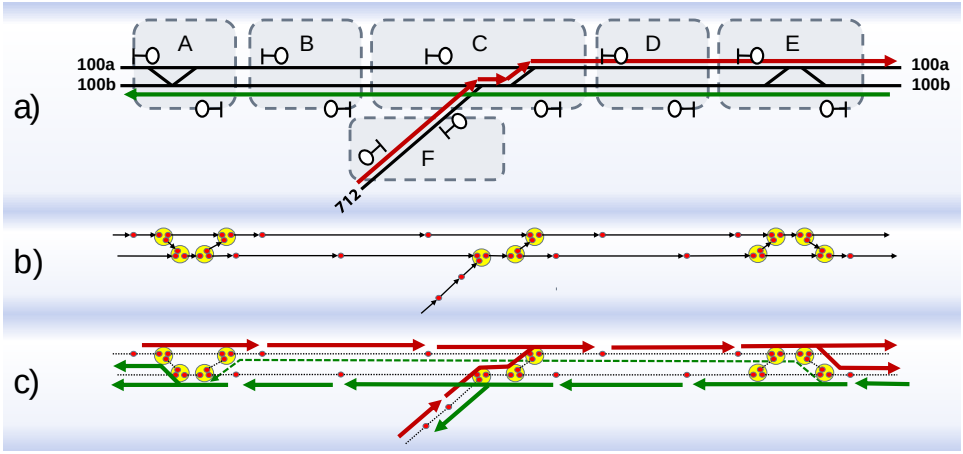

Figure 1: Infrastructure Example

While a) shows a short example of infrastructure with some control points (*A/B/C/D/E/F*), switches, signals, a double track line (*100*), a branching track (*712*) and two train runs (*F/C/D/E* and *E/D/C/B/A*) the first abstraction level b) shows the transformation into node objects and relations between them. The approach shown here is the one e.g. specified by XML-ISS [1] or used by the railway operation tool *LUKS*. It becomes clear, that the topology directly implies the objects and their relationships. As a further level c) a logical layer is added to represent the usability of the infrastructure. In this example, several routes are shown by the arrows connecting signal objects and defining the sequence of objects (elements) covered by the routes. The dashed route is a possible route on the wrong side track without blocking signals. Train runs like shown in a) are finally defined as a sequence of matching routes. With these different layers, train runs, running and occupation times are determined and conflicts are detected by the railway operation tools based on such data.

With rules the topology itself can be faded-out. For running times, train runs and headway times only the characteristics of the infrastructure are relevant and therefore modelled by rules.

To model e.g. that there is a route for trains from track *F-712* to track *D-100a* passing *C* a rule like *route(F-712* $\rightarrow$ *C* $\rightarrow$ *100a-D)* can be defined. Assuming that also the second track within *D* can be reached by trains coming from *F*, the rule *route(F-712* $\rightarrow$ *C* $\rightarrow$ *100b-D)* is true as well. These are all possibilities, to pass *C* from *F* and reach any track in *D*. Therefore these two rules might be generalized using a general selector □ representing any possible value: *route(F-□* $\rightarrow$ *C* $\rightarrow$ *□-D)* expresses, that all tracks in *D* can be reached from all tracks in *F* when passing *C*. In this way, the universal selector can be used to specify most general information and knowledge about infrastructure topology.

The exclusion of routes as they can be derived from the object approach can also be expressed by rules. As an example the rule *routeHeadway(F-712* $\rightarrow$ *C* $\rightarrow$ *100a-D, D-100b* $\rightarrow$ *C* $\rightarrow$ *100b-B, 180)* might tell the system, that trains from *F-712* to *D-100a* passing *C* prohibit the passage of any other train from *D-100b* to *B-100b* for 180 seconds.

Instead of specifying all possible route exclusions and headway times, the general selector can be used here, too, to define elementary knowledge, which is valid as long as

no more detailed rule exists. The rule *routeHeadway(*□-□ → □ → □-□,□-□ → □ → □, *240)* e.g. specifies, that a minimum headway time of four minutes should be used as long as no more detailed information is available. On the other hand, with this single rule an elementary knowledge base is set up and all queries querying headway times can be evaluated on a defined data basis. This illustrates the adaptive character of our rule-based approach.

Representing detailed information on the other hand leads to a huge number of rules, which decrease the evaluation and querying performance and requires additional performance optimization considerations.

3 Rules Syntax and Semantics

The following chapters introduce the rules, their syntax and formalism more detailed and give more examples how to use them and how to optimize rule evaluation.

First, we define some basic terms:

- A selector *S* for a type *T* is an operator $S: T \rightarrow \{true, false\}$
- A rule system consists of *n* rules $r_0 .. r_{n-1}$
- Each rule r_i consists of *m* selectors S_{ij} and S_{ij} is a selector of type T_j
- A concrete query is a vector of query elements $q = [q_0 .. q_{m-1}]$, such that q_j is of the type T_j
- A selector matches a query if all selectors match the corresponding query element.
- The result of the query is the matching rule with the highest order index.

The rules are ordered such that specific rules follow general rules. This sequence supports a top-down refinement by appending more specific rules at the end.

3.1 Types

We introduce a type system with the following types:

Table 1: Type System

Type	Attributes	Example
Train	number	123
	id	123_17
	type	IC
OperationPoint	id (UIC-code followed by national code)	85ZUE
	code (national code)	ZUE
	UIC-code (two-digit country code)	85
Section	id	100
Track	id	5
DayTime	---	12:00:00
Date	---	20.12.2012

DayTime and Date are value-types and have no attributes. The type set is just a basic set, types and attributes may easily be added as required.

Instances are represented e.g. as *train(123, 123_17, IC)*, which represents a train with number=123, id=123_17 and type=IC.

3.2 Selectors

Rules are built of selectors. Abstractly, a selector describes a set of instances of a certain type. Typically, this is done by requiring a certain instance attribute to have a certain value.

The most general selector used for all types is the wildcard (written as □). It matches every object.

A selector references an attribute of the type or the value of the type itself.

The first attribute is the default for the type and we may leave it out to increase compactness, thus instead of "number=123" we just write "123".

Additionally, since all attribute types have an underlying order, ranges are supported.

Table 2: Selector Syntax

trainSelector	:= ["id="] stringValue | "number=" stringValue | "type=" stringValue | wildcard
operPointSelector	:= ["id="] stringValue | "code=" stringValue | "UIC-code=" integerValue | wildcard
sectionSelector	:= ["id="] stringValue | wildcard
trackSelector	:= ["id="] stringValue | wildcard
dayTimeSelector	:= hours [" :" minutes [":" seconds]] | wildcard
dateSelector	:= day "." month "." year | wildcard
stringValue	:= string | ".." string | string ".." | string ".." string
integerValue	:= integer | ".." integer | integer ".." | integer ".." integer
uicCodeValue	:= uicCode | ".." uicCode | uicCode ".." | uicCode ".." uicCode
wildcard	:= "□"
hours	:= integer
minutes	:= 0..59
seconds	:= 0..59
day	:= 1..31
month	:= 1..12
year	:= integer
uicCode	:= 10..99

Following this syntax, some examples for train selectors would be written as:

number=123
type=EC
id=123_17

Possible operation point selectors could be:

code=FF
UIC-code=85
id=85FF

Examples of Day Time selectors:

19:30
08:00..09:00

3.3 Rules

Rules have a left-hand side (selectors) and a right-hand side (values). Each of the selectors decides on a particular property of the query that is evaluated. If this is the case, the selector matches. If all selectors of the rule match, the rule matches and the right-hand side is returned as value for further processing. The right-hand side could be one value or

an ordered set of values depending on the design of the rule.

The matching operator is defined as follows:

$$\otimes : S \times q \rightarrow \{true, false\} \quad (1)$$

A rule matches, if all its selectors match their corresponding query element. Among the rules matching the query, the value of the one with the highest rule order index is the result of the query. If there is no matching rule, □ is returned. However, we encourage the use of a default rule to avoid to model default behaviour in the code.

3.4 Example: Transfer Time

As an extended example, we model transfer time dependent of the following elements: The date, the time of day, the operation point the transfer occurs, and the tracks the feeder and waiter trains stop at. This gives us the following selector columns with their corresponding types in parenthesis:

```
date (Date)
dayTime (DayTime)
operationPoint (OperationPoint)
feederTrack (Track)
waiterTrack (Track).
```

The right-hand side is the waiting time in seconds. It turns out to be a good strategy to always use ISO units to avoid unit conversion issues.

At the beginning we start with a default 3 minute (180 seconds) transfer time, so we add the first rule (rule indices start at 0):

```
r0 = □, □, □, □, □                    → 180
```

This rule assures that for every query there is always a well-defined result. Now, the Zurich main station (ZUE) is a bit larger and transfers take longer, so we add the rule

```
r1 = □, □, ZUE, □, □                  → 300
```

But if the tracks belong to the same platform, the time can be reduced, thus we add some more rules for neighbour tracks

```
r2 = □, □, ZUE, 5, 6                  → 80
r3 = □, □, ZUE, 7, 8                  → 80
r4 = □, □, ZUE, 9, 10                 → 80
```

Now, we evaluate some example queries against this rule-set and get the following results:

```
q(09.12.2012, 10:00, op(ZUE), track(5), track(7))     →  300
q(01.1.2050, 12:00, op(ZUE), track(5), track(6))      →  80
q(20.12.2012, 08:00, op(FF), track(2), track(3))      →  180
```

In addition, we could also model that the transfer time during rush hour are slightly

higher (or are they in fact lower since everybody is in a hurry?). The following rules may be added:

```
r5 = □, 6:30..9:00, ZUE, □, □              → 360
r6 = □, 16:30..18:00, ZUE, □, □            → 360
```

Finally, the Zurich main station is currently under construction for the new "Durchmesserlinie". During some project phases, transfer times are slightly longer for some tracks. This fact could be modelled with the following rules:

```
r7 = 1.1.2011..31.5.2011, □, ZUE, 7, □     → 360
r8 = 1.1.2011..31.5.2011, □, ZUE, □, 7     → 360
```

Of course, these rules are just basic examples and they should also be combined in a suitable way. However, we want to demonstrate how a rule system can dynamically be adapted for changing requirements and how a simple model can evolve and represent a lot of refined details when needed.

4 Rule Evaluation

Rules have an external representation such that they can be edited easily. We chose to use CSV files since they can be edited with Excel or a lot of text editors. For identification, the CSV file starts with two comment lines containing the file type and the column headers.

```
#TransferTimes
#Feeder;Waiter;OperationPoint;FeederTrack;WaiterTrack;TransferTime
;;;;;180
;;ZUE;;;300
```

The evaluation of a rule system resembles the evaluation of a logical formula. Basically, it is a disjunction of a conjunction. To evaluate it efficiently, we apply well-known techniques used in evaluating Boolean formulas, see for instance [7][11]. In the next Section, we discuss which algorithmic techniques have been applied and we analyse their complexity with respect to the direct approach.

4.1 Average Complexity

The complexity of query evaluation is measured in number of matching operations per query. With n rules and m selectors, the worst case complexity (always the last selector fails) is $n \cdot m$, and the minimal complexity is m (all columns of one rule checked).

For analysis, we use probability theory: The a-priori probability that a selector in S_{ij} matches is noted as p_j. The expected number of checks until selector S_{ij} matches is p_j^{-1}. Then, the notation for the expected number of checks with given probability vector p is $Ec(p)$.

The formula for $Ec(p)$ is developed recursively. If there are no columns, no checks have to be done. Otherwise the number of checks is the expected number of checks in the columns but the last one, plus one in the last column multiplied by the expected number of times until the selector matches. In mathematical notation:

$$Ec([]) = 0$$
$$Ec([p_0..p_{m-1}]) = p_{m-1}^{-1}(1 + Ec([p_0..p_{m-2}])) \quad (2)$$

This can be reformulated in a closed form as follows.

$$Ec([p_0..p_{m-1}]) = \sum_{0 \le j < m} \prod_{j \le k < m} p_k^{-1} \quad (3)$$

Concrete values for one, two and three selector columns:

$$Ec([p_0]) = p_0^{-1}$$
$$Ec([p_0, p_1]) = (p_0 p_1)^{-1} + p_1^{-1} \quad (4)$$
$$Ec([p_0, p_1, p_2]) = (p_0 p_1 p_2)^{-1} + (p_1 p_2)^{-1} + p_2^{-1}$$

Example:
Let p be *[0.2, 0.05, 0.1]*. Then *Ec(p) = 5·20·10 + 20·10 + 10 = 1210* checks.

Interestingly, the expected number of checks is not directly dependent on the number of rules. On the other hand, however, this also means that with given and fixed probabilities, there is only a limited variability of expressions that can be formulated. For instance, if all p=1, then there is basically only one rule that makes sense covering all queries. We call this number the expressiveness. The expressiveness of a rule with one column with probability p is p^{-1}. The expressiveness of rules with more columns is the product of expressiveness of the columns.

In the following, we consider two strategies to increase the efficiency of the query evaluation: column sorting and pivot evaluation.

4.2 Column Sorting

Observation 1: For the semantic, it does not matter in which order the selectors of a rule are evaluated. However, it does matter for complexity.

Due to short-circuit evaluation, selectors in left columns are evaluated more often than those further on the right. Since the order of columns does not change the outcome of the result, it is, hence, advantageous to sort the columns ascending according to p. In our example from above, sorting the columns would produce the new probability vector *[0.05, 0.1, 0.2]* and reduce the number of checks to 20·10·5 + 10·5 + 5 = 1055 instead of 1210 checks originally.

Column-sorting does not change the most significant term of the complexity but influences the other terms positively with no disadvantage. It should therefore always be included in the evaluation algorithm.

4.3 Pivot Evaluation

Observation 2: A lot of rules share the same selectors (see example in Section 3.4)

This yields a strategy to partition rules by their selector in the pivot column and to test them together. For each selector in the pivot column, collect all lines that have this selector in the pivot column, store highest line number.

The query is evaluated as follows: for each value of the pivot column, test the pivot selector. If it matches, then evaluate the remaining columns with the traditional evaluation strategy.

Evaluation of a query: for each value in the pivot column: if query matches, evaluate lines of this selector with the regular algorithm until a match is found or until line number is smaller than best line number. However, we can remember the previously best (highest) line number and break the evaluation for smaller line numbers.

The resulting complexity (noted as *Ep(p)*) is as follows: we need to check all elements of the pivot columns (this amount is noted as x_0) and, for all positive matches, we need to evaluate the lines belonging to the pivot. Since the expected number of checks is not dependent on the number of the lines, we can expect the same number of checks with one column removed.

$$Ep([p_0..p_{m-1}]) = x_0 + p_0 x_0 Ec([p_1..p_{m-1}]) \tag{5}$$

By this approach, we save the following number of checks:

$$\begin{aligned} &Ec(p) - Ep(p) = \\ &\prod_j p_j^{-1} + Ec([p_1..p_{m-1}]) - (x_0 + p_0 x_0 Ec([p_1..p_{m-1}])) = \\ &\prod p_j^{-1} - x_0 - (1 - p_0 x_0) Ec([p_1..p_{m-1}]) \cong \\ &\prod p_j^{-1} - x_0 \cong \\ &\prod p_j^{-1} - p_0^{-1} \end{aligned} \tag{6}$$

In order to save time analysing whether two selector terms are equal (in general cases this is a hard problem), we just consider two selectors to be equal if there textual representation is equal. In this way, we may lose some equal pairs, e.g. “id=123” and “123”. However, with an internal normalization step (e.g. add default attributes), we soon arrive at a very decent equality checking.

5 Examples

We compare four strategies: regular short-circuit evaluation and pivot evaluation both with and without column sorting. To see the full range of sorting effects, the probabilities are given in the worst possible order. The size of partitions for pivot evaluation is estimated to be $1/p_0$. In addition, we indicate the expressiveness N.

Table 3: Strategy Comparison

Columns (p_i)	N	Regular	RegularSort	Pivot	PivotSort
0.5, 0.1, 0.02	1000	1,550	1,022	552	72
0.5, 0.5, 0.5	8	14	14	8	8
0.1, 0.1, 0.1, 0.1, 0.1	100,000	111,110	111,110	11,120	11,120
0.5, 0.1, 0.1, 0.1, 0.1	20,000	31,110	22,222	11,112	2,232
0.04, 0.02, 0.01	125,000	130,100	126,275	5,125	1,375

The example in the first row contains three columns with probabilities 0.5, 0.1 and 0.02 respectively. This yields an expressiveness of 1000 (1 / 0.5 + 1 / 0.1 + 1 / 0.02). With the regular strategy, 1,550 checks are used. Sorting decreases this to 1,022 (the order is the input reversed). With the Pivot strategy (and column 3 as pivot column), the number of checks is halved to 552. PivotSort is the best approach with just 72 checks.

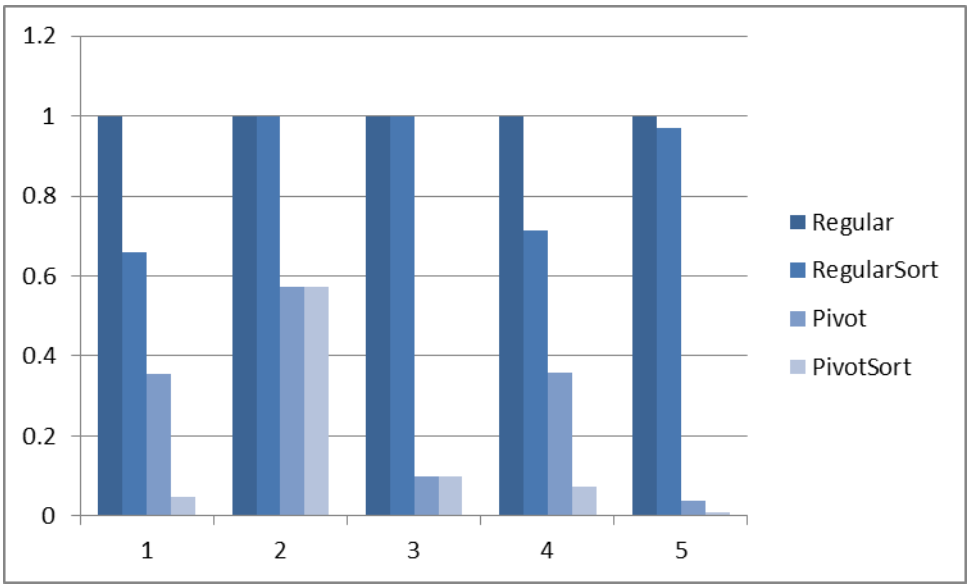

Figure 2: Effectiveness of Strategies

Figure 2 shows the number of checks relative to the regular strategy for the examples above. We see that, depending on the columns and their probabilities, that both pivot and sorting are effective strategies reducing the amount of checks significantly.

5.1 Real-World Example

The tool *OnTime* [2][9] is an application to investigate timetable quality. For this, the timetable is evaluated against a mesoscopic infrastructure model. Since we are not interested in the topology or infrastructure objects but in their attributes the infrastructure is modelled with numerous rules. More than 20 different properties model various aspects of the considered infrastructure, e.g. headway times, primary delays, passing and crossing opportunities, etc. Some of those files are extensively used with more than 100,000 rules generated from external sources in the example below.

The computation in *OnTime* is done in two steps: an activity graph is built first before the graph is evaluated. More details on the algorithm can be found in [2]. For the first step, the rule systems are evaluated extensively.

In real-world examples, the probabilities of selector evaluation are not known explicitly. Therefore, an estimate is needed. One possibility would be to estimate the probability based on the type. Typically, there are more different trains than there tracks, thus train columns would be ordered left of track columns. An even better approach is to count the number of different strings per column and to order columns to the left where more different selectors occur. By this, we benefit in cases where for instance a train column often contains only wildcards. This strategy is used in our application and string counting is used for both column sorting and pivot determination.

In order to demonstrate the order of magnitude, we took a regular example of one day of operation in Switzerland with around 8,000 train runs. The infrastructure is modelled as rule systems with a total of 307,424 rules. During graph configuration, 7,895,229 queries are run against these systems, of which 51% yield no result meaning that the entire rule set had to be evaluated.

Table 4: Strategy Comparison in a Real-World Example

Example SBB-110714

Rules: 307,424
Queries: 7,895,229 (of which 3,988,451 (51%) yield no result (all rules checked))

Strategy	**Checks**		**Time [s]**	
Regular	51,276,012,322	100.0%	1969	100.0%
RegularSort	20,163,489,205	39.3%	704	35.7%
Pivot	1,633,123,716	3.2%	43	2.2%
PivotSort	1,283,644,270	2.5%	30	1.5%

We can easily see that with such large rule systems and such a high number of queries, the efficiency of *OnTime* is heavily dependent on the performance of the rule systems. Here, sorting and especially the pivot strategy are the keys to an efficient rule system.

6 Infrastructure Modelling Comparison and Discussion

In the last chapters, the adaptive rule-based infrastructure model was introduced, some examples were shown and performance evaluation issues were discussed. A rough introduction was given to the rule-based approach. This approach differs from the object-oriented infrastructure modelling approaches in several ways.

First of all, it obviously does not reflect the topology itself but concentrates on (operational) characteristics and properties of the infrastructure. In this way, the perspective on infrastructure shifts from topology representation towards infrastructure functionality and operational requirements. The choice of rules consequently supports an elegant and compact representation of such properties, which in object-oriented approaches are usually derived from (multiple) objects, their relationships and attributes.

Another, yet unconsidered aspect of the rule-based approach is the versioning of infrastructure. With the object-oriented approach topology and objects have to be versioned, validities have to be assigned and evaluated. A complex problem is topology change, where object relationships vary over time periods and network structures are

modified. With the rule-based approach this versioning can be integrated easily and in a consistent way by assigning additional (general) selectors to rules, which allows remaining consistently in the one and only evaluation system. In this way, time or date windows are easily definable whereas with object-oriented approaches specific filtering and restructuring functionalities have to be included.

But maybe the greatest advantage of the rule-based approach is the support of information modelling in a top-down manner, where quite general information can be provided as a first base of information and can be refined as soon as more detailed information are required or available. In contrast, the object-oriented approach requires a certain base data set, the consistency must be guaranteed, which makes the infrastructure definition a usually time-consuming task within each project.

The rule-based approach still offers some more advantages, which are shortly mentioned here: From a structural point of view, all rules follow the same data paradigm, rule solving systems are stable, available and optimized and a wide variety of powerful tools are available. Object-oriented approaches usually follow their own data model; functionality is especially implemented for the specific structures.

This unified structural simplicity of rule-based approaches moreover allows an easy data management, the efficient organization and optimized access of rules and a content driven optimization independently from the concrete data and rule structures chosen by a user.

But also some disadvantages of the rule-based approach should be stated here. If the structure, the topology of the infrastructure itself is an important aspect of the system – e.g. when positioning switches, signals, tracks or stopping positions is relevant – sometimes the spatial impression is much better representable by objects and their relationships. The lack of an explicit object list of infrastructure elements is just one concrete point that can be considered as a big disadvantage of a rule-based approach in some situations.

Continuing this aspect, it must be stated, that an explicit topology oriented modelling with rules is possible, if desired, but is at least as complex as with object-oriented approaches. Therefore the rule-based approach is quite efficient and elegant, whenever infrastructure behaviours and properties are important for the application using the infrastructure model but might imply extensive and confusing rule systems, whenever the topology is in the applications focus. For such applications, a topology representing object-oriented approach is more promising.

For future developments and from experiences we gained with the different infrastructure modelling worlds of our commercial tools – *LUKS* and *OnTime* – a combination of the object-oriented and the adaptive rule-based approach is promising. Such hybrid systems might offer easy starting condition for new projects as well as detailed evaluation possibilities for extended projects.

References

[1] Brünger, O., Gröger, T., Fahrplantrassen managen und Fahrplanerstellung simulieren, Proceedings der 19. Verkehrswissenschaftlichen Tage (VWT), Dresden.

[2] Büker, Th., Seybold, B., Stochastic Modelling Of Delay Propagation In Large Networks, Journal of Rail Transport Planning & Management 2 (2012), pp. 34-50.

[3] Janecek, D., Kuckelberg, A., Nießen, N., Kapazitätsermittlung von Eisenbahnknoten und -strecken, ETR 10 /2012, pp. 30-37.

[4] Kettner, M., Sewcyk, B., Eickmann, C., Integrating microscopic and macroscopic models for railway network evaluation, European Transport Conference 2003.
[5] Kuckelberg, A., Mikroskopische Disposition spurgebundener Verkehrsmittel unter Echtzeitbedingungen, thesis work, RWTH Aachen, 2011.
[6] Kuckelberg, A.; Hoffmann, M., Computer-aided capacity management life-cycle support – In: Proc. of the 12th World Conference on Transport Research (WCTR), 2010, Lissabon.
[7] Lloyd, J.W., Foundations of logic programming, Springer Verlag, 1987.
[8] Marinov, M., Viegas, J., A mesoscopic simulation modelling methodology for analyzing and evaluating freight train operations in a rail network. Simulation Modelling Practice and Theory, 2011.
[9] OnTime (http://www.ontime-rail.com).
[10] Pachl, J. (eds.), Railway Timetable & Traffic. Analysis - Modelling - Simulation. Eurailpress 2008, 228 ., ISBN 978-3-7771-0371-6.
[11] Paulson, L.C., Smith, A.W., Logic programming, functional programming, and inductive definitions, LNCS 475, 1991, pp 283-309.
[12] Quaglietta E., Punzo V., Montella B., Nardone R., Mazzocca N., Towards a hybrid mesoscopic-microscopic railway simulation model, 2nd International Conference on Models and Technologies for ITS, June 2011, Leuven.
[13] railML (http://www.railml.org/).