

Using Model Transformation for Generating Visualizations from Repository Contents

An Application to Software Cartography

Technical Report TB 0601

Alexander M. Ernst, Josef Lankes, Christian M. Schweda, André Wittenburg

Software Engineering for Business Information Systems (sebis)
Ernst Denert-Stiftungslehrstuhl
Chair for Informatics 19
Technische Universität München

Boltzmannstraße 3, 85748 Garching b. München, Germany

{ernst, lankes, schweda, wittenbu}@in.tum.de

November 2006

Abstract

Documenting application landscapes, a central part of Enterprise Architecture Management, heavily relies on graphical and intuitive visualizations. Planning, deciding, and controlling in respect to measures on the application landscape all benefit from such diagrams.

In our research project *Software Cartography* we have analyzed a multitude of visualizations, originating from several major industrial users. Thereby, we found that the majority of diagrams used is created manually, a tendency we could confirm in a survey on Enterprise Architecture Management Tools. We regard the manual creation of these visualizations leading to significant disadvantages, mainly due to reasons of creation effort and modeling rigor. Therefore, we view the availability of an automatic process for generating well-formed diagrams as essential to Enterprise Architecture Management.

This work presents an approach for leveraging model transformations to automatically generate graphical representations of application landscapes. An object oriented model for describing these visualizations is presented, which is augmented with formal definitions forming a basis for automated diagram layouting. We put this findings into practice and describe an exemplary layouting approach employing a genetic algorithm solving an optimization problem. Thus, we want to make a contribution to the visualization in Enterprise Architecture Management.

0	Preface	1
1	Motivation: Visualizing Application Landscapes	2
2	Generating Software Maps using Model Transformation Techniques	5
2.1	Relationship between Semantic and Symbolic Models	6
2.2	Model Transformation and Software Maps	9
2.2.1	Leveraging Model Transformation for Generating Software Maps . . .	10
2.2.2	Introducing a Visualization Model	11
2.2.3	Introducing Transformation Rules	13
3	Software Map Types	15
3.1	Software maps with a base map for positioning	15
3.1.1	Cluster Map	16
3.1.2	Process Support Map	17
3.1.3	Interval Map	18
3.2	Software maps without a base map for positioning	19
3.3	The Layering Principle	19
4	Visualization Model	21
4.1	Object-oriented Visualization Model	23
4.1.1	Map Symbols	25
4.1.2	Visualization Rules	27
4.2	General Formalization of the Visualization Model	33
4.2.1	Prerequisites	33
4.2.2	Visualization Objects - Definition	34
4.2.3	Map Symbols - Definition	37
4.2.4	Map Symbols - Auxiliary concepts	40
4.2.5	Map Symbols - Examples	42
4.2.6	Visualization Rules - Definition	44
4.2.7	Visualization Rules - Examples	46
4.3	Mitigations of the Visualization model	51

4.3.1	Mitigations of planar map symbols	52
4.3.2	Mitigations of linear map symbols	57
4.4	Concretization of the Visualization Model	57
4.4.1	Concretizations for planar map symbols	58
4.4.2	Concretizations for linear map symbols	62
5	Related Work	66
6	Outlook	68
A	Planar map symbol area homomorphism	71
B	Multiplicity values formalized	73
C	From symbolic model to optimization problem	76

CHAPTER 0

Preface

The research project *Software Cartography*, which started in February 2003, establishes methods and models for documenting, evaluating, and planning application landscapes and especially defines notations as well as visualization techniques utilizing concepts from the field of conventional cartography [HGM02].

Sponsors and partners of the research project are AMB Generali, Allianz, AXA, BMW Group, Deutsche Börse Systems, Deutsche Post, Ernst Denert-Stiftung, Kühne + Nagel, HVB Systems Information Services, Münchener Rück, sd&m, Siemens, Siemens Business Services, T-Com, and TUI.

Thanks to our sponsors and partners!

Motivation: Visualizing Application Landscapes

Application landscapes are complex systems consisting of hundreds or even thousands of business application systems. They are a major investment of modern enterprises and are subsequently changed to support new or changed business goals, business strategies, and business processes. The documentation, evaluation, and planning of these application landscapes are major challenges of enterprises, especially when seen in the context of increasing the *alignment of business and information technology*.

In the research project *Software Cartography*, we made an initial analysis on different types of visualizations and interviewed various stakeholders about their concerns on the application landscape [MW04, LMW05c]. One of the findings was, that nowadays many of the visualizations are created manually by positioning symbols on a canvas. This is a task, which is both time consuming and error-prone, even if a repository based tool ensures data consistency with the actual information about the application landscape. Our approach is meant to support automatic generation of such visualizations.

The absence of a standard for modeling application landscape as well as the diversity of concerns expressed by the stakeholders of the application landscape has lead to a high number of different visualization types, varying in the information displayed therein and in the concepts employed for displaying. The analysis further showed that a certain kind of flexibility regarding the usage of different symbols is necessary, especially if the visualizations should be adapted to suite the needs of stakeholders interested in highlighting aspects of the application landscape. This flexibility, which is of course available when creating visualizations manually, should not be pruned by automatic generation.

To give a short introduction on the visualizations this report focuses on, Figure 1.1 shows an exemplary visualization of the application landscape, called a *software map*. This software map is assembled like a matrix, in which the business processes make-up the x-axis and organizational units make-up the y-axis. Positioning a rectangle in the matrix means that this

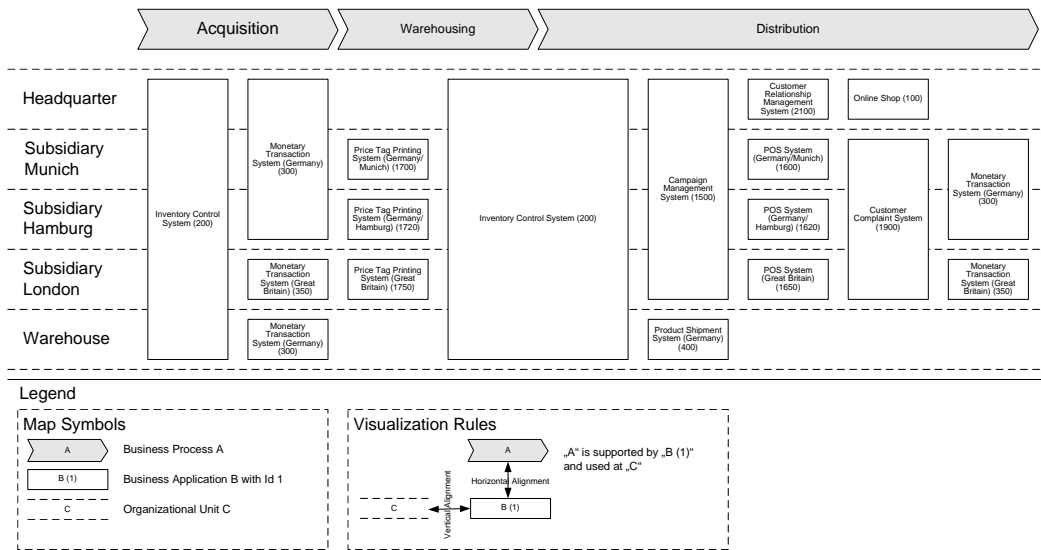


Figure 1.1: Example software map showing the dependencies between business processes, business applications, and organizational units

specific business application supports the business processes to which the rectangle is aligned on the x-axis and is used by the organizational unit to which the rectangle is aligned on the y-axis. This relationship between business processes, business applications, and organizational units can be seen as a ternary relationship, which is discussed in detail in chapter 3. Furthermore stretching the rectangle for a business application in either x- or y-direction refers to the fact that the business application supports either more than one business processes or is used by more than one organizational unit.

From Figure 1.1 it can e.g. be derived that the business application 'Monetary Transaction System (Germany)' supports the business processes 'Acquisition' and 'Distribution'. The supporting system for the 'Acquisition' process is used by the organizational units 'Headquarter', 'Subsidiary Munich', 'Subsidiary Hamburg', and 'Warehouse' and the support function for the 'Distribution' process is used by the organizational units 'Subsidiary Munich' and 'Subsidiary Hamburg'. It can also be derived from Figure 1.1 that the organizational unit 'Subsidiary London' uses a different business application 'Monetary Transaction System (Great Britain)' to support the business processes 'Acquisition' and 'Distribution'.

By now, hardly any tool available on the market is capable of generating such visualizations using a predefined visualization scheme from data stored in a repository, as discussed in the 'Enterprise Architecture Management Tool Survey 2005' [se05]. This coheres with the lack of standardized ways of visualizing application landscapes. Existing standards like the Unified Modeling Language (UML) [OM05b] developed by the Object Management Group (OMG) are used to visualize the architecture of a single business application and are usually not suitable for displaying the whole application landscape. The concepts of UML do not match with the concepts needed to document an application landscape [MW04, LMW05c]. Information elements like organizational units, etc. as well as visualization elements needed

to build Figure 1.1 are not known by UML.

A further problem arising from this situation characterized by manually drawn diagrams is that the visualizations have no well-defined semantics that determines, which graphical symbol in the visualizations refers to which semantic element in the repository. This leads to visualizations, which may only be interpreted correctly by the author of the drawing. Therefore, managing a major investment by drawing figures, which cannot be interpreted correctly, is quite problematic: 'Drawing is no management!' [Ri05].

The following chapters show our approach to *conveniently and flexibly generate software maps* using the well known concept of model transformation (see chapter 2). The approach is designed to offer both flexibility in visualization definitions and a strict definition of what the visualization concepts signify. In chapter 3 we give a summary of the main requirements on software maps and will introduce the types of software maps we have identified so far.

Chapter 4 presents the central part of our approach, the *visualization model*, defined informally using an object-oriented model and formally using mathematical terms. Before we give an outlook in chapter 6, related approaches for solving similar problems are discussed in chapter 5.

Generating Software Maps using Model Transformation Techniques

The motivation in chapter 1 stated that today, software maps are often created manually. Tools employed to create software maps can be distinguished into two groups, the non-repository-based tools like Microsoft Visio or Microsoft PowerPoint and the repository-based tools like ARIS Toolset (from IDS Scheer AG) or Corporate Modeler (from Casewise, Inc.)¹.

The non-repository-based tools do not provide out of the box functionalities to create software maps, e.g. Microsoft Visio stencils providing shapes and rules for drawing software maps. Existing stencils for Microsoft Visio, e.g. the ArchiMate [JLL06] stencil, provide basic functionality for modeling software maps, but some problems still prevail:

- Other stencils, which are not part of the methodology, can be used. This softens the given methodology by enabling the user to add symbols, which are not part of the methodology, resulting in drawings without defined semantics.
- The stencils do not directly support positioning-related concepts such as alignment, which is e.g. used in Figure 1.1 to show a support/use-relationship between business processes, business applications, and organizational units.
- Non-repository-based tools do not use an information model, which defines the semantics of elements, their attributes, and relationships to other elements. A correspondent information model for Figure 1.1 would define the three elements business process, organizational unit, and business application. All elements would have an attribute 'name' and a ternary relationship 'supports' would connect the three elements (see chapter 3 for further explanations)².

¹For further tools utilized to create software maps refer to the 'Enterprise Architecture Management Tool Survey 2005' [se05].

²Of course, this does not comprise definitions what concepts like *business process* etc. exactly are. Therefore, an additional glossary is needed.

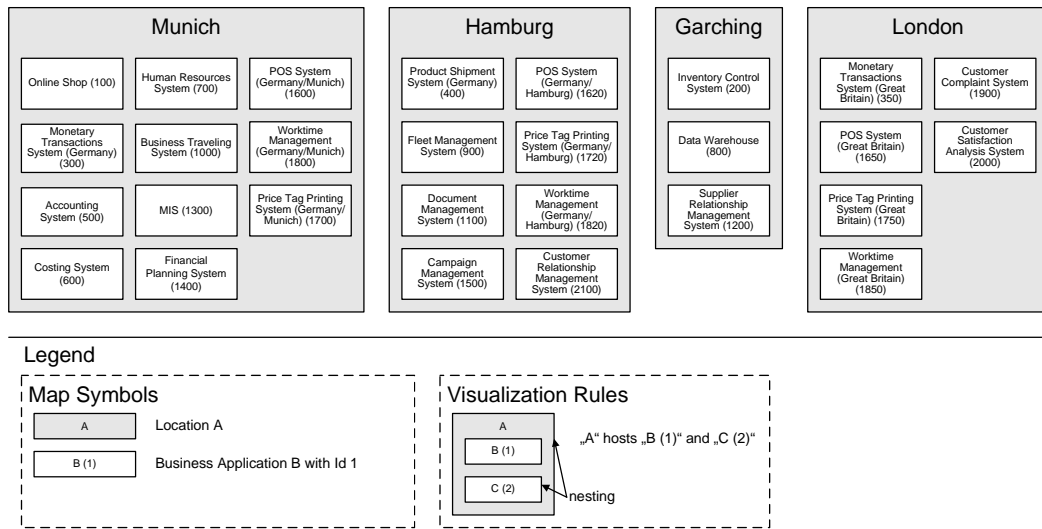


Figure 2.1: Exemplary cluster map showing which business application is hosted at which location

The repository-based tools either have a predefined information model or provide the possibility to create or adapt the information model. However, the 'Enterprise Architecture Management Tool Survey 2005' [se05] showed that the visualization concepts of such tools are still lacking some capabilities, especially concerning flexible and automated generation of software maps.

To clarify the problems arising during the creation of software maps, we explain the differences between a semantic model and a symbolic model in the following section 2.1, using the software map shown in Figure 2.1 as a running example. The differentiation of the semantic and the symbolic model is also a key concept behind our approach for software map generation.

2.1 Relationship between Semantic and Symbolic Models

Figure 2.1 shows a software map, which is of the type *Cluster Map*³, showing which locations host which business applications. The business application 'Online Shop (100)' for example is hosted at the location 'Headquarter'.

To gain a more in depth understanding of the software map in Figure 2.1, we outline how we re-engineered four models that address different aspects of the software map. These four models, described in the following paragraphs, led us to our approach for software map generation.

Creating a semantic model, containing the information of the software map, and an information model, representing the metamodel, results in the UML object and class diagrams in Figures 2.2 and 2.3 if an object-oriented notation is chosen. The object diagram in Fig-

³For a description of software map types see chapter 3.

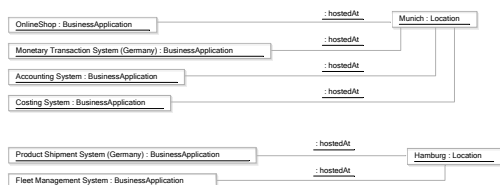


Figure 2.2: Semantic model of the cluster map in Figure 2.1



Figure 2.3: Information model of the cluster map in Figure 2.1

Figure 2.2 visualizes parts⁴ of the semantic model and each object in the diagram represents an information object (e.g. the location 'Munich', the business application 'Human Resources System (700)'). The properties of the information objects, not visualized in Figure 2.2, are the values the attributes of an object, e.g. the value 'Munich' for the name of the location Munich and '100' for the id of the business application 'Online Shop'.

The information model in Figure 2.3 is the metamodel on which the semantic model is based, it consists of

- the two classes **Location** and **BusinessApplication**,
- the attribute **name** for both classes,
- an attribute **id** for the class **BusinessApplication**, and
- an association **hosted at**.

Beside the class diagram the semantics of the elements (classes, attributes, associations, etc.) has to be specified, which can for example be done in a glossary in a textual way or more formally, if e.g. specific use cases call for this. An entry in such a glossary for the class **BusinessApplication** may be:

A Business Application is a software system with a minimum of one user and a database to store information. Further, a business application has to support at least one business process.

More semantic information may be retrieved from the visualization, if e.g. the gray background color of the locations also has a semantics. In our example all semantic information is covered by the class diagram in Figure 2.3⁵.

Beside the semantic model and the information model we can also derive a symbolic model and a visualization model from the software map in Figure 2.1, which are also modeled using UML object and class diagrams are shown here. The symbolic model (Figure 2.4) shows what

⁴The whole semantic model is too large for a figure in this report, because each symbol in Figure 2.1 must be represented by an object, resulting in an object diagram with 34 objects.

⁵It has to be noted that re-engineering a software map can only be done with the creator of the map if no information model and/or legend does exist.

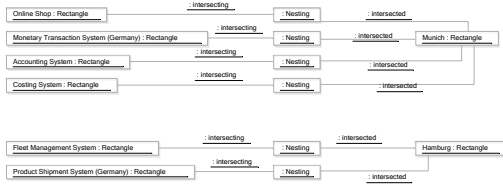


Figure 2.4: Symbolic model of the cluster map in Figure 2.1

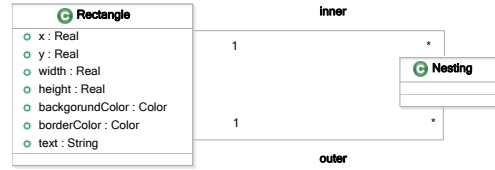


Figure 2.5: Visualization model of the cluster map in Figure 2.1

symbols a visualization consists of and what demands and constraints exist regarding their positioning. The visualization model (Figure 2.5) is the metamodel of the symbolic model, specifying what kinds of symbols can be used in a visualization and what constraints can be imposed to their positioning.

The only kind of symbol used in Figure 2.1 is a rectangle, which also provides a text-field and is called **Rectangle** in the visualization model in Figure 2.5. In addition there exists a positioning constraint, which is expressed by a visualization rule, in the software map. This visualization rule, which is represented in the visualization model by the class **Nesting**, meaning that a symbol is nested inside another symbol, is used to represent the association *hosted at* (see Figure 2.3). An additional visualization rule **PlanarNonIntersection**, which defines that two specific symbols may not intersect, is not contained in Figure 2.5 due to readability issues and will be introduced in chapter 4.

The visualization objects of the symbolic model, which are modeled as objects in Figure 2.4, are instances of classes from the visualization model in Figure 2.5 and correspond to the symbols used in Figure 2.1. The attributes of each object are not shown in Figure 2.4, e.g. the object `Munich:Rectangle` has the attribute value 'Munich' for the attribute `text` and the attribute value 'gray' for the attribute `backgroundColor`.

In the research project Software Cartography we call symbols like **Rectangle** *map symbols*⁶ and the variables of each symbol, e.g. the `backgroundColor` attribute, are called *visual variables*⁷. The rules in a visualization, e.g. the 'Nesting', are called *visualization rules*⁸. These three terms are adopted from cartography (see [HGM02] and [Sl03]). Further map symbols, visual variables, and visualization rules are introduced in section 2.2. Section 4 develops formal definitions of the terms in the context of Software Cartography and automated visualization generation.

After having retrieved the four models from the visualization we gain a deep understanding of the software map and know which information should be contained in the map: Figure 2.1 shows, which business application is hosted by which location using rectangles with a gray background color for locations and rectangles with a white background color for business applications. The hosting relationship is expressed in Figure 2.1 by nesting a rectangle for a business application inside a rectangle for a location.

⁶in German *Gestaltungsmittel*

⁷in German *Gestaltungsvariable*

⁸in German *Gestaltungsregel*

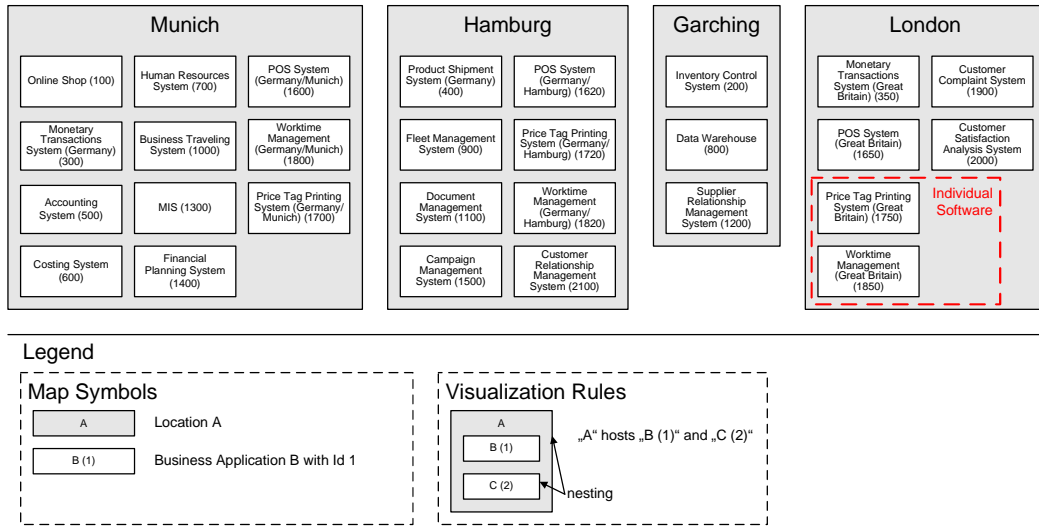


Figure 2.6: Cluster map with an additional map symbol

A problem, which we have often been confronted with in practice, are visualizations without such a defined information model, visualization model, and a connection between these models, defining what information is expressed by which visualization concept. Either all or some of the models and their connection are not documented, neither with class diagrams nor in a textual way, or models are defined but not applied correctly.

If e.g. the modeling tool allows adding further elements to a visualization without having them properly connected to information in the semantic model, visualizations such as in Figure 2.6 can be created. In Figure 2.6, the rectangle with a red colored dotted border has no connection to any element in the semantic model (see Figure 2.2). This gap leads to a visualization, of which the information can neither be interpreted correctly nor be stored in a repository in an accessible and maintainable way. This is because the information model defines the structure for the information and the information that some systems are 'Individual Software' is not intended here.

The argumentation above shows that a *tight* link between the information model and the visualization model is needed to guarantee consistency between the the information (semantic model) in the repository and the visualization (symbolic model). The following section 2.2 presents our approach to ensure this consistency.

2.2 Model Transformation and Software Maps

Model transformation is a technique to translate one or more source models into one or more target models. In software engineering, model transformation is used in many different ways. For example, E/R models are transformed to SQL statements capturing the data definition language for a database schema or, for example, UML class diagrams are transformed to Java source code. An OMG recommendation using model transformation in software engineering

is the Model Driven Architecture [OM01] (MDA)⁹.

In case of the MDA, a *Platform Independent Model* (PIM) might be a UML class diagram or a set of diagrams, which are using only concepts on a highly abstract level without being platform specific, where being not platform specific means that the PIM makes no assumptions about the implementation platform and is technology-neutral. For example a PIM which models a software system might be independent of a specific programming language and can be transferred to a model using Java, C++, or SQL which is then the *Platform Specific Model* (PSM), using concepts of the specific platform.

A PIM might use a concept named *String*, which would be transferred into a Java-PSM concept `java.lang.String` or a SQL-PSM concept `VARCHAR(20)`. The concepts for transforming a PIM-String to a concept in a PSM vary depending on the platform. In the case of SQL additional information that the string has a maximum length of 20 characters is also needed, since SQL only supports a string with a given maximum length. Therefore, information might have to be added during the transformation.

In addition the transformation in our example has to specify how a metamodel concept from the PIM is transferred to a metamodel concept of the PSM. If the PIM is modeled using UML class diagrams the transformation to Java will transform a UML class to a Java class, whereas a transformation to SQL may transform a UML class to a SQL table definition.

2.2.1 Leveraging Model Transformation for Generating Software Maps

The idea of transforming models is used in automated generation of software maps to tighten the link between the semantic model and the symbolic model. Transforming an element of a semantic model (e.g. `Accounting System:BusinessApplication`¹⁰) to an element of a symbolic model (e.g. `Accounting System:Rectangle`) can be performed by a set of rules. One rule may state that every business application is transformed to a rectangle with text using a white background color.

The concepts of model transformation used for generating software maps are shown in Figure 2.7. Beside the concepts semantic model, information model, symbolic model, and visualization model, which have been introduced in section 2.1, a *Metamodel* and a *Transformation* are introduced.

The *Metamodel* is the common language for the information model and the visualization model. As the semantic model with its information objects is an instance of the information model, the information model itself is an instance of a metamodel, likewise the visualization model is. The common metamodel for the information model and the visualization model simplifies the requirements for the *Transformation*, because if no common metamodel exists, the transformation must also transform one metamodel into the other metamodel. The transformation sketched in Figure 2.7 is meant to be a bidirectional transformation, which is expressed by the arrow showing to the semantic and the symbolic model. A bidirectional transformation provides the possibility to create visualizations by transforming a semantic

⁹It has to be noted that a semantic model and a symbolic model cannot be compared to a CIM, PIM, or PSM. The concepts PIM and PSM refer to a dependency on a platform. Semantic and symbolic models do not deal with platform dependencies.

¹⁰The notation `A:B` means the object A is an instance of the class B.

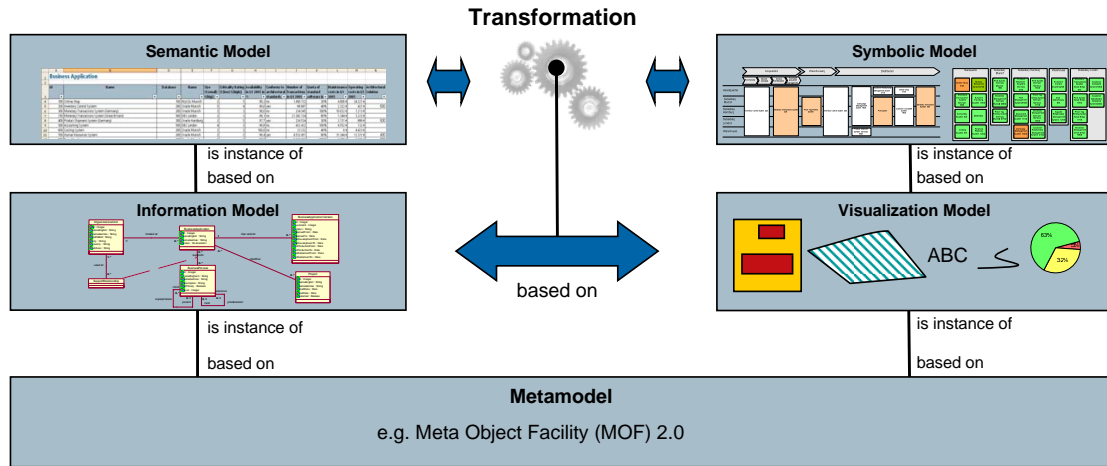


Figure 2.7: Model transformation techniques in software map generation

model to a symbolic model and also to change the semantic model by interacting with the visualization¹¹.

An important requirement on the metamodel is that all concepts needed for conveniently modeling information and visualization models and also for a straightforward expression of the transformation with the help of a suitable language or approach, are supported by the metamodel. In the case of leveraging model transformation for generating software maps, we propose an object-oriented metamodel for the information model and the visualization model. Buckl [Bu05] showed that the assumption of an object-oriented metamodel like the Meta Object Facility (MOF) [OM06a] is sufficient for information models [Ha04, La05, Br05a] commonly in use in the domain of Software Cartography. That the assumption of an visualization model using MOF concepts also holds, is shown in chapter 4.

Furthermore, a common metamodel simplifies the transformation, since both models are using the same *language*. A transformation rule stating that each **BusinessApplication** is transformed to a **Rectangle** (see above) needs a mechanism to create rectangles with text for each business application. In MOF, this can be performed using the **Factory** allowing to create new **Elements**.

2.2.2 Introducing a Visualization Model

The goal of using model transformation for the generation of software maps is to provide a flexible approach for generating visualizations. Furthermore, the transformation ensures that the semantic and the symbolic models fit to each other.

Some possible elements of an information model have been introduced in section 2.1, e.g. a business application or a business process. Elements of the visualization model are for example rectangles, circles, and lines, called map symbols. Other important elements of the

¹¹In this report we only cover the direction from semantic to symbolic model.






Name	Symbol
Rectangle	
Ellipse	
Chevron	
Cylinder	
Line	

Table 2.1: Examples for map symbols









Name	Visual Variable
Background color	 
Border color	 
Line size	 
Line type	 

Table 2.2: Example for visual variables





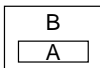





Name	Example	Visualization Rule
XSequence	Chevron for process A is to the left of chevron for process B	 
PlanarNonIntersection	Rectangle for application A does not overlap with rectangle for application B	 
Nesting	Rectangle for application A is nested in the rectangle for organizational unit B	
Alignment	Rectangle for application A is positioned under the chevron for process B	 
Attachment	Line representing a data stream is attached at the rectangles for applications A and B	  

Table 2.3: Examples for visualization rules

visualization model are the visualization rules, which define constraints on map symbols, mostly regarding their positioning, i.e that one symbol is nested inside of another symbol.

To leverage the idea of using model transformation for generating software maps a visualization model is needed, which covers all *visual concepts* used on software maps. Tables 2.1 and 2.2 show some examples for map symbols and different values for visual variables. The rules relating map symbols geometrically are exemplified in Table 2.3. As currently no common language for describing these symbols and rules exists (see chapter 5), it is yet not directly possible to describe the visualizations in another manner than a quite informal textual or example driven way. If a higher level of preciseness shall be achieved, a more formal description for visualizations is needed. Such a more formal way has to satisfy several requirements, e.g. it has to be precise enough to act as a basis for automatic creation of maps and it has to be easy to understand in order to keep it simple in use. Chapter 4 gives a comprehensive overview of these requirements and introduces a visualization model, which contains elements sufficient for addressing visualization as described in section 3.


```
rule OrgUnit2Rectangle {
  from
    infoObject : Semantic.OrganizationalUnit
  to
    symbol : Symbolic.Rectangle (
      text = infoObject.name,
      backgroundColor = #CCCCCC
    )
)

rule BusinessApp2Rectangle {
  from
    infoObject : Semantic.BusinessApplication
  to
    symbol : Symbolic.Rectangle (
      text = infoObject.name || "(" || infoObject.id || ")"
    ),
    rule : Symbolic.Nesting (
      inner = symbol,
      outer = transforming (infoObject.hostedAt)
    )
)
```

Figure 2.8: Exemplary transformation rule set

2.2.3 Introducing Transformation Rules

The missing part to complete the generation of software maps via model transformation are the transformation rules. Transformation rules on the level of models (*information model* and *visualization model*) can formally express the graphical notation of the application landscape in the context of a viewpoint¹². These rules do not only map certain information model classes, e.g. **BusinessApplication** to visualization model classes, e.g. **Rectangle**; they can further contain predicates and expressions used to filter the information to be transformed, thus hiding concepts not of importance in the particular viewpoint. The transformation rules can then be used to transform information objects and their semantic relationships to visualization objects and their positioning relationships. A simple example for a transformation rule generating the software map in Figure 2.1 by transforming the semantic model in Figure 2.2 to the symbolic model in Figure 2.4 may be the rule set¹³ in Figure 2.8.

The first rule **OrgUnit2Rectangle** transforms each information object, which is an instance of the class **OrganizationalUnit**, to an instance of **Rectangle**. In addition, the text field is set to the name of the information object and the background color is set to gray (**#CCCCCC**). The second rule transforms each business application to a **Rectangle**, setting the text field to a combination of the name and the id of the business application. Furthermore, instances of the

¹²Viewpoint in terms of the IEEE Std 1471-2000 [IE00, LMW05a].

¹³The transformation rules are expressed using pseudo-code.



Figure 2.9: Excerpt of the symbolic model in Figure 2.4

$$\begin{aligned}
 x_{r1} - \frac{w_{r1}}{2} - \left(x_R - \frac{w_R}{2} \right) &> 0 \\
 y_{r1} - \frac{h_{r1}}{2} - \left(y_R - \frac{h_R}{2} \right) &> 0 \\
 x_R + \frac{w_R}{2} - \left(x_{r1} + \frac{w_{r1}}{2} \right) &> 0 \\
 y_R + \frac{h_R}{2} - \left(y_{r1} + \frac{h_{r1}}{2} \right) &> 0 \\
 \dots
 \end{aligned}$$

Figure 2.10: Inequality systems derived from the symbolic model in Figure 2.9

association **hostedAt** from a business application to an organizational unit are transformed to a instances of the class **Nesting**, which are respectively connected to the rectangle corresponding to the business application and the rectangle corresponding to the organizational unit.

We are currently analyzing, whether model transformation languages like ATL [AT06], which resembles the pseudo-code in Figure 2.8, or other approaches, like automatically generated Java classes, fit the requirements for the transformation mechanism best.

Assuming that the visualization model is sufficiently strictly defined and that there are sufficiently efficient ('computable') ways of finding valid values for the visual variable (i.e. positions of the map symbol instances in most cases) the semantic model can be used to automatically generate a corresponding visualization. In chapter 4 we present such a strictly defined visualization model on which we build a mechanism that translates the problem of finding a suitable layout into an optimization problem. The constraints of the optimization problem, which results from the symbolic model in Figure 2.9, showing the symbolic model with the annotated rectangles **R**, **r1**, and **r2**, and parts of the resulting inequality system are shown in Figure 2.10. The target function of the optimization problem could e.g. be used to control aesthetic aspects as e.g. distances between map symbol instances.

The complexity and kind of optimization problem depends on the type of the visualization rules to be employed. Therefore, special attention has to be paid to the solver used to solve the optimization problem (see chapter 4). The result of the calculations performed by the solver is positioning information, which can be used by a rendering engine to create visualizations.

The automated view generation introduced in section 2 is based upon a method for the representation of application landscapes, the so-called software maps [LMW05c, LMW05b]. In this section the software maps are briefly introduced and basic graphical concepts behind the maps are identified. Based on this, chapter 4 outlines a comprehensive model of these concepts, which is suitable for describing the graphical buildup of software maps. This model constitutes a metamodel of the visualization for software maps and can therefore serve as a visualization model as mentioned in section 2.

Similar to the concept of maps used in cartography, a software map is composed of a *base map* and several *layers*. While in the field of conventional cartography a visualization can rely on prominent topological characteristics, e.g. latitude and longitude, to derive the base map, no such characteristics exist in software cartography. Therefore, no single base map exists, suitable for building all different kinds of visualizations on. After having analyzed various software maps we have been able to derive different types of software maps, differentiated e.g. by the kind of base map they are utilizing:

- Software maps with a base map for positioning - utilizing the concept of positioning map symbol instances to transport (formally) defined semantics
- Software maps without a base map for positioning - performing ad-hoc positioning of the map symbol instances

3.1 Software maps with a base map for positioning

The main characteristic of these software map types is that an explicit meaning is assigned to the position of the map symbols present on the base map, i.e. the position has semantics. These software maps are further subdivided into three map types the *cluster map*, the

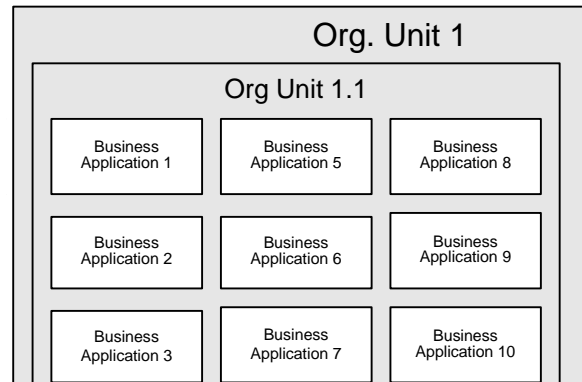


Figure 3.1: Excerpt from a cluster map

#	Graphical Concept	Meaning
1	Gray rectangle with label “Organizational unit *”	Organizational unit called “*”
2	White rectangle with label “Business Application *”	Business application called “*”
3	Nesting of a rectangle as in 1 in a rectangle as in 1	Organizational unit belongs to another organizational unit
4	Nesting of a rectangle as in 2 in a rectangle as in 1	Organizational unit uses a business application

Table 3.1: Meaning of the graphical concepts in Figure 3.1

process support map, and the *interval map*, according to the layout of the base map and the information contained therein.

3.1.1 Cluster Map

The characteristic feature of a *cluster map* is that it utilizes the concept of *nesting* to visualize a relationship between the cluster element and the nested element, as it is shown in Figure 3.1. In this figure, the rectangle labeled *Org. Unit 1.1* is nested within the rectangle labeled *Org. Unit 1*, whereas the small rectangles (with labels *Business Applications 1*, *Business Applications 2*, etc.) are all nested within the rectangle *Organizational Unit 1.1*.

The nesting may e.g. visualize an *use*-relationship between an organizational unit and a business application, or a *belongs to*-relationship in the organizational chart, as detailed for the example from Figure 3.1 in Table 3.1. Therefore, Figure 3.1 shows that organizational unit 1.1 belongs to organizational unit 1, and e.g. that this organizational unit uses business application 1.

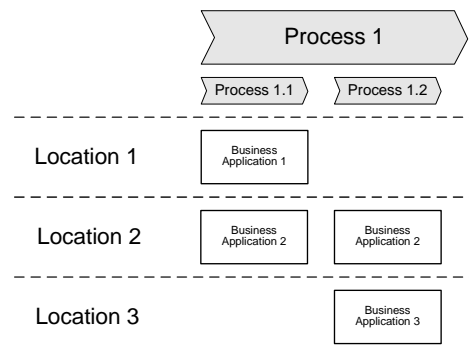


Figure 3.2: Excerpt from a process support map

3.1.2 Process Support Map

The main graphical concept behind the *process support map* is the alignment of one map symbol in respect to another map symbol. Thereby, it is possible that a map symbol totally or partially overlaps another map symbol, in respect to its x-coordinate or also in respect to its y-coordinate. An example is shown in Figure 3.2 (Figure 1.1 depicts a complete exemplary process support map), where the rectangle labeled *Business Application 1* overlaps with the chevron labeled *Process 1.1* in respect to the x-coordinate and with the text box *Location 1* in respect to the y-coordinate.

Furthermore, in a process support map, the map symbols that are used as a reference for the alignment are always located at the edge of the map, forming two axes: a horizontal and a vertical one. The vertical axis is in most cases located at the left edge of the map, the horizontal one at the top. As it is often made up of map symbols representing processes [LMW05c], this software map type is called *process support map*.

Thus, two kinds of map symbols on a process support map can be distinguished: the *axis map symbols*, which are part of the two axes of the map, and the *aligned map symbols*, which are aligned to the axes.

The buildup of the axes suggested above leads to a major difference to axes as they are *usually* employed in coordinate systems. While the values denoted on a coordinate axis are commonly continuous attributes, the axes of a process support map convey discrete attributes, as there is only a finite number of axis map symbols on each axis.

The exemplary process support map in Figure 3.2 visualizes information that conforms to the information model (metamodel) shown in Figure 3.3. Thereby, the concepts described in this information model are visualized by the graphical concepts of the process support map as follows:

- A *business process*-instance is visualized as a chevron, these chevrons are also used to build a x-axis, thereby they are ordered as indicated by the *predecessor*-relationships of the business *business process*-instance they represent. If a *business process*-instance is a subprocess of another business process-instance, the respective chevrons are shown

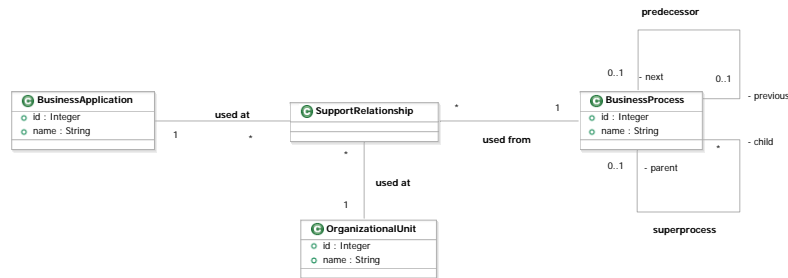


Figure 3.3: Relationship between organization unit, business process and business application

below the parent process.

- Each of the boxes forming the y-axis is used to visualize one *organizational unit*-instance. In this case, an arbitrary ordering is used.
- Each *business application*-instance is visualized by one or more rectangles, which are the aligned map symbols of the process support map. For each *supports relationship*-instance, a rectangle representing the *business application*-instance indicated by the association end *supports* is shown, and this rectangle is aligned to the chevron representing the supported business process and the rectangle representing the location where this business process is supported (in the context of the respective *supports relationship*). Thus, it can e.g. be deduced from Figure 3.2 that *Business Application 1* supports *Process 1.1* at *Location 1* and that *Business Application 2* supports the same process *Process 1.1* at *Location 2*.

When considering the objects that can be assigned to an axis in a *process support map*, it has to be distinguished whether the objects are ordered or not. If these objects are linearly ordered, it is possible and advantageous to order the map symbol instances representing these objects, leading to the concept of an *ordered axis*. If there is no order established on the objects (or such a relation exists, but is not used), the objects can nonetheless be used to form an axis, although the ordering of the axis map symbols is arbitrary then. This may e.g. be utilized to optimize the aesthetics or map space consumption of the visualization.

3.1.3 Interval Map

An *interval map* consists of similar types of map symbols as the process support map described above, *axis map symbols* and *aligned map symbols*. The characteristic difference is that at least one of the axes visualizes a continuous attribute. As such an axis is able to convey information about an interval scaled attribute [Fa99] without losing information, this software map type is called *interval map*.

Taking the concept *time* as the continuous attribute is very common for these interval maps [LMW05c], which is then in most cases visualized on the horizontal axis. The at-

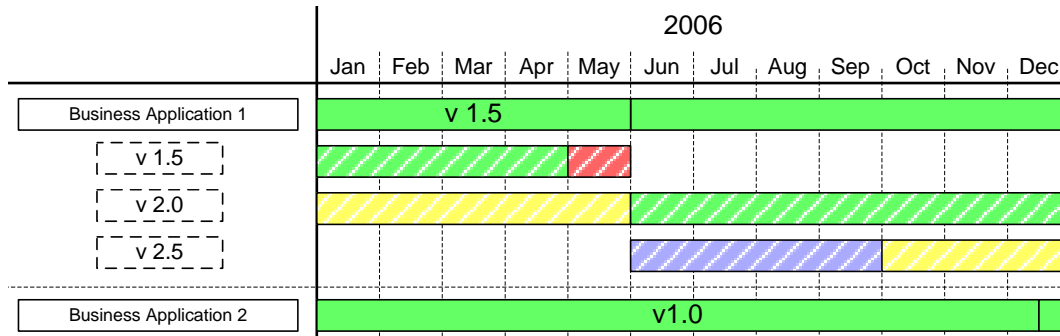


Figure 3.4: Excerpt from an interval map

tribute displayed on the vertical axis varies from usage context to usage context, but mostly, a discrete attribute is used.

As with the process support map, the attribute values for an instance represented by an aligned map symbol are visualized by alignment with certain areas of the axes.

Thereby, a prominent aspect is visualizing time spans, as for example in Figure 3.4. By placing the rectangle labeled *v1.5* below the interval of the horizontal axis that ranges from January to December, the following is expressed: the version *v1.5* has been in existence from January to December¹. Aligning in respect to the horizontal axis determines the corresponding business application *Business Application 1*.

3.2 Software maps without a base map for positioning

The second category of software maps, the software maps without a base map for positioning, does not use positioning of the map symbols on the base map to convey information to the viewers, which means that the position of these symbols can be changed without affecting the meaning of the visualization. Concepts affecting the positioning of map symbols transport an explicit meaning only on the layers above the base map, e.g. a traffic light positioned in the upper right corner of a symbol expresses a relationship between the information object visualized and the measure visualized by the traffic light.

3.3 The Layering Principle

The layering principle, exemplified in Figure 3.5, has been adopted from (geographic) cartography [Sl03]. It constitutes a basis for selective visualization of information in software maps. As it is possible to display various types of information (population density, average income, etc.) on the topographic base map of a geographic map, additional information can also be visualized on the base maps defined by the software map types described above. In

¹For further explanations of the colors of the rectangles, see [LMW05b].

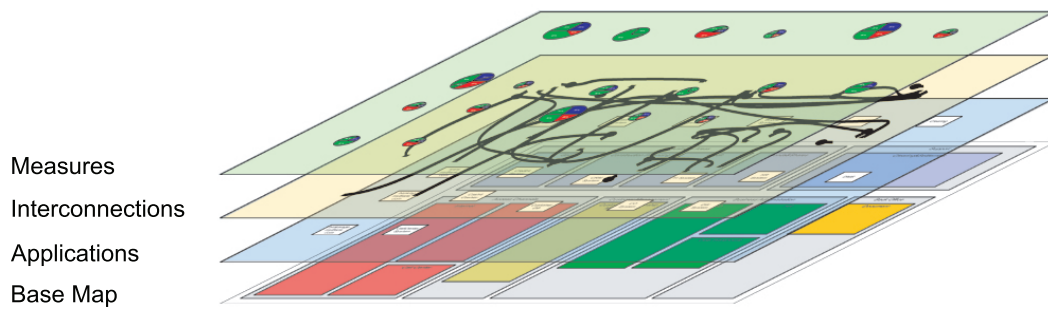


Figure 3.5: Layers of a software map

Figure 3.5, a cluster map - base map is used and on top of this base map, e.g. connections between business applications or some measures (visualized by pie charts) can be displayed. Arranging such additional information in layers yields two main benefits:

- As the layers can be ordered, it becomes clear which map symbol instances are above which other map symbol instances, i.e. cover which map symbol instances.
- More important, the layers can be specifically shown and hidden, making selective information visualization possible. Thus, the layering principle is an instrument suitable to hide complexity, which is unnecessary in a given usage context.

Chapter 2 identified a *visualization model* as an integral part of sebis' approach to software map generation and chapter 3 presented in detail the visualizations that should be covered by such a model. Now, a visualization model consisting of visualization concepts that fulfill the requirements arising from the demands specified (like *map symbols* and *visualization rules*) is introduced. Visualization objects, e.g. *map symbol instances*, are formally defined here, in terms of this visualization model. Additionally the model is constructed to fulfill specific requirements arising from the way this kind of model is meant to be used (see chapter 2). According to those requirements a visualization model should be

- sufficiently *easy to understand* in order to be a helpful tool in specifying software maps,
- sufficiently *strictly defined* in order to be a basis for automated visualization generation,
- sufficiently *flexible* in order to leverage the introduction of new map symbols and visualization rules,
- sufficiently *adaptable* in order to enable improvement of the automated layouting without having to redesign the basic structure of the model, and
- sufficiently *efficiently calculable* in order to ensure the existence of time efficient (and perhaps memory efficient) algorithms for creating software maps.

As the demands stated above span different levels of abstraction as well as different aspects of use, our approach abstains from a holistic and monolithic model to address all of them in an adequate way. Instead, the visualization model pursues a three tiered approach, as shown in Figure 4.1.

The topmost level of the model (the *Object-oriented Visualization Model*) is formed by an object-oriented representation of the visualization concepts suggested above (i.e. map symbols

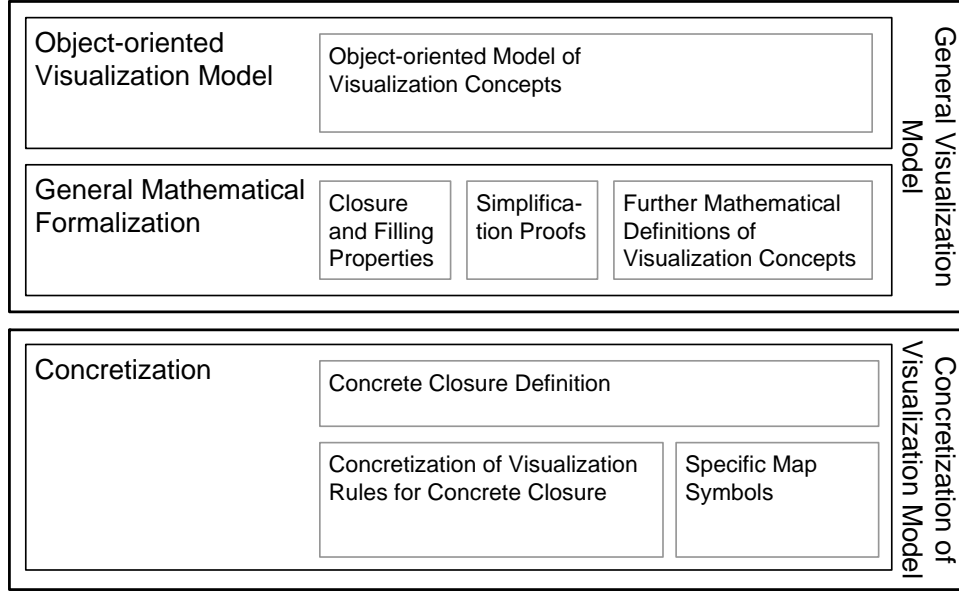


Figure 4.1: Three-tiered structure of the visualization model

and visualization rules), which is described more detailed in section 4.1. This representation strongly depends on the *Meta Object Facility 2.0 (MOF)* [OM06a] and can therefore conveniently be described utilizing *class diagrams*, which themselves constitute a well-known way of providing information about the general structure of an object-oriented model to an user. Despite the obvious advantages of such a representation, MOF was not intended for and hence does not provide sufficient capabilities for describing the exact nature of visualization concepts (their graphical *semantics*) in an unambiguous way.

The second level of the model (the *General Mathematical Formalization*), formed by mathematical and logical terms and expressions, is therefore introduced to complement the object-oriented description by an exact mathematical description of the respective visualization concepts. This mathematical description utilizes both *algebraic* terms as well as terms from the *predicate calculus* in order to cover the different kinds of graphical representations and implications of the visualization concepts by a denotational semantics [Sc04]. Section 4.2 explains this mathematical description more in detail. Together, the object-oriented representation and its mathematical complement form the *General Visualization Model*. Due to the highly different *languages* used for expressing those two parts, one might think, that the General Visualization Model in fact decomposes into two models, just informally coupled. However, a quite formal way of coupling the two layers is possible, based on a formalization of the object-oriented constructs. Such a formalization is out of the scope of this article, thus referencing to considerable work already done in this field, e.g. by Broy [BCR06] or by Evans [Ev98], the latter one pursuing an approach utilizing the Z-notation [Sp92] that strongly bases on concepts of the predicate calculus.

The demands for an *easy to understand* (class diagrams), *strictly defined* (mathematical definitions), and *flexible* (using object-oriented mechanisms to introduce new concepts) visualization model are full or in major parts already satisfied by the General Mathematical

Visualization Model. The third level (the *Mathematical Concretization*) of our model is concerned with concepts ensuring the model to be *efficiently calculable* and *adaptable*, when concerning its calculation paradigm. The fulfillment of those demands strongly depends on constructs introduced in the General Visualization Model, such as the *filling* and the *closure*, which form a basis for an effectively and efficiently calculable concretization of this model. These concepts are introduced in section 4.3 and complemented by proven *mitigations* for visualization rules employing closures and fillings instead of the symbols concerned¹.

The actual concretization of the General Visualization Model, which makes use of these concepts, is described in section 4.4. Thereby, the approach pursued through the concepts mentioned above separates the universally valid specification of the General Visualization Model from its possible concretizations, enabling the implementor to choose a concretization (and thus a mitigation of the universally valid concepts) that best suits his decision in the context of the *calculation complexity* vs. *visualization optimacy* trade-off. Hereby, the *adaptability* demanded is achieved. The General Visualization Model is not affected by any change in the selection of the simplifications made, only the concretization has to be changed, if desired due to optimizations regarding calculation speed or visualization aesthetics. The concretization contains information on how to translate instances of the visualization model concepts, e.g. into an optimization problem as subsequently described in section 4.4.1.

4.1 Object-oriented Visualization Model

As mentioned above, the object-oriented representation of the visualization concepts makes up the top level tier of the visualization model. In order to conveniently model the concepts on this tier, MOF [OM06a] and UML class diagrams (see [OM05a]) as a notation are subsequently used. Thus the *classes* displayed in the diagrams are instances of `MOF:Constructs:Class`. Thereby, we distinguish between a `CoreVisualizationModel`, which defines the basic concepts of the visualization model and extending packages, which have to be introduced to be able to describe visualizations in a way to fit specific application needs. Specific map symbols (such as a `Rectangle` or a `Chevron`) and visualization rules (such as `Nesting` or `Ordering`) can be added in these extending packages, which are thereby built upon the core package by importing it. For exemplary purposes, we introduce a `SoCaVisualizationModel` package here, which builds visualization rules and map symbols suitable for describing software maps as introduced in chapter 3 on the concepts of the core visualization model. Due to the genericity of the core visualization model, it is of course possible to use it as the basis of other visualization model packages, stressing again the exemplary nature of the `SoCaVisualizationModel` package, which, while being suitable for the maps introduced above, is certainly not without potential for further improvement. The package structure made up thereby is shown in Figure 4.2.

Figure 4.3 introduces the core visualization model. Its basic classes and their meaning are explained here briefly:

Diagram represents the model of a complete visualization, as exemplified in Figure 2.1. It

¹Proven *mitigations* here mean, that the fulfillment of certain rules on closures and fillings is proven to ensure the fulfillment of the rule on the corresponding map symbol instances.

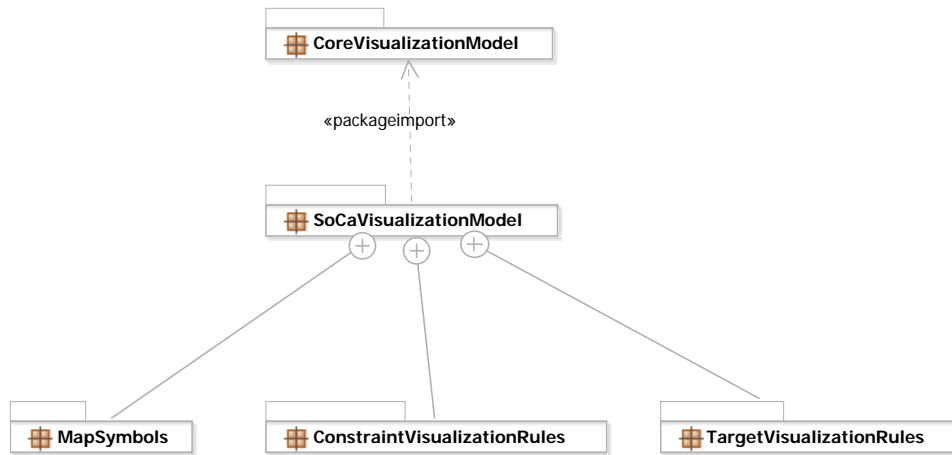


Figure 4.2: Package structure of the visualization model

is made up of the visualization objects that form the respective diagram. Packages that use the core visualization model may add meta-information (via attributes) as e.g. creator or users of the diagram.

VisualizationObject represents an object of the kind that make up a diagram description and may be categorized into map symbols and visualization rules.

MapSymbol represents a type of graphical element of which instances can be used in a visualization, e.g. a rectangle. An instance of a map symbol **Rectangle**, e.g. the concrete rectangle labeled *Munich* in Figure 2.1, is called a map symbol *instance*.

VisualizationRule represents a type of restriction that can be imposed upon one or more instances of visualization objects, e.g. in respect to their placement or other aspects of their appearance in case of map symbols as suggested in section 3. An actual restriction is called a visualization rule *instance*.

ruling represents a bidirectional association. It represents for a map symbol instance the union of all visualization rule instances that are connected to this map symbol instance. For a visualization rule instance, it returns all map symbol instances ruled (i.e. connected) by the respective rule instance.

For purposes of software cartography, the concepts from the **CoreVisualizationModel** package are refined to a **SoCaVisualizationModel** package (see Figure 4.4) containing the following basic concepts:

Real is a primitive type of which an instance a number from the set of real numbers (also denoted as \mathbb{R}).

Color is a primitive type representing color information. This type is explained in detail in section 4.2, whereas here the shorthanded explanation via the considerably different displaying capabilities of different drawing canvases is given, which have to be accommodated by an adequately abstract and hence flexible definition.

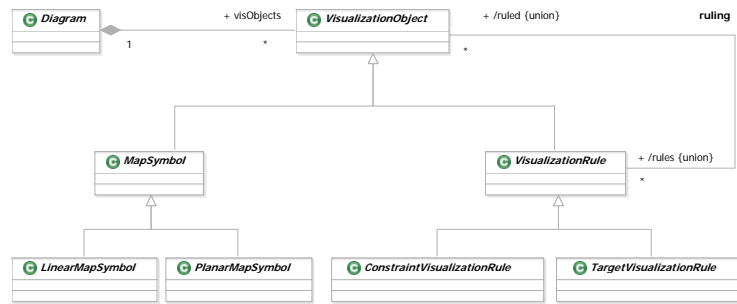


Figure 4.3: Object-oriented core visualization model (Package CoreVisualizationModel)

SoftwareMap represents a specialization of **Diagram**, containing a creation date.

MapSymbol is extended with an attribute **zIndex** that is used to determine, which map symbol instance is meant to be on top of which other and an attribute **visibility** that is used to determine, whether a particular map symbol instance is visible or not.

PlanarMapSymbol is extended with an attribute **borderColor** representing the color of the borderline of the symbol, an attribute **fillColor** representing the color for filling inner parts of the symbol, an optional attribute **text**, representing the contents of a label included in the symbol and an optional attribute **textColor**, used to represent the color of the text in the label included in the symbol.

LinearMapSymbol is extended with an attribute **lineColor** representing the color the linear map symbol instance is drawn with.

Canvas is a specialization of **MapSymbol**, representing the drawing surface of a software map. All map symbols contained in the software map topdress the **Canvas**. This symbol is introduced especially for facilitating reflections over the overall properties of the software map, e.g. its dimension.

4.1.1 Map Symbols

Despite the rather intuitive distinction² between linear and planar map symbols, as introduced above, a formal way of determining whether a symbol is planar or linear has to be backed by a mathematic understanding of the map symbol, which is here postponed until the mathematic description is established in section 4.2.

Figure 4.5 shows parts of the structure of the **SoCaVisualizationModel** package. This package imports the contents of the package **CoreVisualizationModel**. The classes **MapSymbol**, **LinearMapSymbol**, and **PlanarMapSymbol** in Figure 4.5 have already been extended in the **SoCaVisualizationModel** package shown in Figure 4.4. The attributes of the map symbol classes constitute the *visual variables*³ (outlined in section 2.1) of the respective map symbol.

A quick overview over the classes introduced in Figure 4.5 is given below:

²This distinction is made as different types of rules apply to different types of map symbols: while it makes sense to ask, whether something is inside an area (planar map symbol instance), it does not make sense to ask,

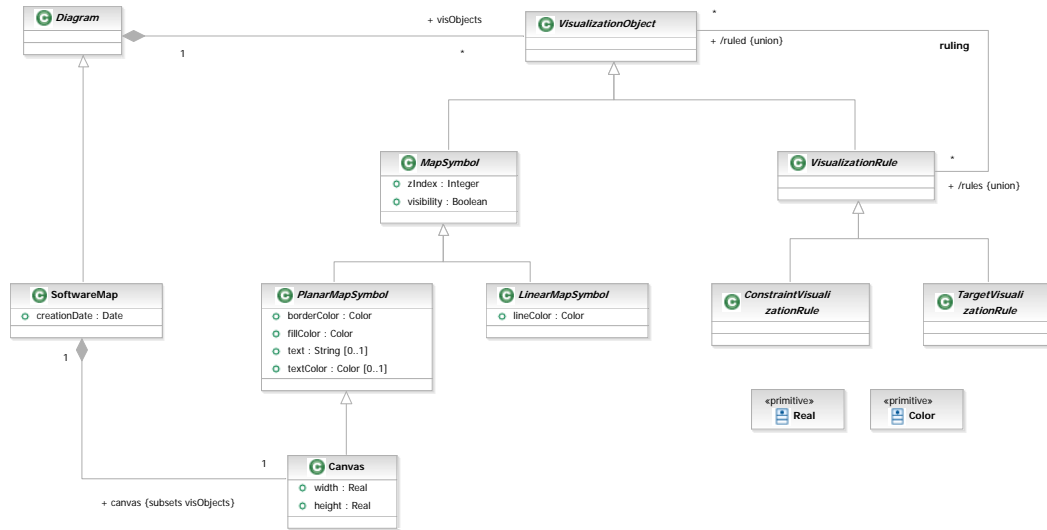


Figure 4.4: Package SoCaVisualizationModel

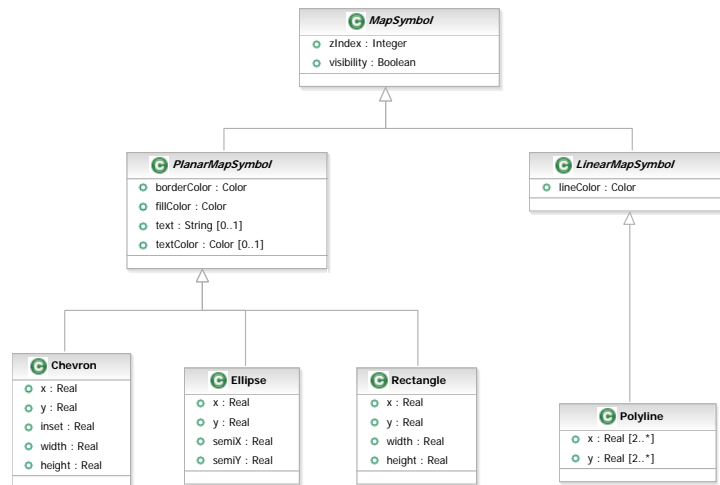


Figure 4.5: Package SoCaVisualizationModel::MapSymbols

Ellipse is a map symbol representing an elliptical shape and is defined via the position of its center (`x` and `y`) as well as via the attributes representing the length of its respective semiaxes (`semiX` and `semiY`). Such elliptical shapes may e.g. be used for visualizing business objects or organizational units in a similar fashion as in *Event Driven Process Chains* [Sc01].

Rectangle is a map symbol representing a rectangular shape and is defined via the attributes representing center (`x` and `y`) as well as via attributes representing the lengths of the sides (`width` and `height`). Such shapes may be used for e.g. representing information objects or functional areas.

Chevron is a map symbol representing a special kind of polygon and is defined via the attributes representing center (`x` and `y`) as well as via attributes representing the lengths of the sides (`width` and `height`). It furthermore employs an additional variable, the `inset`, for describing the distance of the bendpoint in the vertical lines from the rectangular line. This symbol is for example used for representing a business process as shown in Figure 3.2.

Polyline is a map symbol representing a multiply bended line and is defined via the position of its bendpoints (`x` and `y`)⁴. Such shapes may be used for e.g. connecting other symbols to display information flows.

4.1.2 Visualization Rules

Taking a closer look at the visualization rules, shown in Figure 4.3, one can see them being classified into two major categories. The first category contains classes inheriting from `ConstraintVisualizationRule`, which are rules that have to be fulfilled in a correct layout. Primarily, those rules are employed to express semantics of the visualization, thus ensuring that a visualization corresponding to the rules indeed carries the respective semantic information from the corresponding semantic model (i.e. the information objects), as outlined in section 2.1. The fulfillment of those rules is crucial to the generation of a software map, as not obeying them would compromise the information contained in the visualization.

The other category is formed by classes inheriting from `TargetVisualizationRule`. These are rules that can be fulfilled more or less, as e.g. a demand to minimize the space between two map symbols can be satisfied in different grades according to the space actually present between those symbols. The most prominent use case for such a rule is controlling aesthetic aspects of visualizations. The correctness of a visualization is not affected by the extent to which target visualization rules are fulfilled, although a visualization should have a layout fulfilling them to the maximum possible degree in order to achieve optimal aesthetic appeal. A formalization of this behavior is described in section 4.2.

Figures 4.6, 4.7, and 4.9, visualizing parts of the `SoCaVisualizationModel` package, show the classes `ConstraintVisualizationRule` and `TargetVisualizationRule`, which have been

whether it is inside a line (linear map symbol instance).

³It has to be ammended that additional *visual variables* can easily be found, but are omitted here reflecting the exemplary character of this model.

⁴We regard the starting point and the end point also to be bendpoints.

imported from the `CoreVisualizationModel` package. Specific rules are introduced to the `SoCaVisualizationModel` package via subclassing from `ConstraintVisualizationRule` and `TargetVisualizationRule`, leading to a set of classes that are adequate for the layouting principles of the different kinds of software maps introduced in chapter 3.

In chapters 2 and 3 some visualization rules included in Figure 4.6 have already been introduced or been pointed out. They are shorthanded below:

IdentityOfProjection enforces that the instances of `VisualizationObject` referenced by an instance of this rule have the same value assigned to a variable shared by the instances, which is supplied as value of `targetVariable`. This rule is e.g. employed in the construction of axes and used to specify that all map symbol instances forming the axis start at the same x- or y-coordinate (depending on the kind of axis).

Ordering demands the instance of `VisualizationObject` that is referenced by the association `rulingOrderingUpper` to have a greater value assigned to a specific variable than the value assigned to the `targetVariable` in the visualization object instance referenced by the `rulingOrderingLower` association. The variable, which has to be specified when using the visualization rule, is a visual variable of this rule.

Intersection is an abstract base class for a couple of visualization rules that all demand two symbols to intersect horizontally and/or vertically as described in section 3.1.2.

FullXSpecificIntersection demands that the map symbol instance referenced by `intersectingSymbol` shares its x coordinates with the `intersectedSymbol`, providing the degree of freedom to be totally located directly below, above, or at the same y level as the map symbol instance specified by `intersectedSymbol`. This is exemplified by *Location 1* and *Business Application 1* in Figure 3.2.

FullYSpecificIntersection demands that the map symbol instance referenced by `intersectingSymbol` shares its y-coordinates with the `intersectedSymbol` and is thus totally located directly left, right, or at the same x level as the map symbol instance specified by `intersectedSymbol`.

Nesting demands that the map symbol instance referenced by `intersectingSymbol` is totally located inside the map symbol instance specified by `intersectedSymbol`. This rule has been introduced to be able to express the nesting relationships essential to the *cluster map* described in section 3.1.1.

PlanarNonIntersection demands that two planar map symbol instances, which are referenced by `nonIntersectingSymbols`, do not overlap at all. This rule has been introduced in order to be able to use other visualization rules in a meaningful manner, e.g. in order to prevent map symbol instances from being nested accidentally.

XSequence demands that the map symbols supplied by the `ordered` association end `ruled` of `xsequencing` are positioned from left to right on the canvas without having common points. This rule is prominently used, when making up an axis as the one shown by the *Business Processes* in Figure 3.2.

YSequence demands that the map symbols supplied by the `ordered` association end `ruled` of `ysequencing` are positioned from top to bottom on the canvas without having common

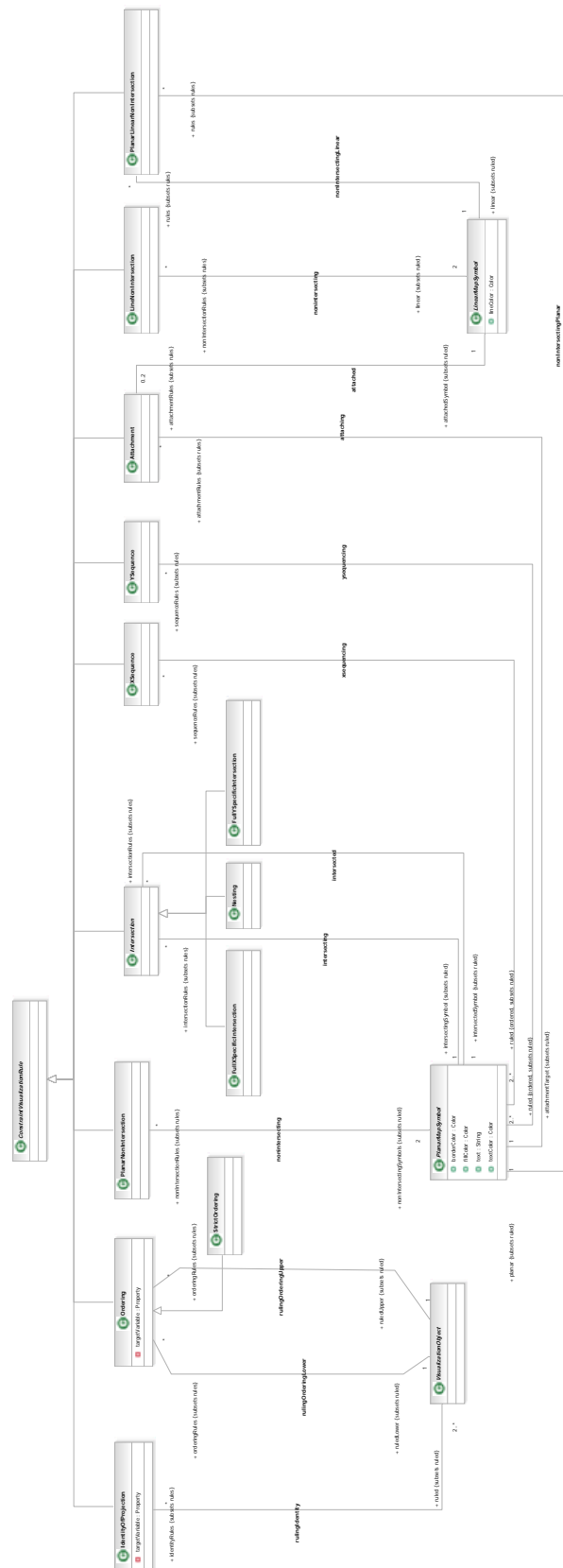


Figure 4.6: Package SoCaVisualizationModel::ConstraintVisualizationRule; *Property* is EMOF::Property [OM06a]

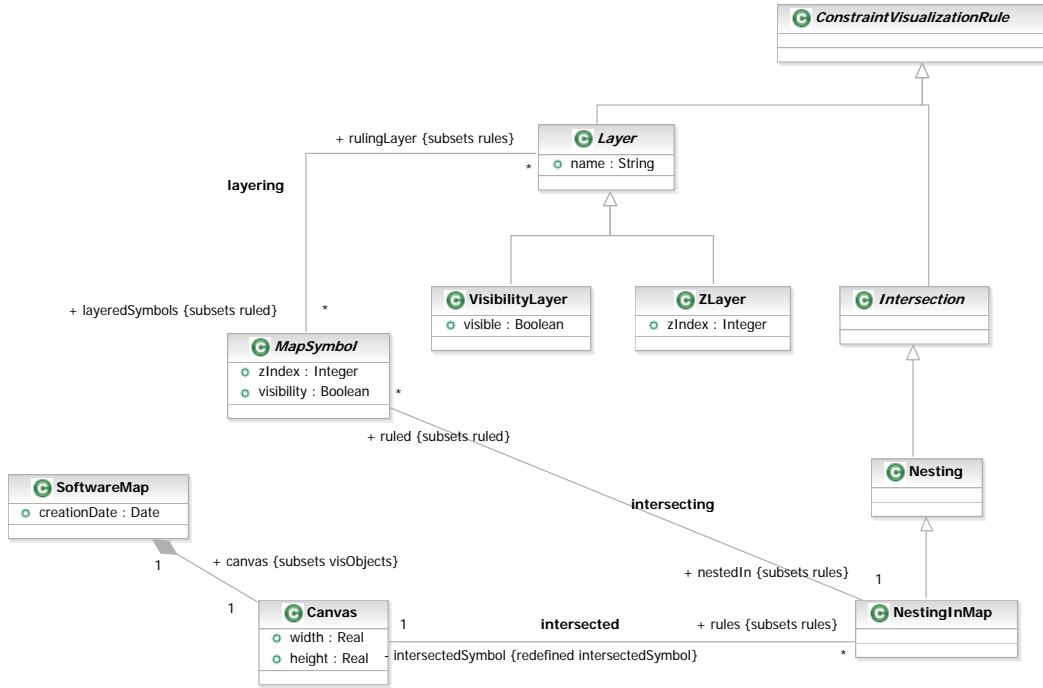


Figure 4.7: Layering principle and the map canvas in Package SoCaVisualizationModel

points. This rule is prominently used, when making up an axis as the one shown by the *Locations* in Figure 3.2.

Attachment is employed in building software maps containing instances of **LinearMapSymbol** for e.g. showing a data flow between two business applications, where a visualization rule for attaching an end of a linear map symbol to a planar map symbol is needed. This rule attaches the linear map symbol instance specified by **attachedLine** to the planar map symbol instance referenced to by **attachmentTarget**.

LineNonIntersection demands, that the linear map symbol instaces referenced to by **linear** do not have any common point except their potentially identical ends.

PlanarLinearNonIntersection demands that the linear map symbol instance referenced to by **linear** must not intersect the area made up by the planar map symbol instance specified by **planar**.

To be able to address visualizations as the ones shown in chapter 3 with the constructs contained in the exemplary SoCaVisualizationModel package, some concepts are still missing: layers, as introduced in section 3.3, and rules concerning the **Canvas**. These constructs are introduced in Figure 4.7.

Targeting at both linear and planar map symbols, the layering principle as described in section 3.3 can be handled by constraint visualization rules as shown in Figure 4.7. Instances of these rules are relating a set of map symbol instances specified by **layeredSymbols**, prescribing the values of one of their visual variables, as e.g. their **zIndex** or **visibility**. Thereby,

an instance of a layer visualization rule enforces that all map symbol instances referenced by `layeredSymbols` share the same value for a specified visualization variable. The exact variable targeted thereby is determined by a subclass of `Layer`, which then also can contain a variable that can hold the desired value of the layered symbol instances' target visual variables. A precondition of this is, of course, that the map symbols of which the layered symbol instances are instantiations, share the respective variable.

Thus, an instance of a rule `VisibilityLayer` can control the `visibility` attribute of a set of map symbol instances, thereby offering the behavior to show or hide a *layer* of a visualization. Likewise, an instance of a `ZLayer` rule can impose requirements on the values of the `zIndex` attribute of its layered map symbol instances. Changing these restrictions allows bringing portions (i.e. *layers*) of a diagram to the front or to the back.

Figure 4.7 also introduces the rule `NestingInMap` targeting the `Canvas`, in order to ensure that all map symbol instances employed in a software map are placed within the canvas. This rule is derived from the `Nesting` visualization rule. Additionally the following OCL constraint on class `SoftwareMap` exists, demanding that each map symbol instance, which is not a `Canvas` instance, is nested in the `Canvas` of its corresponding software map:

```
self.visObjects -> forall(vo | vo.metaClass = VizualiationRule or
                           vo = self.canvas or
                           (vo.metaClass = MapSymbol and vo.nestedIn.intersectedSymbol = self.canvas))
```

Applying a rule for area size control on the canvas can now e.g. be used to control the size of a software map.

A set of instances of the rules described above may potentially be inherently contradictory, meaning, that complete fulfillment of all the rules included in the set might not be possible for any visualization. This might be caused by rather obviously contradictory demands, as e.g. the demands of map symbol instances A and B to be mutually nested in each other. But there can also be an elaborate inherent contradiction, caused by e.g. `Attachment` and `LineNonIntersection` instances making up a graph, which might contain an a priori *non-planar* subgraph. Despite the possibility to demand unsatisfiable sets of constraints on the visualization, no *meta-rules* or *meta-constraints* are imposed for preventing inherently contradictory sets of constraints, demanding of a potential concretization thereof in order to handle contradicting constraints adequately.

When considering the example software maps described in section 3, the *constraint visualization rules* described above are sufficient to specify the graphical relationships that have to be fulfilled in order to create a map that has the desired semantics. However, there are still deficiencies e.g. in respect to specifying, which of the possible layouts are more desirable due to their superior aesthetics. For example, the two software maps in Figure 4.8 carry the same semantics when considering the semantics specification in table 3.1, but the rules described above offer no possibility to express that a layout mechanism should tend to produce a layout more similar to the one on the left side.

Therefore, the `SoCaVisualizationModel` package introduces the following *target visualization rules* (see Figure 4.9), as subclasses of the class `TargetVisualizationRule`, which has been imported from `CoreVisualizationModel`. It has to be noted that these rules are not mere

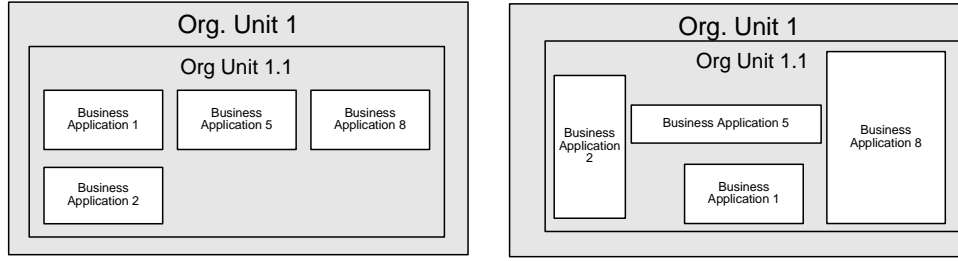


Figure 4.8: Two semantically equivalent software maps of different aesthetic appeal

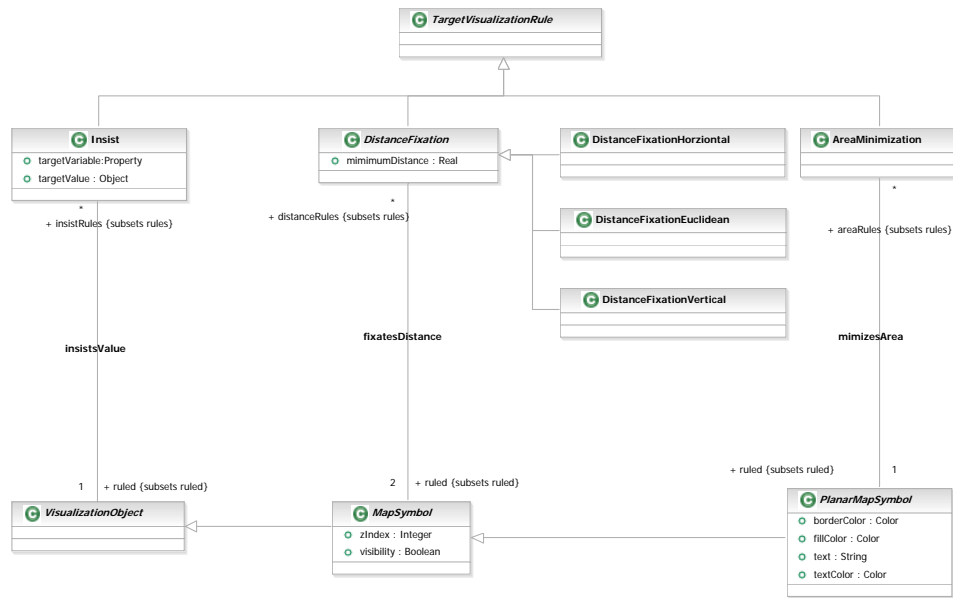


Figure 4.9: Package SoCaVisualizationModel::TargetVisualizationRule)

constraints which can be fulfilled or not, but do more have the nature of a target function. A more formal description of this is presented in section 4.2.

The target visualization rule **Insist** demands of a visualization object instance to have the value of one of its visual variables, which is specified in the rule instance's **targetVariable** attribute, to be as close as possible to a value specified in the rule instance's **targetValue** attribute.

The target visualization rule **DistanceFixation** targets the distance of two map symbol instances specified by **ruled** to be as close as possible to a specified value or exceeding this value, providing subclasses to further refine the kind of distance measurement employed by the rule, to e.g. a **DistanceFixationHorizontal** or a **DistanceFixationVertical** relying on a projection of the distance, or the **DistanceFixationEuclidean**, called this may as it is computed by an *euclidean norm*.

A last target visualization rule introduced here is called **AreaMinimization**, demanding the minimization of the area covered by a planar map symbol instance referenced by **ruled**.

Concludingly, a last concept from the object-oriented visualization model has to be mentioned, the possibility to restrict instances of certain visualization rule classes to relating only instances of certain map symbol classes (e.g. rectangles and ellipses). A restriction of that kind could be achieved in various ways, e.g. by introducing a constraint specified by an OCL-expression [OM06b] or by referencing sub- and superclasses for corresponding map symbol classes. None of these approaches is further pursued at the moment, as an actual demand for such restrictions has not yet been experienced in the visualization model derived from real world examples of software maps.

Introducing the concepts of a visualization model in an object-oriented way is meant to make the model easy to understand and to use (e.g. in specifying transformation rules or software map types), but nevertheless needs a stricter definition. This definition, which lays the ground for the automated generation of software maps, has to be more formal and is introduced as a general mathematical specification of the concepts' semantics in respect to their graphical appearance in section 4.2.

4.2 General Formalization of the Visualization Model

The object oriented model, outlined in Figures 4.3, 4.5, 4.6, 4.7, and 4.9, gives, as already mentioned, no exact but only a textual definition of the concepts represented. Therefore, the classes representing map symbols and visualization rules are augmented and complemented with a mathematical definition of their visualization specific semantics. The augmentation employed in our three tiered visualization model (see Figure 4.1) and the concepts incorporated therein are presented in the following sections.

4.2.1 Prerequisites

The mathematical formalization of the visualization-specific denotational semantics [Sc04] of the visualization concepts is built on some basic concepts, which are presented below.

Drawing surface: coordinate system and representation

The concept of a *drawing surface* for the visualization is constituted by a *coordinate system* and a *representation*, which are closely related to each other. Therein, it has to be kept in mind, that such a formalization has to address the characteristics of any kind of drawing surface on a level that is abstract enough for not unnecessarily restricting possible concretizations of the visualization model. Nevertheless, the definition of the drawing surface has to be sufficient to form a base for the formalization of the other concepts of the visualization model.

Hence, the drawing surface is defined as a triple of sets, first the set \mathbb{R}^2 of points from a two dimensional space, second the set \mathbb{P} of presentation values assignable to any point, and third the set \mathbb{B} of Boolean values ($\{true, false\}$). Exemplifying \mathbb{P} , one could think of it as a set of *colors*. Every point $k \in \mathbb{R}^2$ is subsequently presented as a tuple of coordinates $k = (x, y)$ from a two dimensional *cartesian coordinate system*, of which the two vectors $((1, 0)^T$ and $(0, 1)^T$) form a basis (see [Br05b]) spanning the whole two dimensional space.

In the context of the general formalization of the visualization model, \mathbb{P} is not specified in detail, as the visualization concepts of the drawing surface chosen strongly influence the concepts supported. In the object-oriented visualization model tier, this set is expressed by the type `Color`. The general mathematical visualization model tier (see figure 4.1) only demands the set \mathbb{P} to support a function l . This function maps a multiset s of tuples of kind $(r, i) \in \mathbb{P} \times \mathbb{N}_0$ to a value $r' \in \mathbb{P}$. Thereby, the tuples (r, i) consist of the presentation information r of a map symbol and a positive integer, representing the value of the *z-Index* of the corresponding map symbol. This function l is used for composing the representation information of the map symbol instances to the representation information of the diagram on the drawing surface. This may be achieved e.g. by *top-dressing* their presentation information according to their *z-Indices*. Furthermore, a *neutral element* $\epsilon \in \mathbb{P}$ is demanded to exist. Understanding \mathbb{P} as colors, the neutral element ϵ could be interpreted as *transparent color*. Regarding this neutral element, the following holds for s_0 being any kind of multiset of tuples as described above:

$$l(s_0 \uplus \{(\epsilon, i)\}) = \begin{cases} l(s_0) & \text{if } s_0 \neq \emptyset \\ \epsilon & \text{if } s_0 = \emptyset \end{cases}.$$

For this expression it has to be noted, that the operation \uplus (*join*) operates on multi sets, therefore including duplicates, instead of leaving them out, as the standard set calculus would demand.

Assignment of Variables

When concerning instantiations of visualization object classes to e.g. specific rectangles or circles on the drawing surface, another concept has to be introduced, the concept of *assignment*. An assignment is a function from a set of variables (e.g. from a term) to the cartesian product of the domains of these variables, such that every variable is assigned a value from its domain. For further information about the concepts of assignment, see e.g. Broy [Br98]. Subsequently we refer to the concept assignment using the following notation: given an exemplary term a employing e.g. two integer variables a_1 and a_2 , the application of a specific assignment $\beta \in (a_1, a_2) \rightarrow \mathbb{Z}_0 \times \mathbb{Z}_0$ to the term is denoted as $a[\beta]$.

4.2.2 Visualization Objects - Definition

On the object-oriented visualization model tier, we introduced the class `VisualizationObject` as a core concept of our visualization model. In this section we provide mathematical augmentations, applicable for visualization objects, which are all direct or indirect subclasses of the corresponding class. The set of all non-abstract subclasses of `VisualizationObject` is subsequently denoted as \mathbb{O} .

In order to facilitate reflections on the attributes of the visualization objects, to which we refer as *visual variables*, as well as on the association ends, visualization objects might have, we introduce the following functions taking subclasses $o \in \mathbb{O}$ `VisualizationObject` as arguments:

- a function $\mathbb{V}(o)$, which supplies the set of all (including inherited) attributes of the visualization object class, excluding associations and
- a function $\mathbb{A}(o)$, which supplies the set of all (including inherited) association ends of the visualization object class.

Regarding the domain for every attribute and association, we supply two more functions, which are necessary, as not every visual variable or every association end shares the same domain:

- a function $\mathbb{W}^V(v)$ supplying the domain for a visual variable $v \in \mathbb{V}(o) \wedge o \in \mathbb{O}$ and
- a function $\mathbb{W}^A(a)$ supplying the domain for an association end $a \in \mathbb{A}(o) \wedge o \in \mathbb{O}$.

The issue of multiple domains employed in our general mathematical visualization model tier could be addressed by introducing a many sorted algebra [Ga86], which is deliberately not used here, as the complexity tied to this construct seems inadequate to us compared to its benefit at the current evolution stage of the visualization model.

Regarding the multiplicity of attributes on the object-oriented visualization model tier, it has to be mentioned that regarding specific attributes v , $\mathbb{W}^V(v)$ may not only be a set of distinct values, but can also be

- a subset of the power set of a value set, when regarding a *multivalued*, *unique*, and *not ordered* attribute,
- a set of multisets over a value set, when regarding a *multivalued*, *not unique*, and *not ordered* attribute,
- a set of sequences, with each kind of element from a value set occurring at most once in each sequence, when regarding a *multivalued*, *unique*, and *ordered* attribute, and
- a set of sequence of elements from a value set, when regarding a *multivalued*, *not unique*, and *ordered* attribute.

Similarly, the multiplicity of the corresponding association end on the object-oriented visualization model tier induces that for a specific association end a , $\mathbb{W}^A(a)$ cannot only be a set of distinct visualization object instances, but can also be

- a subset of the power set of a set of visualization object instances, when regarding a *multivalued*, *unique*, and *not ordered* association end,
- a set of multisets over a set of visualization object instances, when regarding a *multivalued*, *not unique*, and *not ordered* association end,
- a set of sequences, with each kind of element from a set of visualization object instances occurring at most once in each sequence, when regarding a *multivalued*, *unique*, and *ordered* association end, and

- a set of sequences of elements from a set of visualization object instances, when regarding a *multivalued*, *not unique*, and *ordered* association end.

A more formal definition of the functions \mathbb{W}^V and \mathbb{W}^A is provided in appendix B.

Regarding the two direct and abstract subclasses of class `VisualizationObject` provided on the object-oriented visualization model tier, `MapSymbol` and `VisualizationRule`, two sets are introduced as follows

- the set $\mathbb{G} \subseteq \mathbb{O}$ containing all non-abstract subclasses of class `MapSymbol` and
- the set $\mathbb{D} \subseteq \mathbb{O}$ containing all non-abstract subclasses of class `VisualizationRule`.

It furthermore has to be mentioned that these two sets form a partition of \mathbb{O} , i.e. that $\mathbb{G} \cap \mathbb{D} = \emptyset$ and $\mathbb{G} \cup \mathbb{D} = \mathbb{O}$ hold.

Instantiation of Visualization Objects

Complementing the creation of an instance of a subclass of class `VisualizationObject` in the object-oriented visualization model, the general mathematical visualization model utilizes the concept of assignment as introduced above. The instantiation of a class $o \in \mathbb{O}$ is thereby represented in the general mathematical formalization in section 4.2 by two distinct assignments:

- the assignment β of one value from $\mathbb{W}^V(v)$ to each attribute $v \in \mathbb{V}(o)$ and
- the assignment α of one value from $\mathbb{W}^A(a)$ to each association end $a \in \mathbb{A}(o)$.

These assignments can be interpreted as functions:

$$\beta : v_i \in \mathbb{V}(o) \mapsto w_j \in \mathbb{W}^V(v_i)$$

and

$$\alpha : a_i \in \mathbb{A}(o) \mapsto w_j \in \mathbb{W}^A(a_i).$$

The following notation is used for an instance of a subclass o of `VisualizationObject`: $o[\beta][\alpha]$. As in some cases the map symbol instances or visualization rule instances are not directly influenced by one of the two assignments, partial notations of assignment as e.g. $o[\beta]$ as shorthand for $o[\beta][\alpha]$ are used. In order to facilitate some reflections about visualization rules, two functions $\mathbb{E}_\beta(o)$ and $\mathbb{E}_\alpha(o)$ for a visualization object class o are defined as the sets of all potential assignments of its variables or association ends - a so called *environment* (see Broy [Br98]). Thereby following auxiliary functions for a set $\mathbb{O}_i \subseteq \mathbb{O}$ can be introduced⁵:

⁵In the following expression the notation $o[\beta]$ is not the shorthanded notation, but does denote a partial assignment.

$$\begin{aligned}
\mathbb{I}_\beta(\mathbb{O}_i) &= \bigcup_{o \in \mathbb{O}_i} \{o[\beta] \mid \beta \in \mathbb{E}_\beta(o)\}, \\
\mathbb{I}_\alpha(\mathbb{O}_i) &= \bigcup_{o \in \mathbb{O}_i} \{o[\alpha] \mid \alpha \in \mathbb{E}_\alpha(o)\}, \text{ and} \\
\mathbb{I}_{\beta\alpha}(\mathbb{O}_i) &= \bigcup_{o \in \mathbb{O}_i} \{o[\beta][\alpha] \mid \beta \in \mathbb{E}_\beta(o) \wedge \alpha \in \mathbb{E}_\alpha(o)\}.
\end{aligned}$$

Relying on the definitions of $\mathbb{E}_\beta(o)$ and $\mathbb{E}_\alpha(o)$ as the sets of potential assignments, the auxiliary function $\mathbb{I}_{\beta\alpha}(\mathbb{O}_i)$ can be seen as the set of all distinct visualization object instances that can be created from the visualization objects in a given subset of \mathbb{O} . As this function is subsequently rather frequently used, $\mathbb{I}(\mathbb{O}_i) := \mathbb{I}_{\beta\alpha}(\mathbb{O}_i)$ is introduced as a short notation here.

Projection

Regarding the assignment of values to variables of a visualization object class o , a function for accessing the value of a specific variable is introduced here as the concept of *projection*⁶:

$$o[\beta]_x = \begin{cases} w & \text{if } (x \in \mathbb{V}(o)) \wedge (\beta : x \mapsto w) \\ \perp & \text{otherwise} \end{cases}.$$

4.2.3 Map Symbols - Definition

Outlined in [LMW05c] as a common concept behind a software map, further specified in section 4.1 as a subclass of class `MapSymbol` holding certain attributes, the map symbol class is here further augmented by a mathematical definition. Thereby, each of the map symbol classes $g \in \mathbb{G}$ is augmented with a tuple of functions $(p(g), r(g))$, supplied for each map symbol by functions p and r .

These functions, of which the domains both are the set of map symbols \mathbb{G} , while their ranges again are functions themselves, are defined as:

$$p : g \in \mathbb{G} \mapsto (\mathbb{R}^2 \times \bigtimes_{v_m \in \mathbb{V}(g)} \mathbb{W}^V(v_m) \rightarrow \mathbb{B})$$

and

$$r : g \in \mathbb{G} \mapsto (\mathbb{R}^2 \times \bigtimes_{v_m \in \mathbb{V}(g)} \mathbb{W}^V(v_m) \rightarrow \mathbb{P}).$$

In order to promote readability, abbreviations regarding the functions above are introduced and consequently employed in this report. Given a map symbol class $g \in \mathbb{G}$ we shorthanded write

$$p_g := p(g) \text{ and } r_g := r(g)$$

with

⁶ \perp denotes the *undefined* value.

$$p_g : \mathbb{R}^2 \times \prod_{v_m \in \mathbb{V}(g)} \mathbb{W}^V(v_m) \rightarrow \mathbb{B}$$

and

$$r_g : \mathbb{R}^2 \times \prod_{v_m \in \mathbb{V}(g)} \mathbb{W}^V(v_m) \rightarrow \mathbb{P}.$$

Having outlined the role of visual variables, the functions p_g and r_g can be interpreted as follows:

- p_g is a function mapping every point of \mathbb{R}^2 to a Boolean value. This value states whether the given point belongs to the map symbol instance or not. Such a function can also be called a *predicate* and further has to be dependent on the values of this map symbol instance's *visual variables*, which is further detailed below.
- r_g is a function mapping every point of the set \mathbb{R}^2 to a graphical representation value from the set \mathbb{P} , e.g. a color. The function r_g further is also clearly dependent on the values of the map symbol instance's visual variables, which is also detailed below.

For understanding the interpretations above, it is necessary to know, that the visual variables are assigned distinct values in every map symbol instance, while the variables x and y remain free. This concept is exemplified in section 4.2.5.

By separating p_g and r_g , flexibility concerning the descriptive capabilities of the model is achieved. The set of points $\{(x, y) \in \mathbb{R}^2 | p_g(x, y) = \text{true}\}$ makes up a kind of *solid body* for a map symbol instance, which is used for determining, e.g. collisions of map symbol instances and is therefore relevant to the visualization rules. The function r_g can assign presentation information to points including also ones not contained in $\{(x, y) \in \mathbb{R}^2 | p_g(x, y) = \text{true}\}$, making it possible to create a kind of *corona* (see figure 4.10) for a map symbol instance. Concerning this, it has to be noted, that the points contained in $\{(x, y) \in \mathbb{R}^2 | p_g(x, y) = \text{true}\}$ should form a *pathwise-connected space* [We05] in order to avoid visualization effects that might no longer be intuitively perceivable for the viewer, as exemplified in figure 4.11. Furthermore, the effects provided by a corona have to be used with potential misleading visualization effects in mind.

Instatiation of map symbols

The creation of a map symbol instance is defined according to the creation of a visualization object instance, as outlined above. The following notation for an instance of the map symbol $g \in \mathbb{G}$ is used: $g[\beta][\alpha]$. As the assignment α does not exert direct implications on the functions $p(g)$ and $r(g)$, we subsequently shorthand $g[\beta][\alpha]$ by $g[\beta]$, as the assignment β further applies to the tuple of functions extending g , for which we introduce the following shorthanded notations:

$$p_g[\beta] := p(g[\beta]) = p(g[\beta][\alpha])$$

and

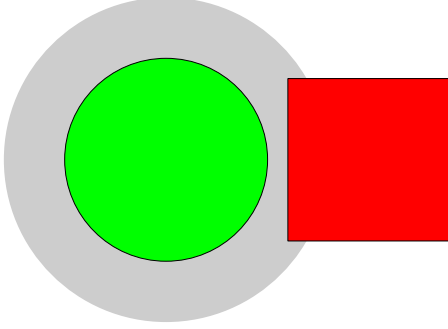


Figure 4.10: Two non touching map symbols with the corona of the circle touching the rectangle.

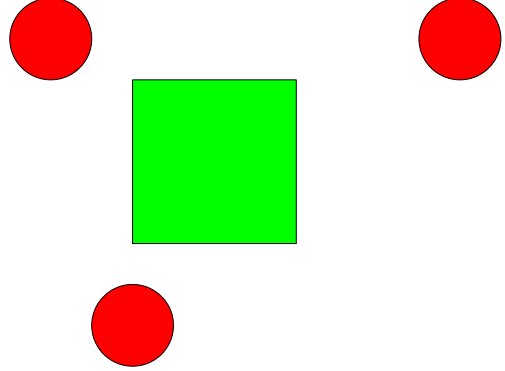


Figure 4.11: Misinterpretable map symbol: is the quadrat contained in the symbol made up by the three circles together?

$$r_g[\beta] := r(g[\beta]) = r(g[\beta][\alpha]).$$

This means that the respective variables in p_g and r_g are assigned the values specified by β .

Similarly, we utilize the $\mathbb{I}(g)$ notation for our reflections about instances of a map symbol g . As shown above, merely the assignment β of the visual variables exerts influence on the predicate p_g and is thereby of interest regarding visualization rules.

Different kinds of map symbols

Two kinds of map symbols, denoted as subclasses of **MapSymbol** in Figure 4.5, are distinguished, mainly due to the different visualization rules that apply to them: *planar map symbols*, which enclose an area and *linear map symbols*, which are for example lines, curves, etc., which do not enclose an area. This informal distinction can be formalized utilizing the terminus *dimension*, e.g. defined as the Hausdorff Dimension [Ma91], and applying it to the predicates p_g of the map symbols. Thus, two sets of map symbols, both subsets of \mathbb{G} , follow:

- \mathbb{G}_L , the set of all linear map symbols and
- \mathbb{G}_F , the set of all planar map symbols.

Furthermore the following expressions hold, making \mathbb{G}_L and \mathbb{G}_F a partition of \mathbb{G} :

$$\begin{aligned} \mathbb{G}_L &\subseteq \mathbb{G} \\ \mathbb{G}_F &\subseteq \mathbb{G} \\ \mathbb{G}_L \cap \mathbb{G}_F &= \emptyset \\ \mathbb{G}_L \cup \mathbb{G}_F &= \mathbb{G}. \end{aligned}$$

4.2.4 Map Symbols - Auxiliary concepts

Especially regarding the formalization of the visualization rules in section 4.2.6, the following auxiliary concepts are of use:

Point removal

A function called σ is used for removing a specific point from the points contained in a map symbol instance, whereas the containment in the preceeding expression is defined by the fulfillment of the map symbol instance's predicate $p_g[\beta]$. Therefore, the function σ is defined as follows:

$$\sigma(p_g[\beta](x, y), (x_e, y_e)) := \begin{cases} p_g[\beta](x, y) & \text{for } (x \neq x_e) \vee (y \neq y_e) \\ false & \text{for } (x = x_e) \wedge (y = y_e) \end{cases}.$$

Pathwise connected space

The function called χ is used for determining, whether the set of points making up the *model*⁷ of a predicate forms a pathwise-connected space [We05]. Therefore the function χ is defined as follows:

$$\chi(p_g[\beta]) := \begin{cases} true & \text{if } p_g[\beta](x, y) \text{ forms a pathwise-connected space} \\ false & \text{for } p_g[\beta](x, y) \text{ not forming a pathwise-connected space} \end{cases}.$$

Utilizing this function, the already introduced demand for every map symbol to form a pathwise connected space can be expressed as follows:

$$\forall i \in \mathbb{I}(\mathbb{G}) : \chi(i).$$

Area of a map symbol instance

Subsequently, a concept regarding planar map symbol instances is introduced, called *area* of a planar map symbol instance. The *area*-function of a map symbol instance is defined as follows:

$$a : \mathbb{I}(\mathbb{G}_F) \rightarrow \mathbb{R}_0^+ \cup \{\infty\}.$$

This definition relies on the fact that the following structures can be proven to be homomorphic and the structure Σ forms a σ -Algebra (see [Br05b]), on which a *measure* (see [Br05b]) can be defined:

- a structure *predicates*: $\Phi = \langle \{p | p : \mathbb{R}^2 \rightarrow \mathbb{B}\}, \vee, \neg \rangle$ with \vee and \neg being the usual boolean operators, and

⁷Herein, the term *model* is used in a logical context as the set of values, for which a given predicate evaluates to *true*. For an exact definition see e.g. [EP02]

- a structure *sets*: $\Sigma = \langle \{\mathbb{X} \in \mathbb{R}^2 \mid \mathbb{X} \subseteq \mathbb{R}^2\}, \cup, \cap \rangle$ with \cup and \cap being the usual set operators.

The proof relying on showing that there exists a function $\phi : \Phi \rightarrow \Sigma$, which is a homomorphism, defined as $\phi(p[\beta]) = \{(x, y) \in \mathbb{R}^2 \mid p(x, y)\}$ and this proof is provided in appendix A.

As the structure Σ as introduced above forms a *Sigma Algebra* (see [Br05b]), it can have a *measure*

$$a_s : \Sigma \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$$

defined upon. Relying on this defined measure, the area function a can be defined as follows:

$$a(g[\beta]) := a_s(\phi(p_g[\beta])).$$

Distance between map symbols

Another concept to be introduced in this section is the concept of *distance* between two map symbol instances. A *distance* function can be defined as follows:

$$d : \mathbb{I}(\mathbb{G})^2 \rightarrow \mathbb{R}_0^+.$$

Relying on the structure Σ introduced above, we can define a distance d_s between two sets of points \mathbb{X}_1 and \mathbb{X}_2 from the \mathbb{R}^2 as follows:

$$d_s(\mathbb{X}_1, \mathbb{X}_2) = \min_{(x_i, y_i) \in \mathbb{X}_i} \left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \right).$$

Having provided two map symbol instances $g_1[\beta_1]$ and $g_2[\beta_2]$, we can define the sets as follows:

$$\mathbb{X}_1 = \phi(p_{g_1}[\beta_1]) \text{ and } \mathbb{X}_2 = \phi(p_{g_2}[\beta_2]),$$

thereby providing a definition for the distance function d :

$$d(g_1[\beta_1], g_2[\beta_2]) := d_s(\phi(p_{g_1}[\beta_1]), \phi(p_{g_2}[\beta_2])).$$

Above, only an *euclidean* distance measurement was introduced, but in a similar way concepts like the *x-specific* and the *y-specific* distance between map symbols can be defined.

Representing map symbol instances on the drawing surface

This section explains the representation of map symbol instances on the drawing surface. Let subsequently be $\mathbb{S} \subseteq \mathbb{I}(\mathbb{G})$ the set of map symbol instances to be represented. From there, we define two functions $p_{\mathbb{S}} : \mathbb{R}^2 \rightarrow \mathbb{B}$ and $r_{\mathbb{S}} : \mathbb{R}^2 \rightarrow \mathbb{P}$, where the first one evaluates to true, if there is at least one map symbol instance at the given point, and the second one indicates, which presentation information is valid for which point. Concerning the presentation information especially the *visibility* attribute is also of interest. The two functions are then defined as follows:

$$p_{\mathbb{S}}(x, y) = \bigvee_{g_i[\beta_j] \in \mathbb{S}} p_{g_i}[\beta_j](x, y)$$

$$r_{\mathbb{S}}(x, y) = l\left(\biguplus_{g_i[\beta_j] \in \mathbb{S} \wedge g_i[\beta_j]_{|visibility=true}} (r_{g_i}[\beta_j](x, y), g_i[\beta_j]_{|zIndex})\right).$$

4.2.5 Map Symbols - Examples

Having introduced the general concepts of a map symbol definition above, these concepts are subsequently illustrated in their usage by defining several exemplary map symbols. Subsequently, text related attributes outlined in the object oriented visualization model are not addressed, as they would greatly rise complexity of the terms presented, while not providing substantial interesting information. While the definition of map symbols is basically built on a mathematical formalization (i.e. the functions p and r), it may be possible to simplify the definition by utilizing concepts for composing map symbols out of primitive, predefined map symbols. Such a concept is not part of the visualization model described in this report, but is alluded to in the outlook in chapter 6.

Ellipse

The Ellipse (see Figure 4.12) map symbol is defined as the class **Ellipse** (see Figure 4.13), holding among others attributes for the coordinates of the central point x and y , attributes for the dimensions of the axes **semiX** and **semiY**, and attributes containing information about fill color **fillColor** and border color **borderColor**. The predicate p_{Ellipse} and the function r_{Ellipse} can therefore be defined as follows:

$$\begin{aligned} p_{\text{Ellipse}}(X, Y, x, y, \text{semiX}, \text{semiY}, \text{fillColor}, \text{borderColor}) &:= \\ \left(\frac{(X-x)^2}{\text{semiX}^2} + \frac{(Y-y)^2}{\text{semiY}^2} \right) &\leq 1 \\ \\ r_{\text{Ellipse}}(X, Y, x, y, \text{semiX}, \text{semiY}, \text{fillColor}, \text{borderColor}) &:= \\ \begin{cases} \text{fillColor} & \text{for } \left(\frac{(X-x)^2}{\text{semiX}^2} + \frac{(Y-y)^2}{\text{semiY}^2} \right) < 1 \\ \text{borderColor} & \text{for } \left(\frac{(X-x)^2}{\text{semiX}^2} + \frac{(Y-y)^2}{\text{semiY}^2} \right) = 1 \\ \epsilon & \text{otherwise} \end{cases} \end{aligned}$$

Rectangle

The Rectangle (see Figure 4.14) map symbol can be defined as the class **Rectangle** (see Figure 4.15), holding among others attributes for the coordinates of the central point x and y , attributes for the dimensions of the rectangle **width** and **height**, and attributes containing information about fill color **fillColor** and border color **borderColor**. The predicate $p_{\text{Rectangle}}$ and the function $r_{\text{Rectangle}}$ can therefore be expressed as follows

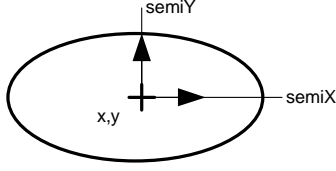


Figure 4.12: An instance of the ellipse map symbol (indications of radii and center added)

Ellipse
zIndex : Integer
visibility : Boolean
x : Real
y : Real
semiX : Real
semiY : Real
borderColor : Color
fillColor : Color
text : String [0..1]
textColor : Color [0..1]

Figure 4.13: The ellipse class (inherited attributes are also shown)

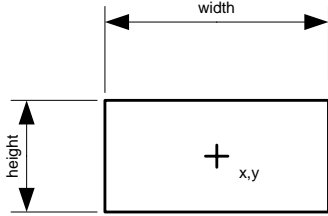


Figure 4.14: An instance of the rectangle map symbol (indications of width, height, and center added)

Rectangle
zIndex : Integer
visibility : Boolean
x : Real
y : Real
width : Real
height : Real
borderColor : Color
fillColor : Color
text : String [0..1]
textColor : Color [0..1]

Figure 4.15: The rectangle class (inherited attributes are also shown)

$$p_{Rectangle}(X, Y, x, y, width, height, fillColor, borderColor) := (|X - x| \leq width/2) \wedge (|Y - y| \leq height/2)$$

$$r_{Rectangle}(X, Y, x, y, width, height, fillColor, borderColor) := \begin{cases} fillColor & \text{for } (|X - x| < width/2) \wedge (|Y - y| < height/2) \\ borderColor & \text{for } (|X - x| = width/2) \vee (|Y - y| = height/2) \\ \epsilon & \text{otherwise} \end{cases}$$

Polyline

The Polyline (see Figure 4.16), a linear map symbol, is defined as the class **Polyline** (see Figure 4.17), holding attributes containing the **x** and **y** coordinates of all bendpoints (including start and end points, too), as well as an attribute for the line color **lineColor**. The attributes **x** and **y** are therein sequences (see appendix B), further constrained to identical lengths, such that $x \rightarrow \text{length}() = y \rightarrow \text{length}()$ holds. The predicate $p_{Polyline}$ and the function $r_{Polyline}$ can therefore be expressed as follows.

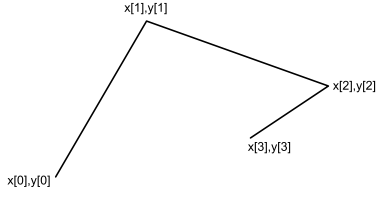


Figure 4.16: An instance of the polyline map symbol (indications of bend-points added)

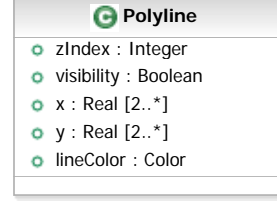


Figure 4.17: The polyline class (inherited attributes are also shown)

$$pPolyline(X, Y, x, y, lineColor) := \exists \eta \in [0..1], i \in 0, \dots, n-1 : (X = \eta(x(i+1) - x(i)) + x(i)) \wedge (Y = \eta(y(i+1) - y(i)) + y(i))$$

$$rPolyline(X, Y, x, y, lineColor) := \begin{cases} lineColor & \text{for } p(X, Y, x, y, lineColor) \\ \epsilon & \text{otherwise} \end{cases}$$

Presentation vs. Map Symbol definition

The map symbols specified above are described in an idealized manner. This is especially obvious when regarding the presentation of lines in general and borders in special. These elements are above defined to be strictly one dimensional and would therefore not be visible in an actual visualization. While any kind of drawing surface is most likely to be only of a finite resolution, it is assumed that each point is actually mapped to at least one display unit, e.g. a *pixel*. Therefore, one should think of lines and borders extended to employ at least one display unit, such that they are actually visible.

4.2.6 Visualization Rules - Definition

Introduced in the description of the object-oriented Visualization Model in section 4.1, a visualization rule is a class that has at least one association to a **VisualizationObject** class and is a subclass of **VisualizationRule**. Instances of the rules influence the instances of subclasses of **VisualizationObject** linked via the associations. In accordance to section 4.1, two distinct kinds of visualization rules have to be distinguished, concerning the effect they exert on **VisualizationObject** instances: subclasses of class **ConstraintVisualizationRule** making up the set \mathbb{D}_{con} and subclasses of class **TargetVisualizationRule** making up the set \mathbb{D}_{tar} . Regarding these sets, the following holds:

$$\begin{aligned} \mathbb{D}_{con} &\subseteq \mathbb{D}, \\ \mathbb{D}_{tar} &\subseteq \mathbb{D}, \\ \mathbb{D}_{con} \cap \mathbb{D}_{tar} &= \emptyset, \text{ and} \\ \mathbb{D}_{con} \cup \mathbb{D}_{tar} &= \mathbb{D}. \end{aligned}$$

These distinct kinds of visualization rules can be, in accordance to the definition in section 4.1, understood as follows:

Constraint visualization rules are rules that can be either fulfilled or not. These rules are augmented in the general mathematical formalization by predicates that take the linked visualization object instances and possibly parameters as their arguments. The name is motivated by their similarity to constraints in optimization problems.

Target visualization rules can be fulfilled to a certain degree. These rules are augmented by functions, that take the visualization object instances as their arguments, also possibly together with some parameters. Also here the name is motivated by the similarity to an element of an optimization problem: the target function.

The functions mentioned above are provided by a function λ for which the following holds:

$$\lambda := \begin{cases} d \mapsto \lambda_{con} \in \left(\prod_{a_i \in \mathbb{A}(d)} \mathbb{W}^A(a_i) \times \prod_{v_i \in \mathbb{V}(d)} \mathbb{W}^V(v_i) \rightarrow \mathbb{B} \right) & , \text{ if } d \in \mathbb{D}_{con} \\ d \mapsto \lambda_{tar} \in \left(\prod_{a_i \in \mathbb{A}(d)} \mathbb{W}^A(a_i) \times \prod_{v_i \in \mathbb{V}(d)} \mathbb{W}^V(v_i) \rightarrow \mathbb{F} \right) & , \text{ if } d \in \mathbb{D}_{tar} \end{cases}$$

The function $\lambda(d)$ is further shorthanded as λ_d .

In above definitions, a set \mathbb{F} is introduced, which has to have an ordering relationship \leq defined upon, such that the structure $\langle \mathbb{F}, \leq \rangle$ constitutes a weak partial order⁸. Therefore, there may be cases, in which the application of a target visualization rule on different visualizations may not decide, which one is the *total optimal* visualization. Instead, there might be differing visualizations satisfying the given target visualization rule to values, which are in the given structure not comparable. The smaller the value from \mathbb{F} is for a specific rule, the better it is fulfilled, given that a comparison is possible.

Additionally, if more than one target visualization rule is applied in a specific symbolic model, the decision, which visualization is *best*, is even more complex. Even if each visualization rule instance evaluates to a total order, no *total optimal* visualization can be found, but only an *efficient set* of visualizations each having its specific advantages and disadvantages. This set contains all visualizations, for which no dominating visualization can be found. A dominating visualization has at least one visualization rule instance fulfilled to a better degree (lower value) and all other visualization rule instances to the same degree. From the efficient set formed by these visualizations no element can be prominently chosen without introducing an arbitrary weighting of the target visualization rule instances and enhancing the partial order structure $\langle \mathbb{F}, \leq \rangle$ to a total ordered one.

Instantiation of visualization rules

An instantiation of a visualization rule class d is connected to assignments of the variables $\mathbb{V}(d)$ of the respective predicate and the respective association ends, $\mathbb{A}(d)$, which is according to the instantiation of visualization objects expressed by two assignments:

- $\beta : v_i \in \mathbb{V}(d) \mapsto w_j \in \mathbb{W}^V(v_i)$ assigns each attribute v_i of the visualization rule to a value from $\mathbb{W}(v_i)$ and

⁸A *weak partial order* is a binary relation that is transitive, antisymmetric, and reflexive.

- $\alpha : a_i \in \mathbb{A}(d) \mapsto w_j \in \mathbb{W}^A(a_i)$ points the association ends of the visualization rule class associations to the respective visualization object instances.

Therefore, a visualization rule instance can be denoted via the concept of assignment as $d[\beta][\alpha]$, having an augmenting predicate $\lambda_d[\beta][\alpha]$, in which the respective variables and association ends are assigned the values indicated by α and β .

4.2.7 Visualization Rules - Examples

The following examples of visualization rules are the ones shown in Figures 4.6, 4.7, and 4.9 that have been introduced to address the visualizations in chapter 3. They are subsequently organized into constraint and target visualization rules.

Constraint visualization rules

IdentityOfProjection: This visualization rule builds on the concept of *projection* as defined in section 4.2.2. The visualization rule **IdentityOfProjection** between two visualization object instances regarding a common attribute means that the projection regarding this attribute results in identical values for the two visualization object instances. The visualization rule **IdentityOfProjection** has a predicate defined that assigns a boolean value to every combination of two visualization object instances and an attribute $v_{targetVariable}$ indicating a variable that the respective visualization objects share⁹:

$$\begin{aligned} \lambda_{d_{ProjId}} : \mathbb{W}^A(a_{ruled}) \times \mathbb{W}(v_{targetVariable}) &\rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{ruled}) &= \{p \in \mathcal{P}(\mathbb{I}(\mathbb{O})) \mid |p| = 2\}. \end{aligned}$$

The predicate of the rule itself is defined as follows:

$$\lambda_{d_{ProjId}}(\{o_1[\beta_1], o_2[\beta_2]\}, w_{targetVariable}) := (o_1[\beta_1]_{|w_{targetVariable}} = o_2[\beta_2]_{|w_{targetVariable}}).$$

Ordering: An ordering between two visualization objects instances may be established regarding an arbitrary common attribute, which the respective visualization objects share and on which an order relation (subsequently written as \leq) is established that is at least a linear preorder (i.e. is both total and transitive).

Formally, the ordering is augmented by a predicate that assigns a Boolean value to every combination of two visualization object instances and an attribute the visualization objects share:

$$\begin{aligned} \lambda_{d_{Ord}} : \mathbb{W}^A(a_{ruledLower}) \times \mathbb{W}^A(a_{ruledUpper}) \times \mathbb{W}(v_{targetVariable}) &\rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{ruledLower}) &= \mathbb{I}(\mathbb{O}), \text{ and} \\ \mathbb{W}^A(a_{ruledUpper}) &= \mathbb{I}(\mathbb{O}). \end{aligned}$$

⁹The operator $\mathcal{P}(\mathbb{X})$ applied on a set \mathbb{X} is called *power set* of \mathbb{X} , which is defined as a set containing all subsets of \mathbb{X} , even the trivial ones - \mathbb{X} itself and the empty set \emptyset . Further information about which structures are chosen for which kind of attribute in the object-oriented model is provided in appendix B

Besides this ordering, there is also a strict ordering, $d_{StrictOrd}$. For strict ordering, a relation $<$ being transitive, trichotomous, and asymmetric, has to be defined on the respective attribute's values.

The predicates for the orderings are defined as follows:

$$\lambda_{d_{Ord}}(o_{ruledLower}[\beta_1], o_{ruledUpper}[\beta_2], w_{targetVariable}) := (o_{ruledLower}[\beta_1]_{|w_{targetVariable}} \leq o_{ruledUpper}[\beta_2]_{|w_{targetVariable}})$$

and

$$\lambda_{d_{StrictOrd}}(o_{ruledLower}[\beta_1], o_{ruledUpper}[\beta_2], w_{targetVariable}) := (o_{ruledLower}[\beta_1]_{|w_{targetVariable}} < o_{ruledUpper}[\beta_2]_{|w_{targetVariable}}).$$

Layer: A layer represents a grouping of any number of map symbol instances, establishing additional constraints on those symbols regarding one specific attribute. Distinguishing the different specializations of **Layer** by the affected attribute, the following subclasses are introduced:

The **ZLayer** has a predicate defined that assigns a boolean value to a set of map symbol instances and a given target **zIndex**:

$$\lambda_{d_{ZLayer}} : \mathbb{W}^A(a_{layeredSymbols}) \times \mathbb{W}(v_{zIndex}) \rightarrow \mathbb{B} \text{ with } \mathbb{W}^A(a_{layeredSymbols}) = \mathcal{P}(\mathbb{I}(\mathbb{G})) \text{ and } \mathbb{W}(v_{zIndex}) = \mathbb{N}_0.$$

The predicate for the **ZLayer** is defined as follows ($\mathbb{Y} \in \mathcal{P}(\mathbb{I}(\mathbb{G}))$):

$$\lambda_{d_{ZLayer}}(\mathbb{Y}, w_{zIndex}) := \bigwedge_{y_i[\beta_i] \in \mathbb{Y}} (y_i[\beta_i]_{|zIndex} = w_{zIndex}).$$

The **VisibilityLayer** has a predicate defined that assigns a Boolean value to a set of map symbol instances and a given target **visibility**:

$$\lambda_{d_{visibilityLayer}} : \mathbb{W}^A(a_{layeredSymbols}) \times \mathbb{W}(v_{visibility}) \rightarrow \mathbb{B} \text{ with } \mathbb{W}^A(a_{layeredSymbols}) = \mathcal{P}(\mathbb{I}(\mathbb{G})) \text{ and } \mathbb{W}(v_{visibility}) = \{true, false\}.$$

The predicate for the **VisibilityLayer** is defined as follows ($\mathbb{Y} \in \mathcal{P}(\mathbb{I}(\mathbb{G}))$):

$$\lambda_{d_{visibilityLayer}}(\mathbb{Y}, w_{visibility}) := \bigwedge_{y_i[\beta_i] \in \mathbb{Y}} (y_i[\beta_i]_{|visibility} = w_{visibility}).$$

Intersection: As it is shown in Figure 4.6, different kinds of intersection are distinguished in the visualization model, which all act on planar map symbols. Therefore all intersection rules are augmented as follows:

$$\begin{aligned} \lambda_{d_{*Intersect}} : \mathbb{W}^A(a_{intersecting}) \times \mathbb{W}^A(a_{intersected}) &\rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{intersecting}) &= \mathbb{I}(\mathbb{G}_F), \text{ and} \\ \mathbb{W}^A(a_{intersected}) &= \mathbb{I}(\mathbb{G}_F). \end{aligned}$$

It has to be distinguished whether the intersection is regarding the x-coordinate, the y-coordinate or the area of a planar map symbol.

The **FullXSpecificIntersection** means that every point of the intersecting map symbol instance shares his x-coordinate with at least one point of the intersected map symbol instance. Formally, this is expressed by the augmenting predicate¹⁰:

$$\lambda_{d_{XIntSec}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]) := \forall x \in \mathbb{R} \exists y_1 \in \mathbb{R}, y_2 \in \mathbb{R} : p_{g_{intersecting}}[\beta_1](x, y_1) \Rightarrow p_{g_{intersected}}[\beta_2](x, y_2).$$

The **FullYSpecificIntersection**, $d_{YIntSec}$, means that every point of the first map symbol instance shares its y-coordinate with at least one point from the second map symbol instance.

The definition of this predicate is similar to its x-specific counterparts, with x and y changed respectively. Thus, the predicate augmentation of a **FullYSpecificIntersection** is:

$$\lambda_{d_{YIntSec}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]) := \forall y \in \mathbb{R} \exists x_1 \in \mathbb{R}, x_2 \in \mathbb{R} : p_{g_{intersecting}}[\beta_1](x_1, y) \Rightarrow p_{g_{intersected}}[\beta_2](x_2, y).$$

The **Nesting** relationships between planar map symbol instances demands that the first map symbol, $g_{intersecting}$, is placed totally inside the second one, $g_{intersected}$. This relationship is augmented by a predicate, which assigns a boolean value to every combination of two planar map symbol instances:

$$\begin{aligned} \lambda_{d_{Nesting}} : \mathbb{W}^A(a_{intersecting}) \times \mathbb{W}^A(a_{intersected}) &\rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{intersecting}) &= \mathbb{I}(\mathbb{G}_F), \text{ and} \\ \mathbb{W}^A(a_{intersected}) &= \mathbb{I}(\mathbb{G}_F). \end{aligned}$$

The predicate is defined as:

$$\begin{aligned} \lambda_{d_{Nesting}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]) &:= \\ \forall (x, y) \in \mathbb{R}^2 &p_{g_{intersecting}}[\beta_1](x, y) \Rightarrow p_{g_{intersected}}[\beta_2](x, y). \end{aligned}$$

PlanarNonIntersection: **PlanarNonIntersection** demands that two instances of planar map symbols do not have any points in common, which is defined by the augmenting predicate that assigns a boolean value to every combination of two planar map symbol instances:

$$\begin{aligned} \lambda_{d_{PlanarNonIntersection}} : \mathbb{W}^A(a_{nonintersectingSymbols}) &\rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{nonintersectingSymbols}) &= \{p \in \mathcal{P}(\mathbb{I}(\mathbb{G}_F)) \mid |p| = 2\}. \end{aligned}$$

The predicate is defined as:

$$\lambda_{d_{PlanarNonIntersection}}(\{g_1[\beta_1], g_2[\beta_2]\}) := \forall (x, y) \in \mathbb{R}^2 : \neg(p_{g_1}[\beta_1](x, y) \wedge p_{g_2}[\beta_2](x, y)).$$

¹⁰ $\Rightarrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}; a \Rightarrow b \iff \neg a \vee b.$

XSequence: **XSequence** demands that two or more instances of planar map symbols are arranged from left to right in a way that neighboring symbols do not have common points. This class is augmented with a predicate that assigns a Boolean value to every sequence of two or more planar map symbol instances¹¹:

$$\lambda_{d_{XSequence}} : \mathbb{W}^A(a_{ruled}) \rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{ruled}) = \bigcup_{n=2}^{\infty} \mathcal{S}_n(\mathbb{I}(\mathbb{G}_F)).$$

The predicate is defined as:

$$\lambda_{d_{XSequence}}(\{i \mapsto g_i[\beta_i] \mid 0 \leq i < n - 1 \wedge n \geq 2\}) := \bigwedge_{i=0}^{n-2} (\max\{x \in \mathbb{R} \mid \exists y \in \mathbb{R} : p_{g_i}[\beta_i](x, y)\} < \min\{x \in \mathbb{R} \mid \exists y \in \mathbb{R} : p_{g_{i+1}}[\beta_{i+1}](x, y)\}).$$

YSequence: **YSequence** demands that two or more instances of planar map symbol are arranged from top to bottom in a way that neighboring symbols do not have common points. This class is augmented with a predicate that assigns a Boolean value to every sequence of two of more planar map symbol instances:

$$\lambda_{d_{YSequence}} : \mathbb{W}^A(a_{ruled}) \rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{ruled}) = \bigcup_{n=2}^{\infty} \mathcal{S}_n(\mathbb{I}(\mathbb{G}_F)).$$

The predicate is defined as:

$$\lambda_{d_{YSequence}}(\{i \mapsto g_i[\beta_i] \mid 0 \leq i < n - 1 \wedge n \geq 2\}) := \bigwedge_{i=0}^{n-2} (\max\{y \in \mathbb{R} \mid \exists x \in \mathbb{R} : p_{g_i}[\beta_i](x, y)\} < \min\{y \in \mathbb{R} \mid \exists x \in \mathbb{R} : p_{g_{i+1}}[\beta_{i+1}](x, y)\}).$$

Attachment: The **Attachment** relationship demands, that a planar map symbol instance $f_{attachmentTarget}[\beta_1]$ has a linear map symbol instance $l_{attachedSymbol}[\beta_2]$ attached to it, while attached therein is regarded as the two instances sharing a point which is one of the endpoints of the linear map symbol instance. The relationship is augmented by a predicate, which assigns a Boolean value to every combination of a linear and a planar map symbol instance:

$$\lambda_{d_{Attachment}} : \mathbb{W}^A(a_{attachmentTarget}) \times \mathbb{W}^A(a_{attachedSymbol}) \rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{attachmentTarget}) = \mathbb{I}(\mathbb{G}_F), \text{ and} \\ \mathbb{W}^A(a_{attachedSymbol}) = \mathbb{I}(\mathbb{G}_L).$$

The predicate is defined as:

$$\lambda_{d_{Attachment}}(f_{attachmentTarget}[\beta_1], l_{attachedSymbol}[\beta_2]) := \exists (x_e, y_e) \in \mathbb{R}^2 : \\ pf_{attachmentTarget}[\beta_1](x_e, y_e) \wedge pl_{attachedSymbol}[\beta_2](x_e, y_e) \wedge \chi(\sigma(pl_{attachedSymbol}[\beta_2], (x_e, y_e)))$$

¹¹Therein $\mathcal{S}_n(\mathbb{X})$ is the set of sequences over a given set \mathbb{X} , mapping the natural numbers $0..n - 1$ to elements from \mathbb{X} . For further details of this definition see appendix B.

LineNonIntersection: The **LineNonIntersection** relationship demands, that two linear map symbol instances do not have any common points except for their endpoints, which may be common but do not have to. The relationship is augmented by a predicate, which assigns a boolean value to every combination of two linear map symbol instances:

$$\lambda_{d_{\text{LineNonIntersection}}} : \mathbb{W}^A(a_{\text{linear}}) \rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{\text{linear}}) = \{p \in \mathcal{P}(\mathbb{I}(\mathbb{G}_L)) \mid |p| = 2\}.$$

The predicate is defined as:

$$\lambda_{d_{\text{LineNonIntersection}}}(\{l_1[\beta_1], l_2[\beta_2]\}) := \forall(x, y) \in \mathbb{R}^2 : (p_{l_1}[\beta_1](x, y) \wedge p_{l_2}[\beta_2](x, y)) \Rightarrow \\ (\chi(\sigma(p_{l_1}[\beta_1], (x, y))) \wedge \chi(\sigma(p_{l_2}[\beta_2], (x, y)))).$$

PlanarLinearNonIntersection: The **PlanarLinearNonIntersection** relationship demands, that a planar map symbol instance $f_{\text{planar}}[\beta_1]$ does not have any common point with a linear map symbol instance $l_{\text{linear}}[\beta_2]$. This relationship is augmented by a predicate, which assigns a Boolean value to every combination of a linear and a planar map symbol instance:

$$\lambda_{d_{\text{PlanarLinearNonIntersection}}} : \mathbb{W}^A(a_{\text{planar}}) \times \mathbb{W}^A(a_{\text{linear}}) \rightarrow \mathbb{B} \text{ with} \\ \mathbb{W}^A(a_{\text{planar}}) = \mathbb{I}(\mathbb{G}_F), \text{ and} \\ \mathbb{W}^A(a_{\text{linear}}) = \mathbb{I}(\mathbb{G}_L).$$

The predicate is defined as:

$$\lambda_{d_{\text{PlanarLinearNonIntersection}}}(f_{\text{planar}}[\beta_1], l_{\text{linear}}[\beta_2]) := \forall(x, y) \in \mathbb{R}^2 : \\ \neg(p_{f_{\text{planar}}}[\beta_1](x, y) \wedge p_{l_{\text{linear}}}[\beta_2](x, y)).$$

Target Visualization Rules

The rules which have been defined and exemplified above, all are constraint visualization rules. Subsequently, several examples of target visualization rules are introduced. Thereby it has to be noted, that for none of these rules an actual value for the evaluation of the rule's function is given. Instead of this, relationships are proposed, which define how the evaluated values of the rule function are demanded to behave under specific conditions.

Insist: The **Insist** rule expresses that the value of a given variable $v_{\text{targetVariable}}$ of a visualization object instance is as close as possible to a given target value $w_{\text{targetValue}} \in \mathbb{W}^V(v_{\text{targetVariable}})$. The visualization rule is augmented by a function

$$\lambda_{d_{\text{Insist}}} : \mathbb{W}^A(a_{\text{ruledObject}}) \times \mathbb{W}^V(v_{\text{targetVariable}}) \times \mathbb{W}^V(v_{\text{targetValue}}) \rightarrow \mathbb{F} \text{ with} \\ \mathbb{W}^A(a_{\text{ruledObject}}) = \mathbb{I}(\mathbb{O}).$$

Regarding this target visualization rule's values for two different visualization object instances $o_1[\beta_1]$ and $o_2[\beta_2]$ the follow expression holds:

$$\lambda_{d_{Insist}}(o_1[\beta_1], w_{targetVariable}, w_{targetValue}) \leq \lambda_{d_{Insist}}(o_2[\beta_2], w_{targetVariable}, w_{targetValue}) \Leftrightarrow |o_1[\beta_1]_{w_{targetVariable}} - w_{targetValue}| \leq |o_2[\beta_2]_{w_{targetVariable}} - w_{targetValue}|.$$

DistanceFixationEuclidean: The **DistanceFixationEuclidean** rule expresses that the distance between two planar map symbol instances (computed by the distance function from section 4.2.3) is as close as possible to or arbitrarily higher than a given value $w_{minimumDistance}$, which is supplied as a visual variable of the visualization rule instance. This is formally expressed as augmentation by a function

$$\lambda_{d_{DistanceFixationEuclidean}} : \mathbb{W}^A(a_{ruled}) \times \mathbb{W}^V(v_{minimumDistance}) \rightarrow \mathbb{F} \text{ with } \mathbb{W}^A(a_{ruled}) = \{p \in \mathcal{P}(\mathbb{I}(\mathbb{G})) \mid |p| = 2\}$$

Therefore, the following expression can be made regarding the target visualization rule values for two different sets of map symbol instances $\{(g_1[\beta_1], g_2[\beta_2])\}$ and $\{(g_3[\beta_3], g_4[\beta_4])\}$:

$$\begin{aligned} \lambda_{d_{DistanceFixationEuclidean}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{minimumDistance}) &\leq \\ \lambda_{d_{DistanceFixationEuclidean}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{minimumDistance}) &\Leftrightarrow \\ \max\{0, w_{minimumDistance} - d(g_1[\beta_1], g_2[\beta_2])\} &\leq \max\{0, w_{targetValue} - d(g_3[\beta_3], g_4[\beta_4])\}. \end{aligned}$$

The functions for **DistanceFixationHorizontal** and **DistanceFixationVertical** can be defined similarly, relying on the respective distance functions from section 4.2.4.

AreaMinimization: The **AreaMinimization** rule expresses that the area of a planar map symbol instance (computed by the area function introduced in section 4.2.3) should be as small as possible. The visualization rule is augmented by a function

$$\lambda_{d_{AreaMinimization}} : \mathbb{W}^A(a_{ruled}) \rightarrow \mathbb{F} \text{ with } \mathbb{W}^A(a_{ruled}) = \mathbb{I}(\mathbb{G}_F).$$

Regarding this target visualization rule's values for two different planar map symbol instances $f_1[\beta_1]$ and $f_2[\beta_2]$ the follow expression holds:

$$\lambda_{d_{AreaMinimization}}(f_1[\beta_1]) \leq \lambda_{d_{AreaMinimization}}(f_2[\beta_2]) \Leftrightarrow a(f_1[\beta_1]) \leq a(f_2[\beta_2]).$$

4.3 Mitigations of the Visualization model

Having provided definitions and examples for the concepts of the visualization model both on an object-oriented and on a general mathematical level, calculating a visualization satisfying the demands imposed by the constraint visualization rules and achieving optimal results for the target visualization rules cannot efficiently be executed relying on those definitions. Concretizations of the visualization model concepts have to be introduced, providing an easily calculable model version. Although many different concretization possibilities might exist, we provide elementary concepts that are the basis of many concretizations. These concepts are introduced and defined subsequently. We further distinguish these mitigations in respect to whether solely planar or also linear map symbol instances are involved.

4.3.1 Mitigations of planar map symbols

Two core concepts relevant to possible concretizations of the visualization model in respect to planar map symbols are the *closure* and the *filling* of planar map symbol instances. Below we introduce these concepts, subsequently operationalizing the demands of visualization rules to demands on these concepts.

Definitions of closure and filling

Postponing an example for a specific closure and a specific filling to section 4.4, we define these concepts abstractly as follows:

A *closure* of a planar map symbol instance $g[\beta_1]$ (with $g \in \mathbb{G}_F$) is a predicate $p_{c(g)}[\beta_1] : \mathbb{R}^2 \rightarrow \mathbb{B}$ satisfying the following conditions:

1. $\forall (x, y) \in \mathbb{R}^2 : p_g[\beta_1](x, y) \Rightarrow p_{c(g)}[\beta_1](x, y)$
2. $(\forall (x, y) \in \mathbb{R}^2 : p_{g_1}[\beta_1](x, y) \Rightarrow p_{g_2}[\beta_2](x, y)) \Rightarrow (\forall (x, y) \in \mathbb{R}^2 : p_{c(g_1)}[\beta_1](x, y) \Rightarrow p_{c(g_2)}[\beta_2](x, y))$
3. $\forall (x, y) \in \mathbb{R}^2 : p_{c(c(g))}[\beta](x, y) \iff p_{c(g)}[\beta](x, y).$

Condition (1) can be understood that for every point $(x, y) \in \mathbb{R}^2$ that satisfies the map symbol instance's predicate $p_g[\beta_1]$ also the closure predicate is satisfied. Nevertheless, there might be points $(x, y) \in \mathbb{R}^2$ that only satisfy the closure predicate. Condition (2) imposes a kind of *monotony* regarding the closures of planar map symbol instances: if an instance of a (planar) map symbol is nested in another instance of a planar map symbol, the nesting also holds for the respective closures. Condition (3) imposes an *idempotency* regarding the closures, such that the closure of the closure of a planar map symbol instance is the closure itself.

Similarly, a *filling* of a planar map symbol instance $g[\beta_1]$ (with $g \in \mathbb{G}_F$) is defined as a predicate

$p_{f(g)}[\beta_1] : \mathbb{R}^2 \rightarrow \mathbb{B}$ satisfying the following condition:

1. $\forall (x, y) \in \mathbb{R}^2 : p_{f(g)}[\beta_1](x, y) \Rightarrow p_g[\beta_1](x, y)$
2. $(\forall (x, y) \in \mathbb{R}^2 : p_{g_1}[\beta_1](x, y) \Rightarrow p_{g_2}[\beta_2](x, y)) \Rightarrow \neg(\forall (x, y) \in \mathbb{R}^2 : p_{f(g_2)}[\beta_2](x, y) \Rightarrow p_{f(g_1)}[\beta_1](x, y))$
3. $\forall (x, y) \in \mathbb{R}^2 : p_{f(f(g))}[\beta](x, y) \iff p_{f(g)}[\beta](x, y).$

Condition (1) means that for every point $(x, y) \in \mathbb{R}^2$ that satisfies the filling predicate also the map symbol instance's predicate $p_g[\beta_1]$ is satisfied. Nevertheless, there might be points $(x, y) \in \mathbb{R}^2$ that only satisfy the map symbol instance's predicate $p_g[\beta_1]$. Condition (2) is similar to the *monotony* demand of the closure, while condition (3) demands *idempotency* regarding the fillings.

Relying on these definitions of closure and filling, we demand functions $p_{c(g)}$ and $p_{f(g)}$ to exist that supply for any map symbol g a *parametrized* version of the respective predicates

for closure and filling. A binding $\beta \in \mathbb{E}_\beta(g)$ may then be applied to these predicates leading to $p_{c(g)}[\beta]$ and $p_{f(g)}[\beta]$ respectively. These functions are subsequently used to facilitate some reflections about auxiliary functions introduced on planar map symbols.

Auxiliary concepts on closure and filling

In section 4.2.4 we have introduced several auxiliary concepts on map symbol instances, especially on instances of planar ones. Some of these concepts are subsequently mapped to closure and filling, providing insights relevant for defining a solvable version of the model.

Area function The area function a is defined for planar map symbol instances in section 4.2.4, based on a measure on the set of points belonging to a map symbol instance. Regarding the area occupied by a map symbol instance $g[\beta]$, its closure predicate $p_{c(g)}[\beta]$, and its filling predicate $p_{f(g)}[\beta]$, the following expression holds:

$$a_s(\phi(p_{f(g)}[\beta])) \leq a_s(\phi(p_g[\beta])) = a(g[\beta]) \leq a_s(\phi(p_{c(g)}[\beta])).$$

This can easily be proven, defining sets \mathbb{X} , \mathbb{X}_f , and \mathbb{X}_c as follows:

$$\mathbb{X} = \phi(p_g[\beta]), \mathbb{X}_f = \phi(p_{f(g)}[\beta]), \text{ and } \mathbb{X}_c = \phi(p_{c(g)}[\beta]).$$

From the definition of a filling, we know that $\forall(x, y) \in \mathbb{R}^2 : p_{f(g)}[\beta](x, y) \Rightarrow p_g[\beta](x, y)$ holds, whereas there might exist points only satisfying the map symbol instance's predicate. Therefore, we can define \mathbb{X} as unification of two *disjoint* sets $\mathbb{X} = \mathbb{X}_{\bar{f}} \cup \mathbb{X}_f$. Relying on the definition of a measure we can thereby prove that:

$$a_s(\mathbb{X}) = a_s(\mathbb{X}_{\bar{f}}) + a_s(\mathbb{X}_f) \geq a_s(\mathbb{X}_f).$$

We can prove a similar inequality for the closure, as we know, from the definition of a closure, that $\forall(x, y) \in \mathbb{R}^2 : p_g[\beta_1](x, y) \Rightarrow p_{c(g)}[\beta_1](x, y)$ holds. Therefore \mathbb{X}_c can be defined as unification of two *disjoint* sets $\mathbb{X}_c = \mathbb{X}_{\bar{c}} \cup \mathbb{X}$. Relying on the definition of a measure we can then prove that:

$$a_s(\mathbb{X}_c) = a_s(\mathbb{X}_{\bar{c}}) + a_s(\mathbb{X}) \geq a_s(\mathbb{X}).$$

This can be summarized into $a_s(\mathbb{X}_f) \leq a_s(\mathbb{X}) \leq a_s(\mathbb{X}_c)$, from which the above inequality follows due to the definitions of \mathbb{X} , \mathbb{X}_f , and \mathbb{X}_c .

Distance function The distance function d has been introduced on arbitrary map symbol instances in section 4.2.4. Here, we present some reflections on a restricted version of the distance function, only mapping instances of planar map symbols. For this version, one can easily prove that the following expression holds:

$$\begin{aligned} d_s(\phi(p_{f(g_1)}[\beta_1]), \phi(p_{f(g_2)}[\beta_2])) &\geq d_s(\phi(p_{g_1}[\beta_1]), \phi(p_{g_2}[\beta_2])) = d(g_1[\beta_1], g_2[\beta_2]) \text{ and} \\ d_s(\phi(p_{g_1}[\beta_1]), \phi(p_{g_2}[\beta_2])) &= d(g_1[\beta_1], g_2[\beta_2]) \geq d_s(\phi(p_{c(g_1)}[\beta_1]), \phi(p_{c(g_2)}[\beta_2])). \end{aligned}$$

A proof of this can rely on the fact, that on the one hand, every point contained in the actual symbol instance is also contained in the instance's closure, but there might be points in the closure that are closer to the other symbol instance's closure than the symbol instance's points. On the other hand every point contained in the instance's filling is also contained in the instance itself, whereas the filling might be stripped of the point responsible for the minimal distance.

Mitigating constraint visualization rules on planar map symbols

Subsequently, we provide mitigations for some of the constraint visualization rules that are outlined in section 4.2.7, and which solely relate instances of planar map symbols. We show, that the fulfillment of a mitigated rule on closures and fillings of the planar map symbol instances employed, is a sufficient prerequisite for the fulfillment of the visualization rule instance on the actual map symbol instances. For the remainder of these rules, which are for the presented model **IdentityOfProjection**, **Ordering**, **ZLayer**, and **VisibilityLayer**, no mitigations are provided, as we regard them to be sufficiently computable for themselves.

Nesting This relationship demands a planar map symbol instance $g_{intersecting}[\beta_1]$ to be nested into a planar map symbol instance $g_{intersected}[\beta_2]$. It can be shown that the following relationship holds, as the subsequent proof demonstrates:

$$(\forall(x, y) \in \setminus^2 p_{c(g_{intersecting})}[\beta_1](x, y) \Rightarrow p_{f(g_{intersected})}[\beta_2](x, y)) \Rightarrow \lambda_{d_{Nesting}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]).$$

The lefthand property $\forall(x, y) \in \setminus^2 p_{c(g_{intersecting})}[\beta_1](x, y) \Rightarrow p_{f(g_{intersected})}[\beta_2](x, y)$ is subsequently referred to as (1). From (1), the first closure property, here denoted as (2), and the first filling property, here denoted as (3), the following can be deduced:

$$(4) \quad \forall(x, y) \in \setminus^2 :$$

$$\underbrace{p_{g_{intersecting}}[\beta_1](x, y) \Rightarrow p_{c(g_{intersecting})}[\beta_1](x, y)}_{(2)} \overset{(1)}{\Rightarrow} \underbrace{p_{f(g_{intersected})}[\beta_2](x, y) \Rightarrow p_{g_{intersected}}[\beta_2](x, y)}_{(3)}$$

Thus, $\forall(x, y) \in \setminus^2 p_{g_{intersecting}}[\beta_1](x, y) \Rightarrow p_{g_{intersected}}[\beta_2](x, y)$ follows, (see the first and the last term of (4)), which is exactly the predicate of $\lambda_{d_{Nesting}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2])$.

As shown in the proof above, operating on closures and fillings instead of the map symbol instances themselves is possible for determining whether a constraint visualization rule instance is fulfilled. Nevertheless, there might be combinations of map symbol instances, where the constraint visualization rule is also fulfilled, while the corresponding rule on the closures and fillings does not indicate that. Therefore fulfillment of constraint visualization rules on closures and fillings is a *sufficient* but not *necessary* condition for the fulfillment of the respective rule on the actual map symbol instances.

For the following rules, only the mitigated versions are provided, while the proofs are left out for brevity. Some of them are presented at our Project Wiki¹².

FullXSpecificIntersection For this rule, as defined in section 4.2.7, it can be shown that the following relationship holds:

$$(\forall x \in \mathbb{R} \exists y_1 \in \mathbb{R}, y_2 \in \mathbb{R} \ p_{c(g_{intersecting})}[\beta_1](x, y_1) \Rightarrow p_{f(g_{intersected})}[\beta_2](x, y_2)) \Rightarrow \lambda_{d_{XIntSec}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]).$$

FullyYSpecificIntersection This relationship has a mitigation similar to the one shown above with **FullXSpecificIntersection**:

$$(\forall y \in \mathbb{R} \exists x_1 \in \mathbb{R}, x_2 \in \mathbb{R} \ p_{c(g_{intersecting})}[\beta_1](x_1, y) \Rightarrow p_{f(g_{intersected})}[\beta_2](x_2, y)) \Rightarrow \lambda_{d_{YIntSec}}(g_{intersecting}[\beta_1], g_{intersected}[\beta_2]).$$

PlanarNonIntersection This relationship demands planar map symbol instances $g_1[\beta_1]$ and $g_2[\beta_2]$ to have no common point. It can be shown that the following relationship holds:

$$\forall (x, y) \in \mathbb{R}^2 : \neg(p_{c(g_1)}[\beta_1](x, y) \wedge p_{c(g_2)}[\beta_2](x, y)) \Rightarrow \lambda_{d_{PlanarNonIntersection}}(\{g_1[\beta_1], g_2[\beta_2]\}).$$

XSequence This relationship demands planar map symbol instances $g_0[\beta_0], \dots, g_n[\beta_n]$ to be arranged from left to right. It can be shown that the following relationship holds:

$$(\forall 0 \leq i < n - 1 : \max\{x \in \mathbb{R} | \exists y \in \mathbb{R} : (p_{c(g_i)}[\beta_i](x, y))\} < \min\{x \in \mathbb{R} | \exists y \in \mathbb{R} : (p_{c(g_{i+1})}[\beta_{i+1}](x, y))\}) \Rightarrow \lambda_{d_{XSequence}}(\{i \mapsto g_i[\beta_i] | 0 \leq i < n - 1\}).$$

YSequence This relationship demands planar map symbol instances $g_0[\beta_0], \dots, g_n[\beta_n]$ to be arranged from top to bottom. It can be shown that the following relationship holds:

$$(\forall 0 \leq i < n - 1 : \max\{y \in \mathbb{R} | \exists x \in \mathbb{R} : (p_{c(g_i)}[\beta_i](x, y))\} < \min\{y \in \mathbb{R} | \exists x \in \mathbb{R} : (p_{c(g_{i+1})}[\beta_{i+1}](x, y))\}) \Rightarrow \lambda_{d_{YSequence}}(\{i \mapsto g_i[\beta_i] | 0 \leq i < n - 1\}).$$

Mitigating target visualization rules on planar map symbols

Subsequently we provide mitigations for the target visualization rules from section 4.2.7, which can be used to determine the level of fulfillment of instances of these rules in a computable way. For the **Insist** rule, no mitigation is provided here, as we regard this rule to be sufficiently easy in computation.

¹²Our Project Wiki (<http://wiki.softwarekartographie.de>) is accessible to interested users.

DistanceFixationEuclidean This rule expresses that the distance between two planar map symbol instances should be as close as possible to or greater than a given value $w_{\text{minimumDistance}}$. Thereby it can be shown that the following expressions regarding the target visualization rule values for two different sets of map symbol instances $\{g_1[\beta_1], g_2[\beta_2]\}$ and $\{g_3[\beta_3], g_4[\beta_4]\}$ hold:

$$\begin{aligned}
(1a) \quad & (d_s(\phi(p_{f(g_1)}[\beta_1]), \phi(p_{f(g_2)}[\beta_2])) < w_{\text{minimumDistance}}) \wedge \\
& (d_s(\phi(p_{f(g_3)}[\beta_3]), \phi(p_{f(g_4)}[\beta_4])) < w_{\text{minimumDistance}}) \wedge \\
& (d_s(\phi(p_{f(g_1)}[\beta_1]), \phi(p_{f(g_2)}[\beta_2])) > d_s(\phi(p_{c(g_3)}[\beta_3]), \phi(p_{c(g_4)}[\beta_4]))) \Rightarrow \\
& (\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{\text{minimumDistance}}) \leq \\
& \lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{\text{minimumDistance}})) \\
(1b) \quad & (d_s(\phi(p_{c(g_1)}[\beta_1]), \phi(p_{c(g_2)}[\beta_2])) \geq w_{\text{minimumDistance}}) \Rightarrow \\
& (\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{\text{minimumDistance}}) \leq \\
& \lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{\text{minimumDistance}})) \\
(2a) \quad & (d_s(\phi(p_{f(g_3)}[\beta_3]), \phi(p_{f(g_4)}[\beta_4])) < w_{\text{minimumDistance}}) \wedge \\
& (d_s(\phi(p_{f(g_1)}[\beta_1]), \phi(p_{f(g_2)}[\beta_2])) < w_{\text{minimumDistance}}) \wedge \\
& (d_s(\phi(p_{f(g_3)}[\beta_3]), \phi(p_{f(g_4)}[\beta_4])) > d_s(\phi(p_{c(g_1)}[\beta_1]), \phi(p_{c(g_2)}[\beta_2]))) \Rightarrow \\
& (\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{\text{minimumDistance}}) \leq \\
& \lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{\text{minimumDistance}})) \\
(2b) \quad & (d_s(\phi(p_{c(g_3)}[\beta_3]), \phi(p_{c(g_4)}[\beta_4])) \geq w_{\text{minimumDistance}}) \Rightarrow \\
& (\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{\text{minimumDistance}}) \leq \\
& \lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{\text{minimumDistance}}))
\end{aligned}$$

The evaluation result of $\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_1[\beta_1], g_2[\beta_2]\}, w_{\text{minimumDistance}})$ and the one of $\lambda_{d_{\text{DistanceFixationEuclidean}}}(\{g_3[\beta_3], g_4[\beta_4]\}, w_{\text{minimumDistance}})$ cannot be compared based on the mitigation, if neither (1a-b) nor (2a-b) are fulfilled. Thus, it is necessary to assign to them values from the set \mathbb{F} of the partial weak order $< \mathbb{F}, \leq$, for which the relation \leq allows no comparison.

The functions for **DistanceFixationHorizontal** and **DistanceFixationVertical** can be mitigated similarly, also potentially leading to uncomparable results.

AreaMinimization This rule expresses that the area of a planar map symbol instance should be as small as possible. Thereby it can be shown that the following expressions regarding the target visualization rule values for two different map symbol instances $f_1[\beta_1]$ and $f_2[\beta_2]$ hold:

$$\begin{aligned}
(1) : \quad & (a_s(\phi(p_{c(f_1)}[\beta_1])) \leq (a_s(\phi(p_{f(f_2)}[\beta_2]))) \Rightarrow \\
& (\lambda_{d_{\text{AreaMinimization}}}(f_1[\beta_1]) \leq \lambda_{d_{\text{AreaMinimization}}}(f_2[\beta_2])). \\
(2) : \quad & (a_s(\phi(p_{c(f_2)}[\beta_2])) \leq (a_s(\phi(p_{f(f_1)}[\beta_1]))) \Rightarrow \\
& (\lambda_{d_{\text{AreaMinimization}}}(f_2[\beta_2]) \leq \lambda_{d_{\text{AreaMinimization}}}(f_1[\beta_1])).
\end{aligned}$$

As above, if neither (1) nor (2) is fulfilled, $\lambda_{d_{\text{AreaMinimization}}}(f_1[\beta_1])$ and $\lambda_{d_{\text{AreaMinimization}}}(f_2[\beta_2])$ cannot be compared based on the mitigation.

4.3.2 Mitigations of linear map symbols

Having provided concepts for mitigating planar map symbol instances above, we subsequently do not introduce such concepts for linear map symbol instances. This is motivated by the lack of need for such concepts that we perceive regarding linear map symbols. Especially in the context of software maps, the shape is a way often used for distinguishing different concepts of the information model by assigning different planar map symbols. Therefore, it is important to enable layout mechanisms to use a wide range of different shapes. This is achieved via the concepts of closure and filling as introduced above.

The situation is different when looking at *lines*. Typical ways of distinguishing different line types are line style (e.g. dotted, solid), line color, or possibly an arrowhead. Using different forms of lines, e.g. Manhattan style lines or Bezier curves, which would be freely supported by a common layout mechanism via a concept similar to closure and filling applicable to linear map symbol instances, is not something we have found to be commonly used in software maps. Therefore, we do not introduce such a concept in the current version of the visualization model, mainly for reasons of simplicity.

Consequently, the concretization of the visualization model (for which an example is presented in section 4.4) has to select a mechanism for laying out lines, which then also acts as a restricting factor to the layout capabilities of the visualization model: only line laying out forms directly supported by this mechanisms may be used.

Of course, visualization rules concerning both linear and planar map symbol instance have to consider the mitigations of planar map symbols via closure and filling as introduced above. Such mitigations are presented below for the respective rules introduced in section 4.2.7.

Attachment: The attachment rule, which demands that a planar map symbol instance $f_{attachmentTarget}[\beta_1]$ has a linear map symbol instance $l_{attachedSymbol}[\beta_2]$ attached to it, can be mitigated relying on attachment to the filling of the planar map symbol instance:

$$(\exists(x_e, y_e) \in \mathbb{R}^2 : pf(f_{attachmentTarget}[\beta_1])(x_e, y_e) \wedge pl_{attachedSymbol}[\beta_2](x_e, y_e) \wedge \chi(\sigma(pl_{attachedSymbol}[\beta_2], (x_e, y_e)))) \Rightarrow \lambda_{d_{Attachment}}(f_{attachmentTarget}[\beta_1], l_{attachedSymbol}[\beta_2]).$$

PlanarLinearNonIntersection: This rule, which demands that a planar map symbol instance $f_{planar}[\beta_1]$ does not have any common point with a linear map symbol instance $l_{linear}[\beta_2]$, can be mitigated relying on **PlanarLinearNonIntersection** of the linear map symbol instance with the closure of the planar map symbol instance.

$$(\forall(x, y) \in \mathbb{R}^2 : \neg(p_c(f_{planar}[\beta_1])(x, y) \wedge pl_{linear}[\beta_2](x, y))) \Rightarrow \lambda_{d_{PlanarLinearNonIntersection}}(f_{planar}[\beta_1], l_{linear}[\beta_2]).$$

4.4 Concretization of the Visualization Model

Building on the concepts of *closure* and *filling* outlined above, an actual concretization has to introduce actual closures and fillings with additional properties that ease computation

when finding positions for map symbol instances that fulfill the visualization rule instances best. Subsequently one possible concretization of the visualization model is exemplarized. This concretization uses *rectangular* closures and fillings in order to employ an optimization algorithm to find possible layouts of the planar map symbols, while it makes use of a graph layout algorithm for finding line routings.

4.4.1 Concretizations for planar map symbols

To further concretize the concepts of closure and filling, actual realizations of those concepts are chosen here, the *rectangular closure* and the *rectangular filling*. The *rectangular closure* in this context is defined as the smallest rectangle containing all points of the map symbol instance to be enclosed. Similarly, the *rectangular filling* is defined as a largest rectangle, of which the points are all contained in the respective map symbol instance. The fulfillment of the requirements for closures and fillings by the rectangular versions can clearly be verified, however, a proof has been omitted here in respect to the report's extent.

Concretizations of map symbols

In accordance to the planar map symbols introduced in section 4.2.5, the following closure and filling functions are given below:

Ellipse: For the **Ellipse** map symbol having the predicate

$$p_{\text{Ellipse}}(X, Y, x, y, \text{semi}X, \text{semi}Y, \text{fillColor}, \text{borderColor}) := \left(\frac{(X-x)^2}{\text{semi}X^2} + \frac{(Y-y)^2}{\text{semi}Y^2} \right) \leq 1$$

$p_{c(\text{Ellipse})}$ gives the predicate of the closure and $p_{f(\text{Ellipse})}$ gives the predicate of the filling. They are specified as follows:

$$p_{c(\text{Ellipse})}(X, Y, x, y, \text{semi}X, \text{semi}Y, \text{fillColor}, \text{borderColor}) := (|X - x| \leq \text{semi}X \wedge |Y - y| \leq \text{semi}Y)$$

$$p_{f(\text{Ellipse})}(X, Y, x, y, \text{semi}X, \text{semi}Y, \text{fillColor}, \text{borderColor}) := \left(|X - x| \leq \frac{\text{semi}X}{\sqrt{2}} \wedge |Y - y| \leq \frac{\text{semi}Y}{\sqrt{2}} \right)$$

Rectangle: For the **Rectangle** map symbol augmented by the predicate

$$p_{\text{Rectangle}}(X, Y, x, y, \text{width}, \text{height}, \text{fillColor}, \text{borderColor}) := (|X - x| \leq \text{width}/2 \wedge |Y - y| \leq \text{height}/2)$$

rectangular closure and filling are the same and identical to the original rectangle itself.

Auxiliarly we rely on the following functions on closure predicates to access properties of these rectangles, which are highly descriptive:

$$\begin{aligned}
\min_x(p_{c(g)}[\beta]) &:= \min\{x \in \mathbb{R} \mid \exists y \in \mathbb{R} : p_{c(g)}[\beta](x, y)\} \\
\max_x(p_{c(g)}[\beta]) &:= \max\{x \in \mathbb{R} \mid \exists y \in \mathbb{R} : p_{c(g)}[\beta](x, y)\} \\
\min_y(p_{c(g)}[\beta]) &:= \min\{y \in \mathbb{R} \mid \exists x \in \mathbb{R} : p_{c(g)}[\beta](x, y)\} \\
\max_y(p_{c(g)}[\beta]) &:= \max\{y \in \mathbb{R} \mid \exists x \in \mathbb{R} : p_{c(g)}[\beta](x, y)\}
\end{aligned}$$

Analogously, such functions are defined on filling predicates.

As the values from above are closely related to the rectangular elements, they can easily be deduced from the closure and filling definitions for the respective map symbols. The value $(\max_x(p_{c(g)}[\beta]) - \min_x(p_{c(g)}[\beta]))$ can be interpreted as the *width* ($w_{c(g)}[\beta]$) of the corresponding rectangle (i.e. the rectangular closure), while the value $(\max_y(p_{c(g)}[\beta]) - \min_y(p_{c(g)}[\beta]))$ can be interpreted as the *height* ($h_{c(g)}[\beta]$) of the rectangle. We additionally introduce the *center* of the rectangle as a point $(x_{c(g)}[\beta], y_{c(g)}[\beta])$ with:

$$\begin{aligned}
x_{c(g)}[\beta] &= \frac{1}{2}(\max_x(p_{c(g)}[\beta]) - \min_x(p_{c(g)}[\beta])), \\
y_{c(g)}[\beta] &= \frac{1}{2}(\max_y(p_{c(g)}[\beta]) - \min_y(p_{c(g)}[\beta])).
\end{aligned}$$

These functions are utilized when actually transforming the concretized symbolic model elements to an optimization problem.

Concretizations of visualization rules

Subsequently, computable demands regarding the placement of map symbol instances are given, which can be proofed to lead to the respective mitigated visualization rule from section 4.3.1 being fulfilled. While the proofs are left out due to readability reasons, a computation strategy for finding actual solutions is given below.

Nesting This relationship, demanding that a planar map symbol instance $g_{intersecting}[\beta_1]$ to be nested into a planar map symbol instance $g_{intersected}[\beta_2]$, is surely fulfilled, if:

$$\begin{aligned}
\min_x(p_{c(g_{intersecting})}[\beta_1]) &\geq \min_x(p_{f(g_{intersected})}[\beta_2]) \wedge \\
\min_y(p_{c(g_{intersecting})}[\beta_1]) &\geq \min_y(p_{f(g_{intersected})}[\beta_2]) \wedge \\
\max_x(p_{c(g_{intersecting})}[\beta_1]) &\leq \max_x(p_{f(g_{intersected})}[\beta_2]) \wedge \\
\max_y(p_{c(g_{intersecting})}[\beta_1]) &\leq \max_y(p_{f(g_{intersected})}[\beta_2]).
\end{aligned}$$

Rewriting the inequalities above using the definitions of *width*, *height*, and *center* for closure and filling, they read as follows:

$$\begin{aligned}
x_{c(g_{intersecting})}[\beta_1] - \frac{1}{2}w_{c(g_{intersecting})}[\beta_1] &\geq x_{f(g_{intersected})}[\beta_2] - \frac{1}{2}w_{f(g_{intersected})}[\beta_2] \wedge \\
y_{c(g_{intersecting})}[\beta_1] - \frac{1}{2}h_{c(g_{intersecting})}[\beta_1] &\geq y_{f(g_{intersected})}[\beta_2] - \frac{1}{2}h_{f(g_{intersected})}[\beta_2] \wedge \\
x_{c(g_{intersecting})}[\beta_1] + \frac{1}{2}w_{c(g_{intersecting})}[\beta_1] &\leq x_{f(g_{intersected})}[\beta_2] + \frac{1}{2}w_{f(g_{intersected})}[\beta_2] \wedge \\
y_{c(g_{intersecting})}[\beta_1] + \frac{1}{2}h_{c(g_{intersecting})}[\beta_1] &\leq y_{f(g_{intersected})}[\beta_2] + \frac{1}{2}h_{f(g_{intersected})}[\beta_2].
\end{aligned}$$

Depending on the kind of map symbol of which $g_{intersecting}[\beta_1]$ and $g_{intersected}[\beta_2]$ are instances, the terms $x_{c(g_{intersecting})}[\beta_1]$, etc. can be replaced by the actual terms from the filling

and closure definitions for the respective map symbols. Thus, the expressions become terms including only the visual variables of the respective map symbol instances. The same is true for the rule concretizations below, which are subsequently outlined but not further detailed for reasons of brevity.

FullXSpecificIntersection This relationship, demanding of a planar map symbol instance $g_{intersecting}[\beta_1]$ to share all of the x-coordinates with a planar map symbol instance $g_{intersected}[\beta_2]$, is surely fulfilled, if:

$$\begin{aligned} \min_x(p_{c(g_{intersecting})}[\beta_1]) &\geq \min_x(p_{f(g_{intersected})}[\beta_2]) \wedge \\ \max_x(p_{c(g_{intersecting})}[\beta_1]) &\leq \max_x(p_{f(g_{intersected})}[\beta_2]). \end{aligned}$$

FullyYSpecificIntersection This relationship, demanding a planar map symbol instance $g_{intersecting}[\beta_1]$ to share its y-coordinates with a planar map symbol instance $g_{intersected}[\beta_2]$, is surely fulfilled, if:

$$\begin{aligned} \min_y(p_{c(g_{intersecting})}[\beta_1]) &\geq \min_y(p_{f(g_{intersected})}[\beta_2]) \wedge \\ \max_y(p_{c(g_{intersecting})}[\beta_1]) &\leq \max_y(p_{f(g_{intersected})}[\beta_2]). \end{aligned}$$

PlanarNonIntersection This relationship, demanding that planar map symbol instances $g_1[\beta_1]$ and $g_2[\beta_2]$ have no common point, is surely fulfilled, if:

$$\begin{aligned} \min_x(p_{c(g_1)}[\beta_1]) &\geq \max_x(p_{c(g_2)}[\beta_2]) \vee \\ \min_y(p_{c(g_1)}[\beta_1]) &\geq \max_y(p_{c(g_2)}[\beta_2]) \vee \\ \max_x(p_{c(g_1)}[\beta_1]) &\leq \min_x(p_{c(g_2)}[\beta_2]) \vee \\ \max_y(p_{c(g_1)}[\beta_1]) &\leq \min_y(p_{c(g_2)}[\beta_2]). \end{aligned}$$

XSequence This relationship, demanding planar map symbol instances $g_0[\beta_0], \dots, g_{n-1}[\beta_{n-1}]$ to be arranged from left to right, is surely fulfilled, if:

$$\bigwedge_{i=0}^{n-2} \left(\max_x(p_{c(g_i)}[\beta_i]) \leq \min_x(p_{c(g_{i+1})}[\beta_{i+1}]) \right).$$

YSequence This relationship demands planar map symbol instances $g_0[\beta_0], \dots, g_{n-1}[\beta_{n-1}]$ to be arranged from top to bottom. It is surely fulfilled, if:

$$\bigwedge_{i=0}^{n-2} \left(\max_y(p_{c(g_i)}[\beta_i]) \leq \min_y(p_{c(g_{i+1})}[\beta_{i+1}]) \right).$$

TargetVisualizationRules Concerning the target visualization rules the same term rewriting as outlined above can be performed. Concretizing the target visualization rules introduced in the software cartography visualization model relies on finding concrete distance and are functions. As these functions merely have to work with the rectangular closures and fillings

used in the mitigations of the target visualization rules, they can be restricted to rectangles here:

- $a_s(\phi(p_{c(g)}[\beta])) = w_{c(g)}[\beta] \cdot h_{c(g)}[\beta]$, the same is true for the filling,
- $d_s(\phi(p_{c(g1)}[\beta_1]), \phi(p_{c(g2)}[\beta_2])) = |x_{c(g1)}[\beta_1] - x_{c(g2)}[\beta_2]| - \frac{1}{2}(w_{c(g1)}[\beta_1] + w_{c(g2)}[\beta_2])$, regarding the *x-distance* on closure - the similar expressions hold for the filling and regarding the other distance functions.

Furthermore, the expressions the target visualization rules evaluate to can be combined to form a single target function by introducing arbitrary weightings. This admittedly introduces an impreciseness into the concretization, as it assumes an interval scale on \mathbb{F} , which is not provided in the definition of the target visualization rules and their mitigation. Moreover, it assumes that fulfilling one rule better can compensate deficit in the fulfillment of other rules. This is contrary to the efficient set approach described in section 4.2.6.

However, we see the introduction of arbitrary weightings together with summing up the results for the target visualization rule instances to a target function as a first approximation. We deliberately do not introduce more complex target functions until practical experience with the layouter based on these simplified functions proves them insufficient for performing layout of visualizations regarding their aesthetic appeal.

Optimization problem approach

Looking at the concretized rules detailed above, it becomes obvious that they form an optimization problem, especially when considering the mentioned *weighting scheme*. The expression augmenting the subclasses of `ConstraintVisualizationRule` lead, when the respective rule is instantiated, to the constraints of the optimization problem. The expressions augmenting the subclasses of `TargetVisualizationRule` are combined by introducing arbitrary weightings and summing them up. Thereby, the visual variables of the visualization object instances become variables of the optimization problem. The set of constraints of the optimization problem can be constructed by applying the concretized rules on the actual closures and fillings of map symbol instances (see example below). A similar procedure leads to the construction of the elements of the target function.

Depending on the chosen concretization of the visualization model, the complexity of the optimization problem may vary. This may both concern the number of equality and inequality terms involved, as well as the kind of such terms, which may span from simple linear terms to nonlinear terms of e.g. quadratic or higher polynomial order. Therefore, the concretization is strongly related to specific kinds of optimization problems, which have to be handled by the solver included in a tool for calculating layouts for symbolic models.

For the concretization chosen above, the optimization problem is potentially non-linearly constrained and does potentially not provide derivable penalty functions incorporating the constraints. Furthermore, the target function is potentially non-linear and non derivable. Additionally, the problem is likely to be high dimensional even for *small* visualizations. Below, we show an excerpt of the concretized rules for the exemplary visualization in Figure 2.1, the corresponding symbolic model excerpt can be found in Figure 2.4. Therein we use number

indexes instead of names for the rectangles representing business applications in order to promote readability. A concretization of all rules leading to Figure 2.1 can be found in appendix C. Note that deliberately no target function is given here.

$$\begin{aligned}
x_{Munich} - \frac{1}{2}w_{Munich} &\leq x_{100} - \frac{1}{2}w_{100}, & y_{Munich} - \frac{1}{2}h_{Munich} &\leq y_{100} - \frac{1}{2}h_{100}, \\
x_{Munich} + \frac{1}{2}w_{Munich} &\geq x_{100} + \frac{1}{2}w_{100}, & y_{Munich} + \frac{1}{2}h_{Munich} &\geq y_{100} + \frac{1}{2}h_{100}, \\
x_{Munich} - \frac{1}{2}w_{Munich} &\leq x_{500} - \frac{1}{2}w_{500}, & y_{Munich} - \frac{1}{2}h_{Munich} &\leq y_{500} - \frac{1}{2}h_{500}, \\
x_{Munich} + \frac{1}{2}w_{Munich} &\geq x_{500} + \frac{1}{2}w_{500}, & y_{Munich} + \frac{1}{2}h_{Munich} &\geq y_{500} + \frac{1}{2}h_{500}, \\
x_{Munich} - \frac{1}{2}w_{Munich} &\leq x_{600} - \frac{1}{2}w_{600}, & y_{Munich} - \frac{1}{2}h_{Munich} &\leq y_{600} - \frac{1}{2}h_{600}, \\
x_{Munich} + \frac{1}{2}w_{Munich} &\geq x_{600} + \frac{1}{2}w_{600}, & y_{Munich} + \frac{1}{2}h_{Munich} &\geq y_{600} + \frac{1}{2}h_{600}, \\
x_{Munich} - \frac{1}{2}w_{Munich} &\leq x_{700} - \frac{1}{2}w_{700}, & y_{Munich} - \frac{1}{2}h_{Munich} &\leq y_{700} - \frac{1}{2}h_{700}, \\
x_{Munich} + \frac{1}{2}w_{Munich} &\geq x_{700} + \frac{1}{2}w_{700}, & y_{Munich} + \frac{1}{2}h_{Munich} &\geq y_{700} + \frac{1}{2}h_{700},
\end{aligned}$$

$$\begin{aligned}
x_{Hamburg} - \frac{1}{2}w_{Hamburg} &\leq x_{400} - \frac{1}{2}w_{400}, & y_{Hamburg} - \frac{1}{2}h_{Hamburg} &\leq y_{400} - \frac{1}{2}h_{400}, \\
x_{Hamburg} + \frac{1}{2}w_{Hamburg} &\geq x_{400} + \frac{1}{2}w_{400}, & y_{Hamburg} + \frac{1}{2}h_{Hamburg} &\geq y_{400} + \frac{1}{2}h_{400}, \\
x_{Hamburg} - \frac{1}{2}w_{Hamburg} &\leq x_{900} - \frac{1}{2}w_{900}, & y_{Hamburg} - \frac{1}{2}h_{Hamburg} &\leq y_{900} - \frac{1}{2}h_{900}, \\
x_{Hamburg} + \frac{1}{2}w_{Hamburg} &\geq x_{900} + \frac{1}{2}w_{900}, & y_{Hamburg} + \frac{1}{2}h_{Hamburg} &\geq y_{900} + \frac{1}{2}h_{900}.
\end{aligned}$$

A possible solution to this problem gives the positions for the map symbol instances leading to the layouted diagram. We are currently experimenting with *genetic algorithms* [BL04] for solving such an optimization problem and thus finding possible layouts for the planar map symbol instances.

4.4.2 Concretizations for linear map symbols

Basically, we consider two different strategies for performing the layouting of linear map symbol instances, which are with their advantages and disadvantages described below. Based on this, we select the strategy more advantageous in respect to runtime efficiency, which is then detailed.

Possible solution strategies

Linear map symbol instances are characterized by special points influencing their appearance in a visualization, which are in the simplest case so-called *bendpoints*. It has to be noted that the number of such points is not necessarily externally specified by means of the visualization model as outlined above. Therefore, the computation of an actual optimal solution has to incorporate this fact, preferably treating the numbers of bendpoints as variables for which optimal values have to be determined.

Thus, an exact solution of the problem basically involves solving two optimization problems:

- Finding an optimal number of bending points for each linear map symbol instance, which is a combinatorial optimization problem.
- Finding an optimal layout of a software map given a specific number of bendpoints for

each linear map symbol instance, which involves as outlined in section 4.4.1 a possibly nonlinearly constrained optimization problem with a possibly nonlinear target function.

This leads to a combinatorial explosion regarding the solution space of the optimization problem in case linear map symbol instances are involved. For every possible number of bendpoints of a linear map symbol instance a software map can be layouted, which is itself, as stated above, a nonlinear optimization problem. Thus the introduction of linear map symbol instances with a varying number of bending points increases the size of the solution space by a factor $|\{2 \dots N_{max}\}|^m$, where m is the number of linear map symbols, and N_{max} the maximum number of bending points allowed for each line. Thus, finding a possible solution with adequate runtime efficiency most probably relies on a heuristic, where basically a one-step or a two-step approach is possible.

One-Step Approach: Integrated optimization problem The one-step approach constructs an optimization problem, of which the solution constitutes a clear-cut description of a correct layout, in respect to both linear and planar map symbol instances. Of course, the solution approach has to be able to address the enormous solution space alluded to above. Additionally, the solution has to consider that the number of variables for which optimal values have to be found is not ex ante fixed: As coordinates have to be found for every bending point, but also the number of bendpoints is to be optimized for each linear map symbol instance, the number of variables of the problem is open. This has to be considered by a heuristic used to support the one-step approach.

Due to such requirements, we do currently not further pursue this approach, while a first solution is pursued via the two-step approach, where we expect a tendency toward more runtime-efficient solution strategies.

Two-Step approach: Separating linear and planar layouting The two-step approach separates finding a solution of the optimization problem into two subsequent steps, where results from the first step supply the starting point of the second step:

- In the first step a possible layout for the planar map symbol instances is found. Linear map symbol instances are not taken into consideration, at least not directly.
- In the second step, the linear map symbol instances are layouted, where the positionings of the planar map symbol instances derived from the first step are considered fixed. This can of course only accidentally produce a globally optimal layout,

Of course, indirect consideration of line layouting in the first step is basically possible, map symbol instances between which lines are to be drawn in the second step can e.g. be placed in proximity to each other.

We deem the simplification into two steps feasible, as, due to the complexity of the optimization problem, no globally optimal solutions are sought, but heuristics are employed. These can be expected to yield only local optima, anyway.

The concretization of the visualization model described here relies on a graph layout algorithm for the second step, while the layouting of the planar map symbol instances is achieved via solving an optimization problem as described in section 4.4.1.

Graph layouting

In order to layout instances of solely linear map symbols with positions for the planar map symbol instances already determined, we translate the problem of layouting these map symbol instances according to the visualization rule instances which reference such map symbol instances into a graph layout problem.

This is possible due to the fact that graph layout algorithms basically consider similar concepts as our visualization model. This can be demonstrated via the concepts employed in the general layout model for graph layouting introduced in [Br99], which are:

Layout elements, which are meant to be positioned by a solution of the graph layout problem. These elements resemble the *map symbol instances* from our visualization model.

Constraints, which restrict the possible layouts and thus limit the layout problem's solution space. These elements correspond to the *constraint visualization rule instances*.

Objective functions, which are used to assess e.g. ergonomic aspects of a layout. These elements resemble the *target visualization rule instances* from our model.

A *translation* of the corresponding concepts, as outlined above, is rather intuitive, therefore we do not further detail it here, as it would only show the conversion between two generic formalisms. Moreover, in most graph layout algorithms, the constraints are restricted to be of a certain type and cannot be set freely, as it is intended with the visualization model. This emphasizes the fact that our visualization model takes an approach differing from those of many graph layout algorithms. The approaches can be classified according to [CT06]:

Algorithmic approach This approach generates the layout of a graph according to a pre-defined set of aesthetic criteria and constraints, which are embodied in the algorithm itself. Therefore, constraints or objective functions not incorporated in the algorithm cannot be supported, nor can the application of these to certain elements be prevented.

Declarative approach This approach describes the layout of a graph by any set of aesthetic criteria and constraints, while it does not provide an actual algorithm for calculating a solution to the constraint system.

As the visualization model has to be regarded as pursuing the *declarative approach*, whereas most of the graph layout algorithms are of an *algorithmic approach*, a graph layout has to be performed choosing an adequate algorithm that supports the constraints imposed by and embodies the aesthetic criteria specified by the respective symbolic model. The rules are then applied by feeding the respective elements to the algorithm. Such a *limited rule set* algorithm might e.g. be the spring algorithm (see [DB99]), which interprets the vertices as electrically charged particles repelling each other, and the edges as springs pulling the connected particles

together. In such a setting the optimal layout is calculated by finding a layout where the forces on each particle sum up to zero.

However in order to be applicable for more than special cases of symbolic models corresponding to the visualization model, a minimum set of requirements on the specific graph layout algorithm can be derived from the concepts of the visualization model:

- As linear and planar map symbol instances are involved, the graph layout algorithm must support lines (most probably as the representation of edges) and areas (most probably as the representation of vertices). A critical point is that the size of the areas might have to be flexible for them being able to serve as fillings or closures.
- As the layouting is carried out in two steps, it has to be expected that the positions are already fixed for the instances of the planar map symbols involved. Therefore, the graph layout algorithm has to support layout elements for which the positioning is ex ante given and not allowed to change.
- In order to be able to support instances of the attachment rule, the graph layout algorithm has to be able to force that a linear map symbol is connected to a planar map symbol. It might e.g. achieve this by representing the areas to be connected by a line as vertices, creating an edge between them and calculating a graph layout.
- In order to support the **PlanarLinearNonIntersection**, the graph layout algorithm has to prevent lines from being drawn through certain areas.
- Supporting **LineNonIntersection** demands of a graph layout algorithm to be able to explicitly forbid certain lines from crossing, i.e. from sharing one or more points that are not endpoints. On the one hand, this might not always be possible in specific layouts, e.g. if non-planar subgraphs are involved. On the other hand, many graph layout algorithm try to optimize in respect to this demand, considering all lines they layout, without focusing on lines specifically targeted e.g. by a constraint. Thus, it might be feasible not to consider the **LineNonIntersection** when choosing a graph layout algorithm here.

These demands e.g. rule out the spring algorithm alluded to above.

As the version of a visualization model exemplified in this article does not comprise target visualization rules involving instances of linear map symbols, it becomes not directly obvious, that the situation is similar regarding the objective function and target visualization rules. Again, here the algorithm chosen would have to support the aesthetic aspects by directly embodying them. Such aspects could be:

- The distance between adjacent vertices u and v are supposed to be approximately equal,
- The vertices are demanded not to be too close together.

Again it has to be emphasized that choosing an algorithm determines the respective aesthetic criterion and also the line form, e.g. *Manhattan style lines* or *Bezier curves*.

CHAPTER 5

Related Work

In the context of Model Visualization one may encounter particularly different approaches that have been taken in addressing requirements similar to the ones described in sections 2 and 3. A very popular approach is strongly focused on the visualization flexibility itself, incorporated in tools like *Microsoft Visio* or the *eclipse Graphical Editor Framework* [Mo04]. Both systems grant the user a maximum of flexibility for adapting the visualizations, but mostly, if not totally, concentrate on manual layouting. Another approach is taken in the field of graph visualization, which is often incorporated in the tools mentioned before. Here basic layout algorithms, like Sugiyama- or Spring-layout (see [JM03], [DB99]) are known.

In the field of graph visualization, a basic differentiation of approaches to layouting problems is used in [CT06]. Therein *algorithmic approaches* widely taken in graph visualizations due to their advantages regarding computational efficiency are opposed by the *declarative approaches*, which have been researched less intensively due to their inherent complexity and computational inefficiency. Nevertheless these approaches for describing the layout of a visualization by constraints are presented in [CT06] to be advantageous in respect to their expressiveness. Such advantages are supposed to be provided by the visualization model.

Another work strongly focusing on graph visualizations but also employing a formal notation for the elements can be found in [DV02], where the approach of model transformation is further leveraged. A quite similar approach utilizing the concepts of model transformation is explained more deeply in [St04], although the focus of this work is narrowly concentrated on the transformation of objects not employing their links through *instances* of associations.

The approach taken in [To04] more strongly incorporates the meaning of absolute and relative positioning, as concepts like rectangles extended to span different fields of a matrix are introduced on the base of a formalism utilizing *signatures*. While a general applicability on the information model of the application landscape is also shown there no simple to use notation for a *visualization model* is provided.

The calculation of absolute positions of elements out of certain requirements regarding their relative positioning is used by [BB98] for rule based definition of user interfaces. Therefore an approach for translating the relative positioning requirements to linear inequality systems, which are then solved by an algorithm is presented in [BB98]. The algorithm presented in this work is called *Cassowary*, who despite of his notable capabilities in constraint solving, stays closely tied to a low level of abstraction, providing a basic system for constraining a set of variables, but not linking them to more abstract concepts like *map symbols* or *visualization rules* to be used in creating software maps.

The visualization model introduced in this paper makes up the formal basis of our concept for automated visualization generation. The applicability of this approach, as well as the general feasibility to the field of *software cartography* have been proofed by a prototypic tool implementation. Nevertheless several issues are still to be explored. Subsequently some of them are outlined.

1. **Concretizations of the visualization model:** The concretization of the visualization model currently in use, translates the symbolic models to *high-dimensional*, *non-linear* (both in constraints and target function), and *non derivable* (target function) optimization problems. The computation of the (near) optimal solution is at the moment performed by a genetic algorithm [BL04], but is considerably lacking performance. Here we see some potential for improvement, e.g. by utilizing hybrid optimization approaches [Ch05].

Nevertheless, computational efficiency could possibly be rised by making adaptations to the concretization strategy. Promising adaptations center around the following strategies:

Develop pattern specific concretizations: The visualization model outlined above has been designed with maximum genericity and expressiveness in mind. Nevertheless, many of the visualizations, especially in the field of software cartography, do not employ the full scale genericity of the model, but are fairly stereotypic. The approach of *pattern specific concretizations* takes advantage from this fact by tailoring algorithms to specifically suite the requirements of a specific pattern, e.g. the *nesting-pattern* behind a *cluster map*.

Reduce number of dimensions: The optimization problem resulting from the concretization outlined in section 4.4 is likely to be high dimensional even for software

maps that are considered small in terms of practical relevance. The number of dimensions could nevertheless be reduced by *splitting* the problem. This can be performed by relying on visualization rules, via which two disjoint optimization problems can be identified.

Linearize the problem: The optimization problem produced by applying the above concretization contains nonlinear constraints. Nevertheless, there might be cases where the constraints could be linearized utilizing a certain heuristic. The development of such a heuristic could greatly reduce the problem's complexity and allow the use of efficient optimizers.

Find derivable smoothing: The optimization problem resulting from the concretization currently in use contains constraints leading to non derivable penalty terms used in the genetic algorithm. Nevertheless, some of these terms might have derivable approximations that could be used for reducing the problem's complexity and allow the use of more efficient optimizers. Here the `PlanarNonIntersection` constraints might especially be of interest.

These approaches for adapting the concretization strategy are all connected to the disadvantage of yielding solutions less optimal.

2. **Enhancing the approach:** An interesting point arising from usability considerations is the *composition of map symbols*. A mechanism for performing map symbol composition should at best be simply focusing on the object-oriented tier of the visualization model (see section 4.1), where basic object-oriented design patterns [Ga94], like e.g. the *composite pattern* could greatly leverage the offered usability. Hence, mathematical aspects of map symbol composition should be preferably addressed in a way sufficiently easy enough to support the creation of a flexible editor for defining new map symbols based on graphic primitives in a tool.

Another possible enhancement is focused on user interaction. As it can be seen in Figure 2.7 the arrows symbolizing the transformation from the semantic to the symbolic model point in two directions. While at the moment the problems arising from generating visualizations from information by transforming from the *semantic* to the *symbolic* model, are understood well, the issues connected to the inverse transformation have not been researched yet. Here we see an important focus for future work in order to support *interactive* software maps by supporting inverse transformation.

3. **Transformation and declarative viewpoint definition:** Concerning the transformation from the *semantic* to the *symbolic* model, a major decision has still to be taken. Whereas different languages for defining transformation rules declaratively or at least in a hybrid style were analyzed [Bu05], no language yet exists, which seemed applicable in the context of the model transformations for defining viewpoints. In our prototypic implementation the transformation from the *semantic* to the *symbolic* model is implemented in Java, thus burdening a person designing viewpoints with the complexity of a programming language, rather than enabling him to focus on viewpoint declaration. In future, this will be replaced by a language for describing transformation rules, preferably this language should support the definition of transformation rules utilizing a mainly declarative (e.g. SQL-like) syntax. Languages potentially suitable for express-

ing the rules needed in order to perform automatic generation of visualizations could be languages like ATL [AT06] and BOTL [BM03].

4. **Other applications of the visualization model:** The visualization model provided in chapter 4 is not limited to software map generation. In accordance to approaches taken by Badros and Borning (see [BB98]) such a model could be applied to automated layout of user interfaces, too. This might especially be interesting, as the object-oriented visualization model provided in this report grants the potential user a more high-level view on the supported concepts, not forcing the usage of inequality or equality constraints but providing an object oriented understanding of concepts like e.g. *sequencing* symbols from left to right.

Planar map symbol area homomorphism

For defining the *area*-function for planar map symbol instances, we rely on the existence of two homomorphic structures Φ and Σ defined in accordance to section 4.2.4 as follows:

- the structure $\Phi = \langle \{p|p : \mathbb{R}^2 \rightarrow \mathbb{B}\}, \vee, \neg \rangle$ with \vee and \neg being the usual boolean operators, and
- the structure $\Sigma = \langle \{\mathbb{X}|\mathbb{X} \subseteq \mathbb{R}^2\}, \cup, \cap \rangle$ with \cup and \cap being the usual set operators.

Defining the function $\phi(p[\beta]) = \{(x, y) \in \mathbb{R}^2 | p(x, y)\}$ we subsequently show that this function is a homomorphism $\phi : \Phi \rightarrow \Sigma$.

Proposition: $\phi(p_1[\beta_1]) \cup \phi(p_2[\beta_2]) = \phi(p_1[\beta_1] \vee p_2[\beta_2])$. For every point $(x, y) \in \mathbb{R}^2$ exactly one of the following terms holds:

- $(x, y) \in \phi(p_1[\beta_1])$ and $(x, y) \in \phi(p_2[\beta_2])$, thereby it follows $(x, y) \in \phi(p_1[\beta_1]) \cup \phi(p_2[\beta_2])$, as well as from the first two expressions it follows $p_1[\beta_1](x, y) = \text{true}$ and $p_2[\beta_2](x, y) = \text{true}$, therefore $p_1[\beta_1](x, y) \vee p_2[\beta_2](x, y) = \text{true}$, which implies $(x, y) \in \phi(p_1[\beta_1] \vee p_2[\beta_2])$.
- $(x, y) \in \phi(p_1[\beta_1])$ and $(x, y) \notin \phi(p_2[\beta_2])$, thereby it follows $(x, y) \in \phi(p_1[\beta_1]) \cup \phi(p_2[\beta_2])$, as well as from the first two expressions it follows $p_1[\beta_1](x, y) = \text{true}$ and $p_2[\beta_2](x, y) = \text{false}$, therefore $p_1[\beta_1](x, y) \vee p_2[\beta_2](x, y) = \text{true}$, which implies $(x, y) \in \phi(p_1[\beta_1] \vee p_2[\beta_2])$.
- $(x, y) \notin \phi(p_1[\beta_1])$ and $(x, y) \in \phi(p_2[\beta_2])$, thereby it follows $(x, y) \in \phi(p_1[\beta_1]) \cup \phi(p_2[\beta_2])$, as well as from the first two expressions it follows $p_1[\beta_1](x, y) = \text{false}$ and $p_2[\beta_2](x, y) = \text{true}$, therefore $p_1[\beta_1](x, y) \vee p_2[\beta_2](x, y) = \text{true}$, which implies $(x, y) \in \phi(p_1[\beta_1] \vee p_2[\beta_2])$.
- $(x, y) \notin \phi(p_1[\beta_1])$ and $(x, y) \notin \phi(p_2[\beta_2])$, thereby it follows $(x, y) \notin \phi(p_1[\beta_1]) \cup \phi(p_2[\beta_2])$, as well as from the first two expressions it follows $p_1[\beta_1](x, y) = \text{false}$ and $p_2[\beta_2](x, y) = \text{false}$, therefore $p_1[\beta_1](x, y) \vee p_2[\beta_2](x, y) = \text{false}$, which implies $(x, y) \notin \phi(p_1[\beta_1] \vee p_2[\beta_2])$.

Proposition: $\overline{\phi(p[\beta])} = \phi(\neg p[\beta])$. For every point $(x, y) \in \mathbb{R}^2$ exactly one of the following terms holds:

- $(x, y) \in \phi(p[\beta])$, thereby it follows $(x, y) \notin \overline{\phi(p[\beta])}$, as well as from the first expression it follows $p[\beta](x, y) = \text{true}$, thereby $\neg p[\beta](x, y) = \text{false}$, which implies $(x, y) \notin \phi(\neg p[\beta])$.
- $(x, y) \notin \phi(p[\beta])$, thereby it follows $(x, y) \in \overline{\phi(p[\beta])}$, as well as from the first expression it follows $p[\beta](x, y) = \text{false}$, thereby $\neg p[\beta](x, y) = \text{true}$, which implies $(x, y) \in \phi(\neg p[\beta])$.

Having proven the propositions above, also the homomorphism is shown to exist.

Multiplicity values formalized

In section 4.2, functions $\mathbb{W}^V(v)$ and $\mathbb{W}^A(a)$ are used to supply the domains of visualization variables and associations to visualization objects (i.e. to the class `VisualizationObject` and its subclasses). Thereby, these functions are only informally specified, due to readability reasons. Subsequently, a formalization of these specifications is delivered in addition.

Preliminary definitions: The definitions of attributes and associations in MOF [OM06a] demand that each attribute or association holds the following properties:

- **isOrdered: boolean** thereby making up a set of two alternatives regarding the ordering of an attribute or association end $\{ordered, \overline{ordered}\}$,
- **isUnique: boolean** thereby making up a set of two alternatives regarding the uniqueness of an attribute or association end $\{unique, \overline{unique}\}$, and
- **lowerBound: integer** and **upperBound: UnlimitedNatural** thereby restricting the multiplicity of an attribute or an association end to an tuple element from the set $\{(l, u) \in \mathbb{N}_0 \times \mathbb{N}_0^\infty \mid l \leq u\}$.

We further define a function μ , mapping every attribute or association end to its corresponding properties:

$$\mu : \bigcup_{o \in \mathbb{O}} \mathbb{V}^A(o) \cup \bigcup_{o \in \mathbb{O}} \mathbb{V}^V(o) \rightarrow \{ordered, \overline{ordered}\} \times \{unique, \overline{unique}\} \times \{(l, u) \in \mathbb{N}_0 \times \mathbb{N}_0^\infty \mid l \leq u\}.$$

μ is then defined as: $\mu(o) = (\underline{o}, \underline{u}, (i, j))$ with

- $\underline{o} = ordered$, if and only if the respective attribute or association end has **isOrdered** = **true** and $\underline{o} = \overline{ordered}$ otherwise.

- $\underline{u} = \text{unique}$, if and only if the respective attribute or association end has `isUnique = true` and $\underline{u} = \text{unique}$ otherwise.
- (i, j) is the `lowerBound` and `upperBound` regarding the multiplicity of the respective attribute or association end.

Additionally, type functions for supplying the type of an attribute \mathbb{T}^V or and association end \mathbb{T}^A are introduced:

- $\mathbb{T}^A(a) \subseteq \mathcal{P}(\mathbb{O})$ for an association end a contains all non-abstract classes, which are directly or indirectly subclasses of the class to which it points, and also that class itself, if it is a non-abstract class.
- $\mathbb{T}^V(v)$ supplies the type of the variable v . Specifically, the set of all elements which are of this type is supplied. For this function, no further formalization is provided here.

Moreover, two general mathematical constructs essential to the subsequent definitions are introduced here:

- $\mathcal{M}(\mathbb{X})$ is the set of multi-sets over a given set \mathbb{X} , which means that elements from \mathbb{X} are alluded to be contained more than once in a multiset $\underline{\mathbb{M}}$ in $\mathcal{M}(\mathbb{X})$. For a given multiset $\underline{\mathbb{M}} \in \mathcal{M}(\mathbb{X})$ an operation providing its cardinality $|\underline{\mathbb{M}}|$ meaning the number of (even duplicate) elements exists. Thereby the set of multi-sets can be restricted to only contain multi-sets of given cardinality, which is denoted as $\mathcal{M}_i(\mathbb{X}) = \{\underline{\mathbb{M}} \in \mathcal{M}(\mathbb{X}) \mid |\underline{\mathbb{M}}| = i\}$. Furthermore, the empty multi-set $\underline{\emptyset}$ is regarded to be included in $\mathcal{M}(\mathbb{X})$ for any set \mathbb{X} , where $|\underline{\emptyset}| = 0$ holds.
- $\mathcal{S}_n(\mathbb{X})$ is the set of sequences of length n over a given set \mathbb{X} with $n \in \mathbb{N}$. A sequence over a set \mathbb{X} is defined as a mapping from a natural number i to an element from the set \mathbb{X} with $0 \leq i < n$. In case $n = 0$ the empty sequence is the only element of \mathcal{S}_0 . We further distinguish, whether the mapping is *injective* (meaning that every element from \mathbb{X} can be mapped to only once) or not. The injective case is subsequently denoted by \mathcal{S}_n^I .

Definitions of \mathbb{W}^V and \mathbb{W}^A : Backed by above definitions the sets informally introduced in section 4.2 can be formalized as follows¹:

$$\mathbb{W}^V(v) = \begin{cases} \mathbb{T}^V(v) & \text{for } \mu(v) = (\underline{o}, \underline{u}, (1, 1)) \\ \mathbb{T}^V(v) \cup \{\perp\} & \text{for } \mu(v) = (\underline{o}, \underline{u}, (0, 1)) \\ \bigcup_{n=i}^j \mathcal{M}_n(\mathbb{T}^V(v)) & \text{for } \mu(v) = (\overline{\text{ordered}}, \overline{\text{unique}}, (i, j)) \wedge j > 1 \\ \{\mathbb{X} \in \mathcal{P}(\mathbb{T}^V(v)) \mid i \leq |\mathbb{X}| \leq j\} & \text{for } \mu(v) = (\overline{\text{ordered}}, \overline{\text{unique}}, (i, j)) \wedge j > 1 \\ \bigcup_{n=i}^j \mathcal{S}_n^I(\mathbb{T}^V(v)) & \text{for } \mu(v) = (\text{ordered}, \text{unique}, (i, j)) \wedge j > 1 \\ \bigcup_{n=i}^j \mathcal{S}_n(\mathbb{T}^V(v)) & \text{for } \mu(v) = (\text{ordered}, \overline{\text{unique}}, (i, j)) \wedge j > 1 \end{cases}$$

¹In these formalizations $\underline{o} \in \{\text{ordered}, \overline{\text{ordered}}\}$ and $\underline{u} \in \{\text{unique}, \overline{\text{unique}}\}$ hold.

$$\mathbb{W}^A(a) = \begin{cases} \mathbb{I}(\mathbb{T}^A(a)) & \text{for } \mu(a) = (\underline{o}, \underline{u}, (1, 1)) \\ \mathbb{I}(\mathbb{T}^A(a)) \cup \{\perp\} & \text{for } \mu(a) = (\underline{o}, \underline{u}, (0, 1)) \\ \bigcup_{n=i}^j \mathcal{M}_n(\mathbb{I}(\mathbb{T}^A(a))) & \text{for } \mu(a) = (\overline{ordered}, \overline{unique}, (i, j)) \wedge j > 1 \\ \{\mathbb{X} \in \mathcal{P}(\mathbb{I}(\mathbb{T}^A(a))) \mid i \leq |\mathbb{X}| \leq j\} & \text{for } \mu(a) = (\overline{ordered}, \overline{unique}, (i, j)) \wedge j > 1 \\ \bigcup_{n=i}^j \mathcal{S}_n^I(\mathbb{I}(\mathbb{T}^A(a))) & \text{for } \mu(a) = (ordered, unique, (i, j)) \wedge j > 1 \\ \bigcup_{n=i}^j \mathcal{S}_n(\mathbb{I}(\mathbb{T}^A(a))) & \text{for } \mu(a) = (ordered, \overline{unique}, (i, j)) \wedge j > 1 \end{cases}$$

From symbolic model to optimization problem

Here we provide the concretization of a symbolic model to an optimization problem. Therefore we use the cluster map visualization from Figure 2.1 as an example. The following optimization problem can be derived from the rules describing the exemplary visualization:

- (1) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{100} - \frac{1}{2}w_{100}$, (2) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{100} - \frac{1}{2}h_{100}$,
- (3) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{100} + \frac{1}{2}w_{100}$, (4) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{100} + \frac{1}{2}h_{100}$,
- (5) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{500} - \frac{1}{2}w_{500}$, (6) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{500} - \frac{1}{2}h_{500}$,
- (7) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{500} + \frac{1}{2}w_{500}$, (8) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{500} + \frac{1}{2}h_{500}$,
- (9) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{600} - \frac{1}{2}w_{600}$, (10) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{600} - \frac{1}{2}h_{600}$,
- (11) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{600} + \frac{1}{2}w_{600}$, (12) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{600} + \frac{1}{2}h_{600}$,
- (13) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{700} - \frac{1}{2}w_{700}$, (14) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{700} - \frac{1}{2}h_{700}$,
- (15) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{700} + \frac{1}{2}w_{700}$, (16) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{700} + \frac{1}{2}h_{700}$,
- (17) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1000} - \frac{1}{2}w_{1000}$, (18) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1000} - \frac{1}{2}h_{1000}$,
- (19) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1000} + \frac{1}{2}w_{1000}$, (20) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1000} + \frac{1}{2}h_{1000}$,
- (21) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1300} - \frac{1}{2}w_{1300}$, (22) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1300} - \frac{1}{2}h_{1300}$,
- (23) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1300} + \frac{1}{2}w_{1300}$, (24) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1300} + \frac{1}{2}h_{1300}$,
- (25) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1400} - \frac{1}{2}w_{1400}$, (26) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1400} - \frac{1}{2}h_{1400}$,
- (27) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1400} + \frac{1}{2}w_{1400}$, (28) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1400} + \frac{1}{2}h_{1400}$,
- (29) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1600} - \frac{1}{2}w_{1600}$, (30) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1600} - \frac{1}{2}h_{1600}$,
- (31) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1600} + \frac{1}{2}w_{1600}$, (32) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1600} + \frac{1}{2}h_{1600}$,
- (33) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1700} - \frac{1}{2}w_{1700}$, (34) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1700} - \frac{1}{2}h_{1700}$,
- (35) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1700} + \frac{1}{2}w_{1700}$, (36) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1700} + \frac{1}{2}h_{1700}$,
- (37) $x_{Munich} - \frac{1}{2}w_{Munich} \leq x_{1800} - \frac{1}{2}w_{1800}$, (38) $y_{Munich} - \frac{1}{2}h_{Munich} \leq y_{1800} - \frac{1}{2}h_{1800}$,
- (39) $x_{Munich} + \frac{1}{2}w_{Munich} \geq x_{1800} + \frac{1}{2}w_{1800}$, (40) $y_{Munich} + \frac{1}{2}h_{Munich} \geq y_{1800} + \frac{1}{2}h_{1800}$,
- (41) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{400} - \frac{1}{2}w_{400}$, (42) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{400} - \frac{1}{2}h_{400}$,
- (43) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{400} + \frac{1}{2}w_{400}$, (44) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{400} + \frac{1}{2}h_{400}$,
- (45) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{900} - \frac{1}{2}w_{900}$, (46) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{900} - \frac{1}{2}h_{900}$,
- (47) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{900} + \frac{1}{2}w_{900}$, (48) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{900} + \frac{1}{2}h_{900}$,
- (49) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{1100} - \frac{1}{2}w_{1100}$, (50) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{1100} - \frac{1}{2}h_{1100}$,
- (51) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{1100} + \frac{1}{2}w_{1100}$, (52) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{1100} + \frac{1}{2}h_{1100}$,

- (53) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{1500} - \frac{1}{2}w_{1500}$, (54) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{1500} - \frac{1}{2}h_{1500}$,
 (55) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{1500} + \frac{1}{2}w_{1500}$, (56) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{1500} + \frac{1}{2}h_{1500}$.
 (57) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{1620} - \frac{1}{2}w_{1620}$, (58) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{1620} - \frac{1}{2}h_{1620}$,
 (59) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{1620} + \frac{1}{2}w_{1620}$, (60) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{1620} + \frac{1}{2}h_{1620}$,
 (61) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{1720} - \frac{1}{2}w_{1720}$, (62) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{1720} - \frac{1}{2}h_{1720}$,
 (63) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{1720} + \frac{1}{2}w_{1720}$, (64) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{1720} + \frac{1}{2}h_{1720}$.
 (65) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{1820} - \frac{1}{2}w_{1820}$, (66) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{1820} - \frac{1}{2}h_{1820}$,
 (67) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{1820} + \frac{1}{2}w_{1820}$, (68) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{1820} + \frac{1}{2}h_{1820}$,
 (69) $x_{Hamburg} - \frac{1}{2}w_{Hamburg} \leq x_{2100} - \frac{1}{2}w_{2100}$, (70) $y_{Hamburg} - \frac{1}{2}h_{Hamburg} \leq y_{2100} - \frac{1}{2}h_{2100}$,
 (71) $x_{Hamburg} + \frac{1}{2}w_{Hamburg} \geq x_{2100} + \frac{1}{2}w_{2100}$, (72) $y_{Hamburg} + \frac{1}{2}h_{Hamburg} \geq y_{2100} + \frac{1}{2}h_{2100}$.

 (73) $x_{Garching} - \frac{1}{2}w_{Garching} \leq x_{200} - \frac{1}{2}w_{200}$, (74) $y_{Garching} - \frac{1}{2}h_{Garching} \leq y_{200} - \frac{1}{2}h_{200}$,
 (75) $x_{Garching} + \frac{1}{2}w_{Garching} \geq x_{200} + \frac{1}{2}w_{200}$, (76) $y_{Garching} + \frac{1}{2}h_{Garching} \geq y_{200} + \frac{1}{2}h_{200}$,
 (77) $x_{Garching} - \frac{1}{2}w_{Garching} \leq x_{200} - \frac{1}{2}w_{800}$, (78) $y_{Garching} - \frac{1}{2}h_{Garching} \leq y_{200} - \frac{1}{2}h_{800}$,
 (79) $x_{Garching} + \frac{1}{2}w_{Garching} \geq x_{200} + \frac{1}{2}w_{800}$, (80) $y_{Garching} + \frac{1}{2}h_{Garching} \geq y_{200} + \frac{1}{2}h_{800}$,
 (81) $x_{Garching} - \frac{1}{2}w_{Garching} \leq x_{200} - \frac{1}{2}w_{1200}$, (82) $y_{Garching} - \frac{1}{2}h_{Garching} \leq y_{200} - \frac{1}{2}h_{1200}$,
 (83) $x_{Garching} + \frac{1}{2}w_{Garching} \geq x_{200} + \frac{1}{2}w_{1200}$, (84) $y_{Garching} + \frac{1}{2}h_{Garching} \geq y_{200} + \frac{1}{2}h_{1200}$,

 (85) $x_{London} - \frac{1}{2}w_{London} \leq x_{350} - \frac{1}{2}w_{350}$, (86) $y_{London} - \frac{1}{2}h_{London} \leq y_{350} - \frac{1}{2}h_{350}$,
 (87) $x_{London} + \frac{1}{2}w_{London} \geq x_{350} + \frac{1}{2}w_{350}$, (88) $y_{London} + \frac{1}{2}h_{London} \geq y_{350} + \frac{1}{2}h_{350}$,
 (89) $x_{London} - \frac{1}{2}w_{London} \leq x_{1650} - \frac{1}{2}w_{1650}$, (90) $y_{London} - \frac{1}{2}h_{London} \leq y_{1650} - \frac{1}{2}h_{1650}$,
 (91) $x_{London} + \frac{1}{2}w_{London} \geq x_{1650} + \frac{1}{2}w_{1650}$, (92) $y_{London} + \frac{1}{2}h_{London} \geq y_{1650} + \frac{1}{2}h_{1650}$,
 (93) $x_{London} - \frac{1}{2}w_{London} \leq x_{1750} - \frac{1}{2}w_{1750}$, (94) $y_{London} - \frac{1}{2}h_{London} \leq y_{1750} - \frac{1}{2}h_{1750}$,
 (95) $x_{London} + \frac{1}{2}w_{London} \geq x_{1750} + \frac{1}{2}w_{1750}$, (96) $y_{London} + \frac{1}{2}h_{London} \geq y_{1750} + \frac{1}{2}h_{1750}$,
 (97) $x_{London} - \frac{1}{2}w_{London} \leq x_{1850} - \frac{1}{2}w_{1850}$, (98) $y_{London} - \frac{1}{2}h_{London} \leq y_{1850} - \frac{1}{2}h_{1850}$,
 (99) $x_{London} + \frac{1}{2}w_{London} \geq x_{1850} + \frac{1}{2}w_{1850}$, (100) $y_{London} + \frac{1}{2}h_{London} \geq y_{1850} + \frac{1}{2}h_{1850}$,
 (101) $x_{London} - \frac{1}{2}w_{London} \leq x_{1900} - \frac{1}{2}w_{1900}$, (102) $y_{London} - \frac{1}{2}h_{London} \leq y_{1900} - \frac{1}{2}h_{1900}$,
 (103) $x_{London} + \frac{1}{2}w_{London} \geq x_{1900} + \frac{1}{2}w_{1900}$, (104) $y_{London} + \frac{1}{2}h_{London} \geq y_{1900} + \frac{1}{2}h_{1900}$,
 (105) $x_{London} - \frac{1}{2}w_{London} \leq x_{2000} - \frac{1}{2}w_{2000}$, (106) $y_{London} - \frac{1}{2}h_{London} \leq y_{2000} - \frac{1}{2}h_{2000}$,
 (107) $x_{London} + \frac{1}{2}w_{London} \geq x_{2000} + \frac{1}{2}w_{2000}$, (108) $y_{London} + \frac{1}{2}h_{London} \geq y_{2000} + \frac{1}{2}h_{2000}$,

 (109) $|x_{100} - x_{500}| \geq \frac{1}{2}|w_{100} + w_{500}|$, (110) $|x_{100} - x_{600}| \geq \frac{1}{2}|w_{100} + w_{600}|$,
 (111) $|x_{100} - x_{700}| \geq \frac{1}{2}|w_{100} + w_{700}|$, (112) $|x_{100} - x_{1000}| \geq \frac{1}{2}|w_{100} + w_{1000}|$,
 (113) $|x_{100} - x_{1300}| \geq \frac{1}{2}|w_{100} + w_{1300}|$, (114) $|x_{100} - x_{1400}| \geq \frac{1}{2}|w_{100} + w_{1400}|$,
 (115) $|x_{100} - x_{1600}| \geq \frac{1}{2}|w_{100} + w_{1600}|$, (116) $|x_{100} - x_{1700}| \geq \frac{1}{2}|w_{100} + w_{1700}|$,
 (117) $|x_{100} - x_{1800}| \geq \frac{1}{2}|w_{100} + w_{1800}|$, (118) $|x_{500} - x_{600}| \geq \frac{1}{2}|w_{500} + w_{600}|$,
 (119) $|x_{500} - x_{700}| \geq \frac{1}{2}|w_{500} + w_{700}|$, (120) $|x_{500} - x_{1000}| \geq \frac{1}{2}|w_{500} + w_{1000}|$,
 (121) $|x_{500} - x_{1300}| \geq \frac{1}{2}|w_{500} + w_{1300}|$, (122) $|x_{500} - x_{1400}| \geq \frac{1}{2}|w_{500} + w_{1400}|$,
 (123) $|x_{500} - x_{1600}| \geq \frac{1}{2}|w_{500} + w_{1600}|$, (124) $|x_{500} - x_{1700}| \geq \frac{1}{2}|w_{500} + w_{1700}|$,
 (125) $|x_{500} - x_{1800}| \geq \frac{1}{2}|w_{500} + w_{1800}|$, (126) $|x_{600} - x_{700}| \geq \frac{1}{2}|w_{600} + w_{700}|$,
 (127) $|x_{600} - x_{1000}| \geq \frac{1}{2}|w_{600} + w_{1000}|$, (128) $|x_{600} - x_{1300}| \geq \frac{1}{2}|w_{600} + w_{1300}|$,
 (129) $|x_{600} - x_{1400}| \geq \frac{1}{2}|w_{600} + w_{1400}|$, (130) $|x_{600} - x_{1600}| \geq \frac{1}{2}|w_{600} + w_{1600}|$,
 (131) $|x_{600} - x_{1700}| \geq \frac{1}{2}|w_{600} + w_{1700}|$, (132) $|x_{600} - x_{1800}| \geq \frac{1}{2}|w_{600} + w_{1800}|$,
 (133) $|x_{700} - x_{1000}| \geq \frac{1}{2}|w_{700} + w_{1000}|$, (134) $|x_{700} - x_{1300}| \geq \frac{1}{2}|w_{700} + w_{1300}|$,
 (135) $|x_{700} - x_{1400}| \geq \frac{1}{2}|w_{700} + w_{1400}|$, (136) $|x_{700} - x_{1600}| \geq \frac{1}{2}|w_{700} + w_{1600}|$,
 (137) $|x_{700} - x_{1700}| \geq \frac{1}{2}|w_{700} + w_{1700}|$, (138) $|x_{700} - x_{1800}| \geq \frac{1}{2}|w_{700} + w_{1800}|$,
 (139) $|x_{1000} - x_{1300}| \geq \frac{1}{2}|w_{1000} + w_{1300}|$, (140) $|x_{1000} - x_{1400}| \geq \frac{1}{2}|w_{1000} + w_{1400}|$,
 (141) $|x_{1000} - x_{1600}| \geq \frac{1}{2}|w_{1000} + w_{1600}|$, (142) $|x_{1000} - x_{1700}| \geq \frac{1}{2}|w_{1000} + w_{1700}|$,
 (143) $|x_{1000} - x_{1800}| \geq \frac{1}{2}|w_{1000} + w_{1800}|$, (144) $|x_{1300} - x_{1400}| \geq \frac{1}{2}|w_{1300} + w_{1400}|$,
 (145) $|x_{1300} - x_{1600}| \geq \frac{1}{2}|w_{1300} + w_{1600}|$, (146) $|x_{1300} - x_{1700}| \geq \frac{1}{2}|w_{1300} + w_{1700}|$,
 (147) $|x_{1300} - x_{1800}| \geq \frac{1}{2}|w_{1300} + w_{1800}|$, (148) $|x_{1400} - x_{1600}| \geq \frac{1}{2}|w_{1400} + w_{1600}|$,
 (149) $|x_{1400} - x_{1700}| \geq \frac{1}{2}|w_{1400} + w_{1700}|$, (150) $|x_{1400} - x_{1800}| \geq \frac{1}{2}|w_{1400} + w_{1800}|$,
 (151) $|x_{1600} - x_{1700}| \geq \frac{1}{2}|w_{1600} + w_{1700}|$, (152) $|x_{1600} - x_{1800}| \geq \frac{1}{2}|w_{1600} + w_{1800}|$,

$$(153) \quad |x_{1700} - x_{1800}| \geq \frac{1}{2} |w_{1700} + w_{1800}|,$$

$$\begin{aligned} (154) \quad & |x_{400} - x_{900}| \geq \frac{1}{2} |w_{400} + w_{900}|, (155) \quad |x_{400} - x_{1100}| \geq \frac{1}{2} |w_{400} + w_{1100}|, \\ (156) \quad & |x_{400} - x_{1500}| \geq \frac{1}{2} |w_{400} + w_{1500}|, (157) \quad |x_{400} - x_{1620}| \geq \frac{1}{2} |w_{400} + w_{1620}|, \\ (158) \quad & |x_{400} - x_{1720}| \geq \frac{1}{2} |w_{400} + w_{1720}|, (159) \quad |x_{400} - x_{1820}| \geq \frac{1}{2} |w_{400} + w_{1820}|, \\ (160) \quad & |x_{400} - x_{2100}| \geq \frac{1}{2} |w_{400} + w_{2100}|, (161) \quad |x_{900} - x_{1100}| \geq \frac{1}{2} |w_{900} + w_{1100}|, \\ (162) \quad & |x_{900} - x_{1500}| \geq \frac{1}{2} |w_{900} + w_{1500}|, (163) \quad |x_{900} - x_{1620}| \geq \frac{1}{2} |w_{900} + w_{1620}|, \\ (164) \quad & |x_{900} - x_{1720}| \geq \frac{1}{2} |w_{900} + w_{1720}|, (165) \quad |x_{900} - x_{1820}| \geq \frac{1}{2} |w_{900} + w_{1820}|, \\ (166) \quad & |x_{900} - x_{2100}| \geq \frac{1}{2} |w_{900} + w_{2100}|, (167) \quad |x_{1100} - x_{1500}| \geq \frac{1}{2} |w_{1100} + w_{1500}|, \\ (168) \quad & |x_{1100} - x_{1620}| \geq \frac{1}{2} |w_{1100} + w_{1620}|, (169) \quad |x_{1100} - x_{1720}| \geq \frac{1}{2} |w_{1100} + w_{1720}|, \\ (170) \quad & |x_{1100} - x_{1820}| \geq \frac{1}{2} |w_{1100} + w_{1820}|, (171) \quad |x_{1100} - x_{2100}| \geq \frac{1}{2} |w_{1100} + w_{2100}|, \\ (172) \quad & |x_{1500} - x_{1620}| \geq \frac{1}{2} |w_{1500} + w_{1620}|, (173) \quad |x_{1500} - x_{1720}| \geq \frac{1}{2} |w_{1500} + w_{1720}|, \\ (174) \quad & |x_{1500} - x_{1820}| \geq \frac{1}{2} |w_{1500} + w_{1820}|, (175) \quad |x_{1500} - x_{2100}| \geq \frac{1}{2} |w_{1500} + w_{2100}|, \\ (176) \quad & |x_{1620} - x_{1720}| \geq \frac{1}{2} |w_{1620} + w_{1720}|, (177) \quad |x_{1620} - x_{1820}| \geq \frac{1}{2} |w_{1620} + w_{1820}|, \\ (178) \quad & |x_{1620} - x_{2100}| \geq \frac{1}{2} |w_{1620} + w_{2100}|, (179) \quad |x_{1720} - x_{1820}| \geq \frac{1}{2} |w_{1720} + w_{1820}|, \\ (180) \quad & |x_{1720} - x_{2100}| \geq \frac{1}{2} |w_{1720} + w_{2100}|, (181) \quad |x_{1820} - x_{2100}| \geq \frac{1}{2} |w_{1820} + w_{2100}|, \end{aligned}$$

$$(182) \quad |x_{200} - x_{800}| \geq \frac{1}{2} |w_{200} + w_{800}|, (183) \quad |x_{200} - x_{1200}| \geq \frac{1}{2} |w_{200} + w_{1200}|,$$

$$(184) \quad |x_{800} - x_{1200}| \geq \frac{1}{2} |w_{800} + w_{1200}|,$$

$$\begin{aligned} (185) \quad & |x_{350} - x_{1650}| \geq \frac{1}{2} |w_{350} + w_{1650}|, (186) \quad |x_{350} - x_{1750}| \geq \frac{1}{2} |w_{350} + w_{1750}|, \\ (187) \quad & |x_{350} - x_{1850}| \geq \frac{1}{2} |w_{350} + w_{1850}|, (188) \quad |x_{350} - x_{1900}| \geq \frac{1}{2} |w_{350} + w_{1900}|, \\ (189) \quad & |x_{350} - x_{2000}| \geq \frac{1}{2} |w_{350} + w_{2000}|, (190) \quad |x_{1650} - x_{1750}| \geq \frac{1}{2} |w_{1650} + w_{1750}|, \\ (191) \quad & |x_{1650} - x_{1850}| \geq \frac{1}{2} |w_{1650} + w_{1850}|, (192) \quad |x_{1650} - x_{1900}| \geq \frac{1}{2} |w_{1650} + w_{1900}|, \\ (193) \quad & |x_{1650} - x_{2000}| \geq \frac{1}{2} |w_{1650} + w_{2000}|, (194) \quad |x_{1750} - x_{1850}| \geq \frac{1}{2} |w_{1750} + w_{1850}|, \\ (195) \quad & |x_{1750} - x_{1900}| \geq \frac{1}{2} |w_{1750} + w_{1900}|, (196) \quad |x_{1750} - x_{2000}| \geq \frac{1}{2} |w_{1750} + w_{2000}|, \\ (197) \quad & |x_{1850} - x_{1900}| \geq \frac{1}{2} |w_{1850} + w_{1900}|, (198) \quad |x_{1850} - x_{2000}| \geq \frac{1}{2} |w_{1850} + w_{2000}|, \\ (199) \quad & |x_{1900} - x_{2000}| \geq \frac{1}{2} |w_{1900} + w_{2000}|. \end{aligned}$$

Bibliography

- [AT06] ATLAS group at LINA & INRIA: *ATL: Atlas Transformation Language*. 2006. [http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual\[v0.7\].pdf](http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual[v0.7].pdf) (cited 2006-11-10).
- [BB98] Badros, G. J.; Borning, A.: *The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation*. Technical report, University of Washington, Department of Computer Science and Engineering, 1998.
- [BCR06] Broy, M.; Cengarle, M. V.; Rumpe, B.: *Semantics for UML - Towards a System Model for UML - The Structural Data Model*. Technical report, Technische Universität München, 2006.
- [BL04] Buttelmann, M.; Lohmann, B.: *Optimierung mit Genetischen Algorithmen und eine Anwendung zur Modellreduktion*. *at - Automatisierungstechnik*, 52:151–163, 2004.
- [BM03] Braun, P.; Marschall, F.: *BOTL - The Bidirectional Object Oriented Transformation Language*. <http://wwwbib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0307.pdf> (cited 2006-11-10), 2003.
- [Br98] Broy, M.: *Informatik: Eine grundlegende Einführung, Band 1: Programmierung und Rechnerstrukturen*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 1998.
- [Br99] Brandes, U.: *Layout of Graph Visualizations*. Doctoral thesis, Fakultät für Mathematik und Informatik, Universität Konstanz, 1999.
- [Br05a] Brendebach, K.: *Integrierte Modelle und Sichten für das IT-Management*. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2005.
- [Br05b] Bronstein, I. et al.: *Taschenbuch der Mathematik*. Harri Deutsch Verlag, Frankfurt am Main, 6 edition, 2005.

- [Bu05] Buckl, S.: *Modell-basierte Transformationen von Informationsmodellen zum Management von Anwendungslandschaften*. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2005.
- [Ch05] Chen, Y. P., Editor. *Bioinformatics Technologies*. Springer-Verlag, Berlin, Heidelberg, New York, 2005.
- [CT06] Cruz, I. F.; Tamassia, R.: *Graph Drawing Tutorial*. <http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf> (cited 2006-10-13), 2006.
- [DB99] Di Battista, G. et al.: *Graph Drawing - Algorithms for the visualization of graphs*. Prentice-Hall, London, 10th edition, 1999.
- [DV02] Domokos, P.; Varró, D.: *An Open Visualization Framework for Metamodel-Based Modeling Languages*. *Electronic Notes in Theoretical Computer Science*, 72(2), 2002.
- [EP02] Erk, K.; Priebe, L.: *Theoretische Informatik - Eine umfassende Einführung*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 2002.
- [Ev98] Evans, A. S.: *Reasoning with UML Class Diagrams*. In *WIFT'98 - Workshop on Industrial-Strength Formal Specification Techniques*. IEEE Press, 1998.
- [Fa99] Fahrmeier, L. et al.: *Statistik - Der Weg zur Datenanalyse*. Springer-Verlag, Berlin, Heidelberg, New York, 1999.
- [Ga86] Gallier, J. H.: *Logic for Computer Science: Foundations of Automatic Theorem Proving*. John Wiley & Sons, New York, USA, 1st edition, 1986.
- [Ga94] Gamma, E. et al.: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Inc., Reading, Harlow, Menlo Park, Berkeley, 1st edition, 1994.
- [Ha04] Halbhuber, T.: *Entwicklung eines Informationsmodells für das IT-Management*. Diploma thesis, Fakultät für Maschinenwesen, Technische Universität München, 2004.
- [HGM02] Hake, G.; Grünreich, D.; Meng, L.: *Kartographie*. de Gruyter, Berlin, New York, 8th edition, 2002.
- [IE00] IEEE: *IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems*. 2000. IEEE Computer Society.
- [JLL06] Jonkers, H.; van Leeuwen, D.; Lankhorst, M.: *ArchiMate Visio Stencils*. https://doc.telin.nl/dscgi/ds.py/Get/File-32177/ArchiMate_Visio_stencils.zip (cited 2006-02-21), 2006.
- [JM03] Jünger, M.; Mutzel, P.: *Graph Drawing Software*. Springer-Verlag, Basel, Switzerland, 2003.

- [La05] Lauschke, S.: *Softwarekartographie: Analyse und Darstellung der IT-Landschaft eines mittelständischen Unternehmens*. Bachelor thesis, Fakultät für Informatik, Technische Universität München, 2005.
- [LMW05a] Lankes, J.; Matthes, F.; Wittenburg, A.: *Architekturbeschreibung von Anwendungslandschaften: Softwarekartographie und IEEE Std 1471-2000*. In (Liggesmeyer, P.; Pohl, K.; Goedicke, M., Editor): *Software Engineering 2005*, volume P-64 of *Lecture Notes in Informatics (LNI) - Proceedings*, pages 43–54, Essen, Germany, 2005. Köllen Druck+Verlag.
- [LMW05b] Lankes, J.; Matthes, F.; Wittenburg, A.: *Softwarekartographie als Beitrag zum Architekturmanagement*. In (Aier, S.; Schönherr, M., Editor): *Unternehmensarchitekturen und Systemintegration*, volume 3. Gita, Berlin, Germany, 2005.
- [LMW05c] Lankes, J.; Matthes, F.; Wittenburg, A.: *Softwarekartographie: Systematische Darstellung von Anwendungslandschaften*. In (Ferstl, O. K. et al., Editor): *Wirtschaftsinformatik 2005*, pages 1443–1462, Bamberg, Germany, 2005. Physica-Verlag.
- [Ma91] Mandelbrot, B. B.: *Die fraktale Geometrie der Natur*. Birkhäuser, Basel, Switzerland, 1991.
- [Mo04] Moore, B. et al.: *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. <http://www.redbooks.ibm.com/redbooks/pdfs/sg246302.pdf>, 2004.
- [MW04] Matthes, F.; Wittenburg, A.: *Softwarekarten zur Visualisierung von Anwendungslandschaften und ihren Aspekten - Eine Bestandsaufnahme*. Technical Report TB0402, Software Engineering betrieblicher Informationssysteme; Lehrstuhl für Informatik 19 (sebis); Institut für Informatik; Technische Universität München, 2004.
- [OM01] OMG: *MDA Guide Version 1.0.1*. 2001.
- [OM05a] OMG: *UML 2.0 Infrastructure Specification (formal/05-07-05)*. 2005.
- [OM05b] OMG: *Unified Modeling Language: Superstructure, version 2.0 (formal/05-07-04)*. 2005.
- [OM06a] OMG: *Meta Object Facility (MOF) Core Specification, version 2.0 (formal/06-01-01)*. 2006.
- [OM06b] OMG: *OCF 2.0 Specification, Version 2.0 (formal/2006-05-01)*. 2006.
- [Ri05] Riihinen, J.: *Email correspondence between J. Riihinen (Director, Chief Enterprise Architect, Nokia) and R. Schlossar (BOC Information Systems)*, 2005.
- [Sc01] Scheer, A.-W.: *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*. Springer-Verlag, Berlin, Heidelberg, New York, 4th edition, 2001.
- [Sc04] Schmidt, D. A.: *Programming Language Semantics*, chapter 98. Chapman and Hall/CRC, 2004.

- [se05] sebis: *Enterprise Architecture Management Tool Survey 2005*, 2005.
- [Sl03] Slocum, T. A. et al.: *Thematic Cartography and Geographic Visualization*. Prentice Hall, 2 edition, 2003.
- [Sp92] Spivey, J.: *The Z Notation, A Reference Manual*. Prentice-Hall, New York, 1992.
- [St04] Steen, M. et al.: *Supporting Viewpoint-Oriented Enterprise Architecture*. Technical report, Information Centre of Telematica Instituut AND University of Kent, Enschede, Netherlands & Canterbury, United Kingdom, 2004.
- [To04] van der Torre, L. et al.: *Landscape Maps for Enterprise Architectures*. Technical report, Information Centre of Telematica Instituut, Enschede, Netherlands, 2004.
- [We05] Weisstein, E. W.: *Pathwise-Connected from Mathworld—A Wolfram Web Resource*. <http://mathworld.wolfram.com/Pathwise-Connected.html> (cited 2005-11-19), 2005.

List of Figures

1.1	Example software map showing the dependencies between business processes, business applications, and organizational units	3
2.1	Exemplary cluster map showing which business application is hosted at which location	6
2.2	Semantic model of the cluster map in Figure 2.1	7
2.3	Information model of the cluster map in Figure 2.1	7
2.4	Symbolic model of the cluster map in Figure 2.1	8
2.5	Visualization model of the cluster map in Figure 2.1	8
2.6	Cluster map with an additional map symbol	9
2.7	Model transformation techniques in software map generation	11
2.8	Exemplary transformation rule set	13
2.9	Excerpt of the symbolic model in Figure 2.4	14
2.10	Inequality systems derived from the symbolic model in Figure 2.9	14
3.1	Excerpt from a cluster map	16
3.2	Excerpt from a process support map	17
3.3	Relationship between organization unit, business process and business application	18
3.4	Excerpt from an interval map	19
3.5	Layers of a software map	20
4.1	Three-tiered structure of the visualization model	22

4.2	Package structure of the visualization model	24
4.3	Object-oriented core visualization model (Package <code>CoreVisualizationModel</code>)	25
4.4	Package <code>SoCaVisualizationModel</code>	26
4.5	Package <code>SoCaVisualizationModel::MapSymbols</code>	26
4.6	Package <code>SoCaVisualizationModel::ConstraintVisualizationRule</code>); <i>Property</i> is <code>EMOF::Property</code> [OM06a]	29
4.7	Layering principle and the map canvas in Package <code>SoCaVisualizationModel</code>	30
4.8	Two semantically equivalent software maps of different aesthetic appeal . . .	32
4.9	Package <code>SoCaVisualizationModel::TargetVisualizationRule</code>)	32
4.10	Two non touching map symbols with the corona of the circle touching the rectangle.	39
4.11	Misinterpretable map symbol: is the quadrat contained in the symbol made up by the three circles together?	39
4.12	An instance of the ellipse map symbol (indications of radii and center added)	43
4.13	The ellipse class (inherited attributes are also shown)	43
4.14	An instance of the rectangle map symbol (indications of width, height, and center added)	43
4.15	The rectangle class (inherited attributes are also shown)	43
4.16	An instance of the polyline map symbol (indications of bendpoints added) . .	44
4.17	The polyline class (inherited attributes are also shown)	44

List of Tables

2.1	Examples for map symbols	12
2.2	Example for visual variables	12
2.3	Examples for visualization rules	12
3.1	Meaning of the graphical concepts in Figure 3.1	16

List of Notations

\mathbb{P}	Set of potential presentation values for a <i>pixel</i> 33
\mathbb{B}	Set of Boolean values 33
l	Function for composing presentation values (e.g. by <i>topdressing</i> 34
ϵ	Neutral presentation value 34
\uplus	<i>Multi set</i> unification operator 34
$a[\beta]$	Assignment β on term a 34
\mathbb{O}	Set of all non-abstract subclasses of VisualizationObject 34
$\mathbb{V}(o)$	Set of all attributes of class o 35
$\mathbb{A}(o)$	Set of all association ends of class o 35
$\mathbb{W}^V(v)$	Domain of an attribute v 35
$\mathbb{W}^A(a)$	Domain of an association end a 35
\mathbb{G}	Set of all non-abstract subclasses of MapSymbol 36
\mathbb{D}	Set of all non-abstract subclasses of VisualizationRule 36
$\mathbb{E}_\beta(o)$	Set of all potential bindings β for the class o in respect to attributes (<i>environment</i>) 36
$\mathbb{I}_\beta(\mathbb{O}_i)$	Set of all distinct visualization object instances instanciable from the classes in \mathbb{O}_i 37

$o[\beta]_x$	Value assigned to the attribute x of o by β37
p_g	Predicate determining the extent of the solid body of a map symbol g ..37
r_g	Function determining the presentation information of a map symbol g ..37
\mathbb{G}_L	Set of all non-abstract subclasses of LinearMapSymbol39
\mathbb{G}_F	Set of all non-abstract subclasses of PlanarMapSymbol39
σ	Function for <i>point removal</i>40
χ	Function for determining <i>pathwise connectedness</i>40
a	Area function for planar map symbol instances40
ϕ	Function for determining the set of points belonging to the solid body of a map symbol instance41
a_s	Function for determining the <i>area</i> inhabited by a point set41
d	Function for determining the distance of two map symbol instances41
d_s	Function for determining the distance of two point sets41
$p_{\mathbb{S}}$	Function for determining the points inhabited by the solid bodies of a set \mathbb{S} of map symbol instances42
$r_{\mathbb{S}}$	Function for determining the combined presentation information of a set \mathbb{S} of map symbol instances42
\mathbb{D}_{con}	Set of all non-abstract subclasses of ConstraintVisualizationRule ...44
\mathbb{D}_{tar}	Set of all non-abstract subclasses of TargetVisualizationRule44
λ_{con}	Function providing the evaluation terms for a constraint visualization rule 45
λ_{tar}	Function providing the evaluation terms for a target visualization rule ..45
\mathbb{F}	Set of evaluation results for target visualization rules45
$p_{c(g)}$	Predicate determining the points inhabited by the closure of a planar map symbol instance52
$p_{f(g)}$	Predicate determining the points inhabited by the filling of a planar map symbol instance52
μ	Function for determining an attribute's or association end's characteristics (uniqueness, orderedness, multiplicity)73
$\mathbb{T}^V(v)$	Type applicable for the domain of an attribute v74

$\mathbb{T}^A(a)$	Type applicable for the domain of an association end a74
$\mathcal{M}(\mathbb{X})$	Set of multi-sets over a given set \mathbb{X}74
$\mathcal{S}_n(\mathbb{X})$	Set of sequences of length n over a given set \mathbb{X}74