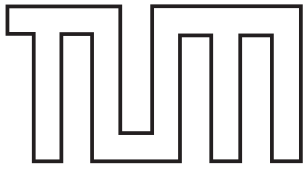


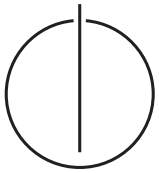
FAKULTÄT FÜR INFORMATIK

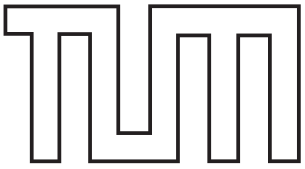
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Design and Prototypical Implementation of a Dashboard System for Visualizing Semi-Structured Data in a Traceable Way

Patrick Bürgin





FAKULTÄT FÜR INFORMATIK

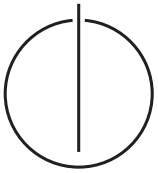
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Design and Prototypical Implementation of a Dashboard System for Visualizing Semi-Structured Data in a Traceable Way

Konzeption und prototypische Umsetzung eines Dashboard-Systems zur rückverfolgbaren Visualisierung semi-strukturierter Daten

Author:	Patrick Bürgin
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Thomas Reschenhofer, M.Sc.
Submission Date:	15.09.2015



I assure the single handed composition of this master's thesis in informatics only supported by declared resources.

Munich, 15.09.2015 Patrick Bürgin

Abstract

Dashboard systems and spreadsheet applications range among the most popular tools to support decision making, allowing users to aggregate, visualize, and share data within minutes. In today's organizations, not only the amount, but also the complexity of data is continuously growing. As the information demands of stakeholders diverge and the relationships between data objects grow more complicated, it becomes increasingly difficult to analyze the data flow in corporate environments.

In an effort to empower end-users to understand and analyze the networks spanned by the data assets, visualizations and people in their organizations, this thesis proposes a modular dashboard architecture that actively supports data flow analysis, and facilitates the use of today's web-based visualization toolkits.

Following the design science approach, the work revolves around a brief screening of related literature and prevalent industry solutions, the development of a suitable system architecture, and an evaluation of a prototypical implementation in cooperation with two industry partners.

Contents

1 Introduction

1.1 Motivation

In today's enterprises, not only the amount, but also the complexity of data is growing continuously. For data to have informational value, "it must be organized, transformed, and presented in a way that gives it meaning" [33]. Information visualization amplifies cognition, i. a. by enhancing pattern detection and reducing the search for information [4], thus supporting decision making. To this end, the creation and use of basic visualization types such as line charts and bar charts is common practice in organizations today—usually preceded by data extraction and transformation steps.

Spreadsheet applications and business intelligence (BI) suites are very popular with organizations, as they empower end-users to transform and visualize data from heterogeneous sources without programming. With few exceptions, the visualization capabilities of these tools revolve around allowing users to select from static palettes of charts with predefined variability points. Such *chart typologies*, albeit end-user friendly, inevitably "offer fewer charts than people want" [35].
In contrast, web-based visualization libraries and frameworks support the creation of sophisticated custom visualizations for arbitrarily structured data—at the cost of increased requirements in terms of tech-savviness and lowered reusability. Organizations miss out, up to the point where users resort to manual drawings when the visualization capabilities of their tools at hand cannot address a given scenario.

Information visualization in organizations involves heterogeneous data sources, cascades of data transformations, and diverging information demands of stakeholders—varying responsibilities and access restrictions account for additional complexity. In consequence, it becomes increasingly difficult to for individuals to understand and analyze the networks of data assets, visualizations and people within their organizations.
As a result, end-users of visualization systems may e.g struggle to assess the impact of changes, fail to find suitable contact persons, or have trouble reproducing calculation results presented in a visualization.

1.2 Objectives

In an effort to tackle these issues, this thesis is governed by the following questions:

RQ1: How can one combine the flexibility of web-based visualization toolkits with the ease of use of self-service visualization tools prevalent in today's organizations?

RQ2: How can individuals be supported in understanding and analyzing the networks spanned by the data assets, visualizations and people in their organizations?

In response, the thesis develops a set of fundamental concepts and instantiates them in a prototypical implementation of *a dashboard system for visualizing data in a traceable way*, which:

1. Enables the definition of custom visualization types using prevalent web-based libraries and frameworks,
2. Provides a modern, end-user friendly interface for composing and browsing data-driven dashboards, and
3. Helps users to understand and analyze the networks spanned by the data assets, visualizations, and people in their organizations.

1.3 Methodology

The previously described objectives result in a *wicked problem*, as characterized i. a. by unstable requirements and constraints, an inherent flexibility with respect to design, and critical dependence upon human cognitive abilities to produce an effective solution [17].

With their *design–science* paradigm, Hevner et al. present an information systems (IS) research framework designed to tackle such problems, seeking to "extend the boundaries of human and organizational capabilities by creating new and innovative artifacts", accompanied by a set of seven research guidelines [17].

In its effort to answer the stated questions, this thesis employs the design-science approach and implements Hevner et al.'s guidelines by creating novel, innovative, and purposeful artifacts to improve the situation specified in section 1.1. The artifacts are based on a broad knowledge base, and evaluated in cooperation with two industry partners.

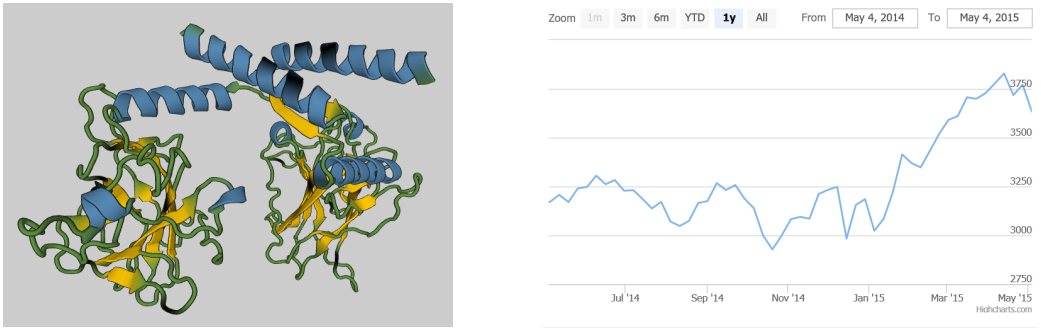
2 Background

This chapter provides an introduction to key areas of the knowledge base that drive the subsequent design and prototypical implementation in chapters 3 and 4.
Section 2.1 gives an overview on information visualization, highlighting modeling approaches and contemporary tools. Afterwards, section 2.2 discusses traceability, data governance, and metadata. Finally, section 2.3 introduces SocioCortex, a social content & model management system which serves as the foundation for the implementation.

2.1 Information Visualization

2.1.1 Terminology and Models

The term *visualization* can be defined as "the use of computer-supported, interactive, visual representations of data to amplify cognition" [4], where cognition describes the acquisition or use of knowledge. In the visualization literature, authors usually further distinguish between *scientific visualization* and *information visualization*, which differ in the types of data they represent: the former encompasses typically physically based, scientific data, whereas the latter deals with abstract, non-physically based data, such as business information or financial data [4]. Figure 2.1 provides exemplary visualizations for both classes.



(a) (b)

Figure 2.1: Examples for *scientific visualization* (a) and *information visualization* (b). The former depicts a three-dimensional protein structure using *Aquaria* [26], and the latter depicts the development of financial time series data using *Highstock* (cf. section 5.3). In line with [22], the spatial representation is *given* in the first example, whereas it is *chosen* in the second one.

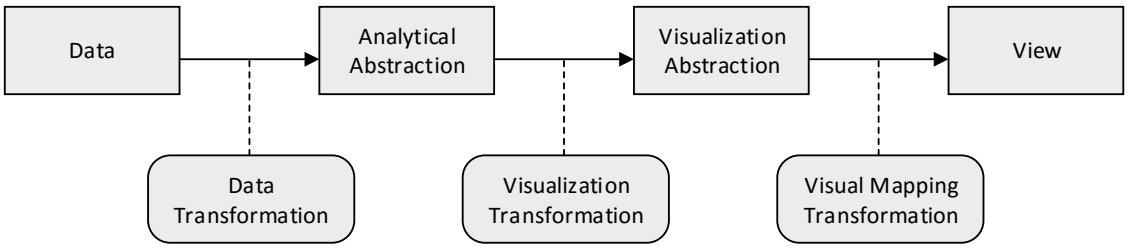


Figure 2.2: The information visualization pipeline, as modeled by Chi and Riedl [5].

Visualization can be thought of as a mapping from raw data (values) to views through a network of cascaded transformations—a data flow network often dubbed *visualization pipeline* [5, 27, 21]. Figure 2.2 shows a variant of Card's reference pipeline, as modeled by Chi and Riedl. The model distinguishes between three classes of transformations [5]:

- *data transformations*, which generate analytical abstractions from data, e.g. vector representations of text documents,
- *visualization transformations*, which transform analytical abstractions into visualizable data structures, and
- *visual mapping transformations*, which map suitable data structures into views.

With their framework for generating visualizations of arbitrary information models, Hauder et al. extend the pipeline by reusable *visualization types* (called *abstract viewpoints*), which are driven by a schema layer comprised of an information model, a view model, and a visualization model [16, 15]—figure 2.3 provides an overview on the core concepts. Noteworthy here is the visualization type's abstract view model, which captures the information demands of a given visualization type, and needs to be connected to the information model through *data bindings* on instantiation.

2.1.2 Dashboards

The term *dashboard* is used ambiguously in practice, however, most tool vendors seem to agree that it describes information displays comprised of multiple visualizations, designed to help users understand a given situation (cf. [10]). In an effort to compose a sound definition, Few defined *dashboard* as "a *visual display* of *the most important information needed to achieve one or more objectives* that has been *consolidated on a single computer screen* so it can be *monitored at a glance*" [10]. Figure 2.4 shows an exemplary dashboard based on sales data.

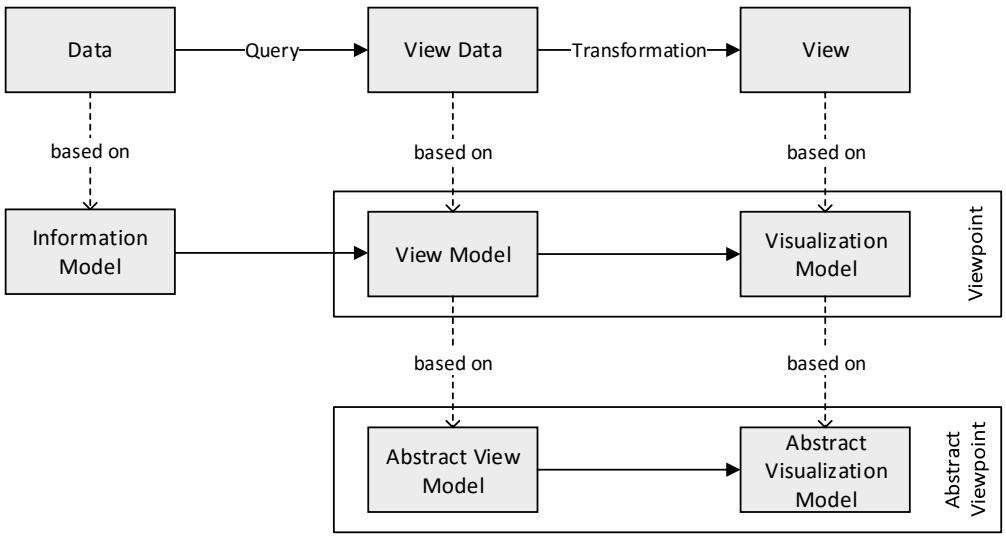


Figure 2.3: Overview of Hauder et al.'s conceptual framework for generating visualizations of arbitrary information models. Slightly modified from [16].

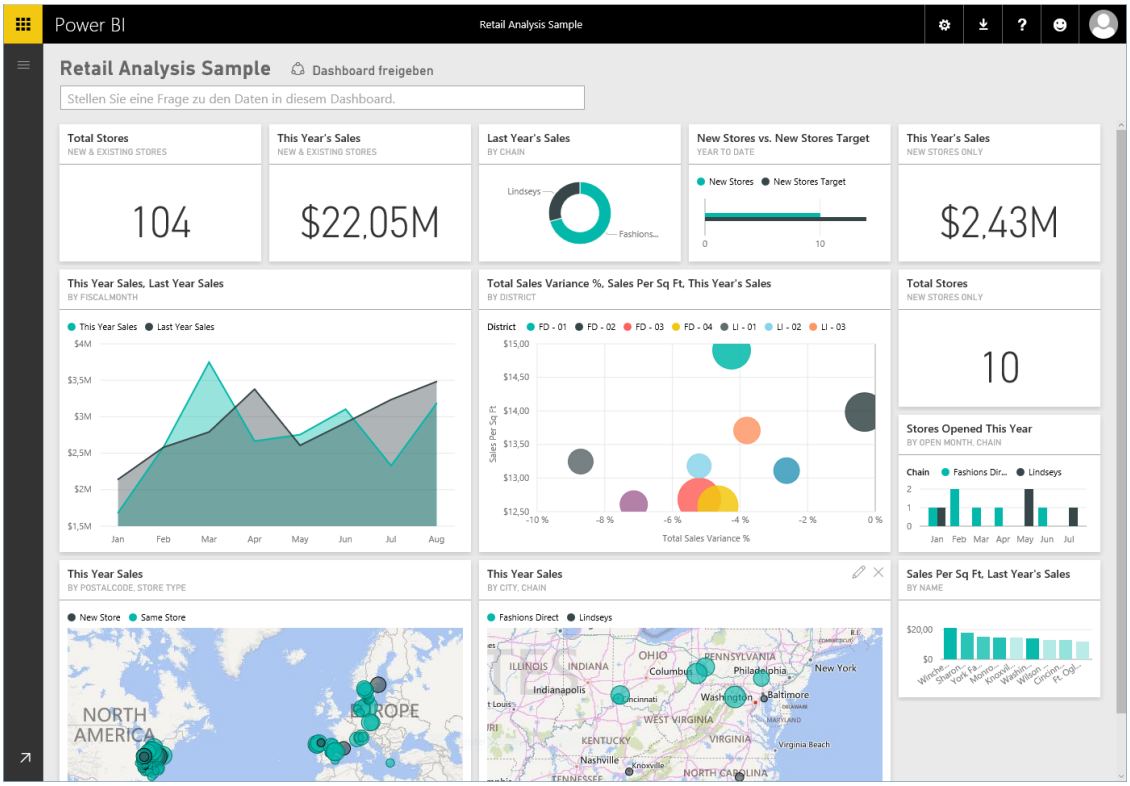


Figure 2.4: A sample dashboard in the web client of Power BI™. The dashboard is based on exemplary retail sales data of items sold across multiple stores and districts, and showcases a variety of different visualization types. Screenshot taken on August 25, 2015. Used with permission from Microsoft.

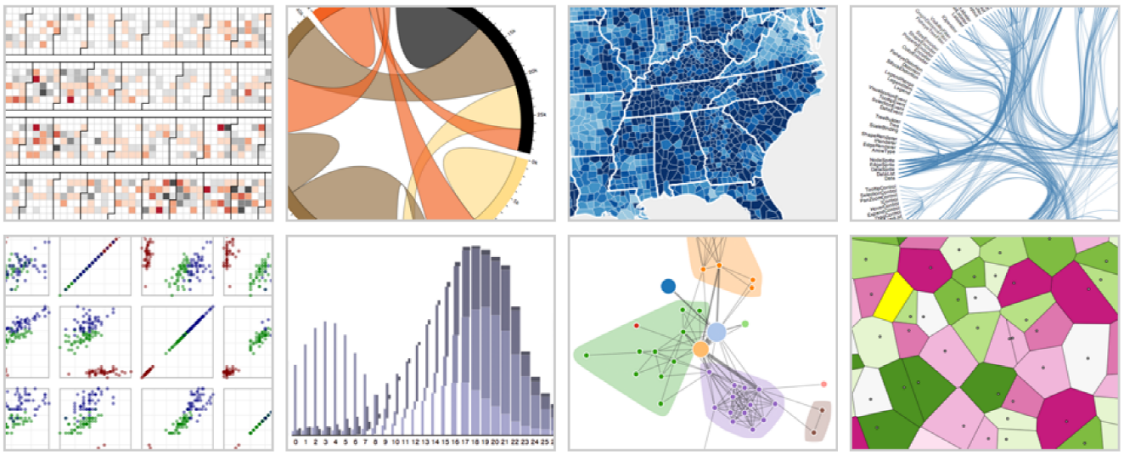


Figure 2.5: A set of different visualizations built with D3.js, including a chord diagram and a node-link diagram with force-based layout. Image extracted from [3].

2.1.3 Existing Visualization Systems

With the rising importance of data visualization, a plethora of libraries, frameworks, and software suites emerged to address different practitioner needs. As a foundation for the subsequent concept and prototype development in chapters 3 and 4, this subsection presents key concepts of two clusters of contemporary visualization systems: *Web-Based Custom Visualization Toolkits* and *Business Intelligence Tools*.

Web-Based Custom Visualization Toolkits

The Web platform[1] combines a broad range of technologies which support the creation and presentation of visualizations in modern browsers, such as the *Hypertext Markup Language* (HTML), *Scalable Vector Graphics* (SVG), *Cascading Style Sheets* (CSS), and the scripting language *JavaScript*. Over the course of the years, developers have created a broad range of visualization frameworks and toolkits on top of this stack to address different practitioner needs.

At the time of writing, the JavaScript library *D3.js*[2] [3] is the de facto standard for specifying dynamic data visualizations on the web. D3 is format agnostic with respect to input data and accepts everything from simple vectors of numbers to arrays of deeply nested objects, thus empowering users to program custom visualizations that match their individual needs, as illustrated by Figure 2.5.

In an effort to make these capabilities more accessible to a broader audience, Satyanarayan and Heer introduced an interactive environment called *Lyra*, which enables the design of custom visualizations without writing code [32]. Following a set of first-use

[1] Refer to `http://edward.oconnor.cx/2009/05/what-the-web-platform-is` for a discussion on what the *Web platform* is and what it is not. Last accessed on 2015-08-29

[2] D3.js – Data-Driven Documents: `http://d3js.org/`

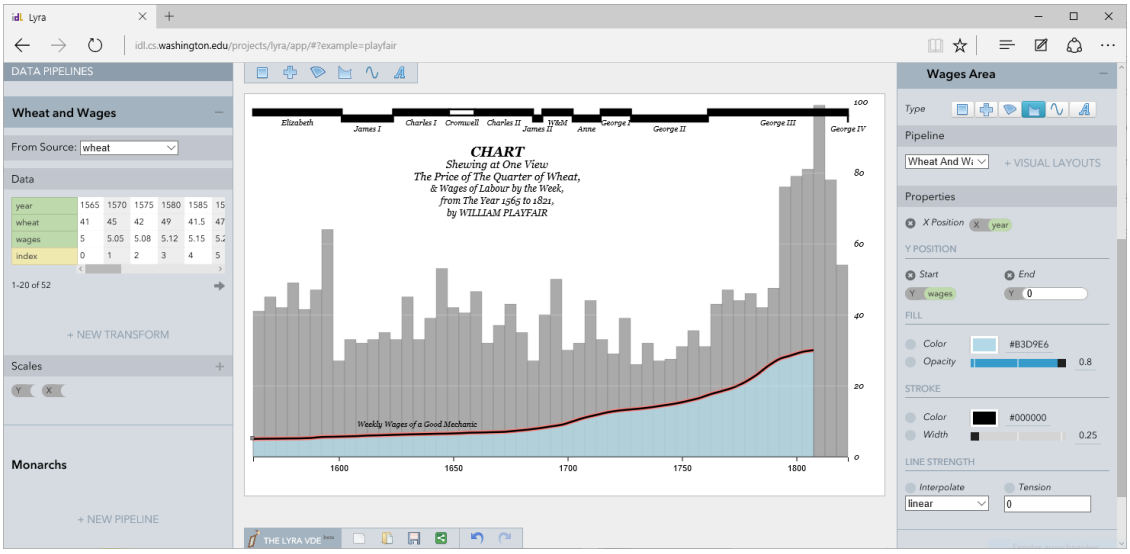


Figure 2.6: A screenshot of the Lyra Visualization Design Environment as introduced in [32]. It shows a predefined usage example, designed to showcase the tool's ability to enable the design of custom visualizations without writing code. Screenshot taken on September 8, 2015.

studies with representative users, the authors conclude that their environment enables users to "create custom visualizations much more quickly than with current tools". Figure 2.6 shows a screenshot of a the *Lyra Visualization Design Environment (VDE)*.

Business Intelligence Tools

Business intelligence (BI) can be defined as "an umbrella term that includes the applications, infrastructure and tools, and best practices that enable access to and analysis of information to improve and optimize decisions and performance" [13]. For the 2015 version of Gartner Inc.'s influential "Magic Quadrant for Business Intelligence and Analytics Platforms", analysts Sallam et al. decided to consider "IT-Developed Reporting and Dashboards" and "Analytic Dashboards and Content" as critical capabilities for current BI tools [31]. Their compiled list of "leaders" includes, among others, Qlik Technologies Inc., MicroStrategy Inc., and Microsoft Corp., with the respective products *Qlik Sense*[3], *MicroStrategy Analytics*[4], and *Power BI*[5].
Most prevalent BI tools, including the aforementioned ones, employ *chart typologies*, allowing users to select from predefined palettes of visualization types—palettes which "inevitably will offer fewer charts than people want" [35]. However, an increasing number of BI vendors aims to bridge this gap by allowing users to extend their palettes

[3]Qlik Sense: `http://www.qlik.com/us/explore/products/sense`
[4]MicroStrategy Analytics: `http://www.microstrategy.com/us/analytics`
[5]Power BI: `https://powerbi.microsoft.com/`

with custom visualizations: Qlik Sense offers an extension API which enables external developers to integrate custom visualizations built using JavaScript, HTML and CSS[6]; MicroStrategy Analytics and Power BI have been announced to offer similar APIs in upcoming releases.[7,8]

[6]Qlik Sense – Introduction to Extensions: https://community.qlik.com/docs/DOC-7033
[7]MicroStrategy 10 Webcast: https://www.youtube.com/watch?v=J0MFuztdDqo
[8]Custom Visuals in Power BI: https://powerbi.microsoft.com/custom-visuals

2.2 Traceability & Data Governance

2.2.1 Traceability

The concept of traceability originates from requirements management, where it refers to the ability to "describe and follow the life of a requirement, in both a forwards and backwards direction" [14]—from its origins all the way to deployment and use. In the more general context of software development, traceability can be regarded as "*any relationship* that exists between artifacts involved in the software-engineering-life cycle" [1], where *artifacts* can take a variety of forms, including source code and documents [6]. Aizenbud-Reshef et al. claim that "a well-defined, automated method of accomplishing traceability would be of value in any domain, on any project, with any methodology" [1]—e.g. by recording rationales and facilitating the impact analysis of change [6].
As discussed by Beier, additional benefits may ensue if one visualizes the networks spanned by artifacts and their relationships [2]. He argues that such visualizations may aid developers in understanding complex systems, and support communication between stakeholders by generating a shared holistic perspective. Possible visualization approaches include lists, dependency matrices, and node-link diagrams.

2.2.2 Data Governance

As organizations become more serious about leveraging data assets and complying to mandates such as the Sarbanes-Oxley (USA) and Basel II (Europe), the need to deploy programs for the governance of data assets increases.
To this end, Khatri and Brown introduced a framework for data governance, specifying five decision domains that should be addressed by practitioners: *data principles*, *data quality*, *data access*, *data lifecycle*, and *metadata* [18]. To aid in governing these domains, they define a set of potential roles, such as the *Data Owner*—a role which will be used representatively in subsequent chapters.

2.2.3 Metadata

Metadata can play an important role in discovering relevant information, organizing electronic resources, and facilitating interoperability. The term can be defined as "structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource", or simply put, data about data [23]. Singh et al. destinguish between five classes [34]:

Physical Metadata: Information about the physical storage of data items.

Domain-Independent Metadata: General metadata attributes that can be applied to data items of all domains, such as information about creators and contributors.

Domain-Specific Metadata: Additional attributes that encompass relevant information for the respective application domain.

Virtual Organization Metadata: Attributes that address organization-specific demands.

User Metadata: Individual attributes by users, f.i. comments and ratings.

In order to support the analysis and exchange of metadata, numerous schemes have been proposed to standardize attributes in different domains. One example is the *Dublin Core Metadata Element Set* for cross-domain resource description, which specifies a set of 15 elements, including *title*, *creator*, *contributor*, and *type* [19].
Metadata management initiatives in business intelligence environments can yield numerous advantages, initial training of new employees, impact analysis and support of data interpretation [7].

2.3 SocioCortex

SocioCortex is platform for social content & model management designed to support collaborative work. The system offers a temporal database for semi-structured content, dynamic information models, as well as typed queries and functions. It has been developed by our research group and serves as the foundation for the prototypical implementation discussed in chapter 4.

2.3.1 Dynamic Information Models

With its dynamic information model, SocioCortex allows users to incrementally and collaboratively enrich semi-structured entities with types, attribute definitions and relations. The model is based on Matthes, Neubert, and Steinhoff's *Hybrid-Wiki data model* [20], and distinguishes between the aforementioned *entities*, and *types*, as well as *workspaces*, at its core, as depicted in Figure 2.7.

All instances of the core elements are versioned and subject to discretionary and role-based access control, thus allowing inferences about creators and collaborators.

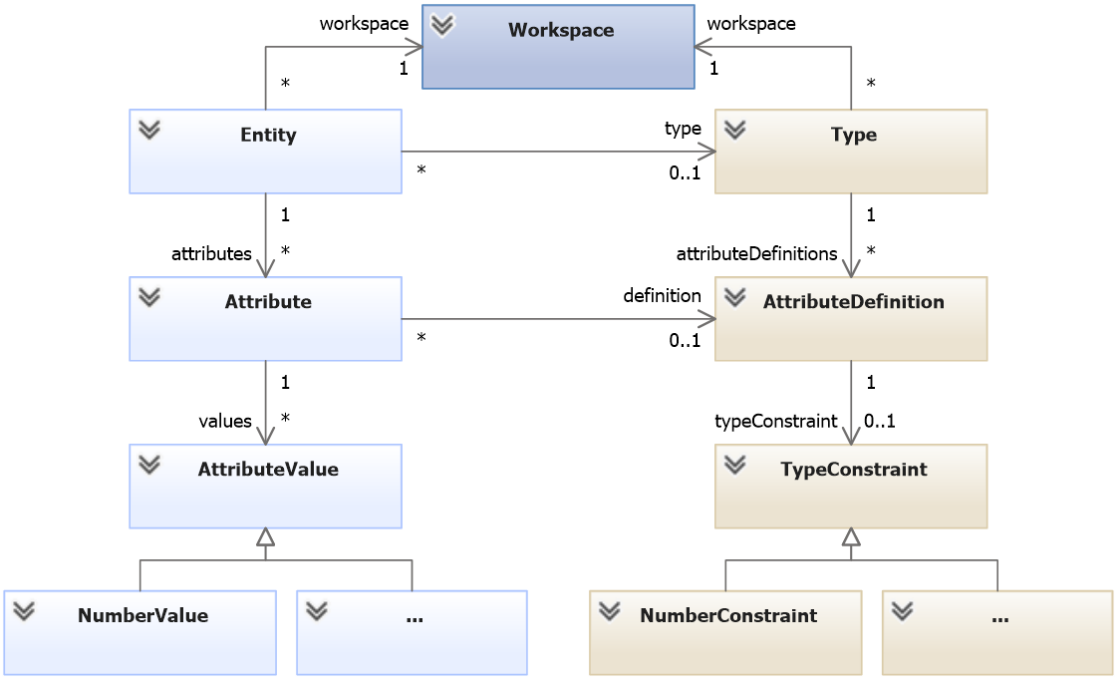


Figure 2.7: SocioCortex's instantiation of the Hybrid-Wiki data model, as introduced by Matthes, Neubert, and Steinhoff [20]. Model adapted from [28].

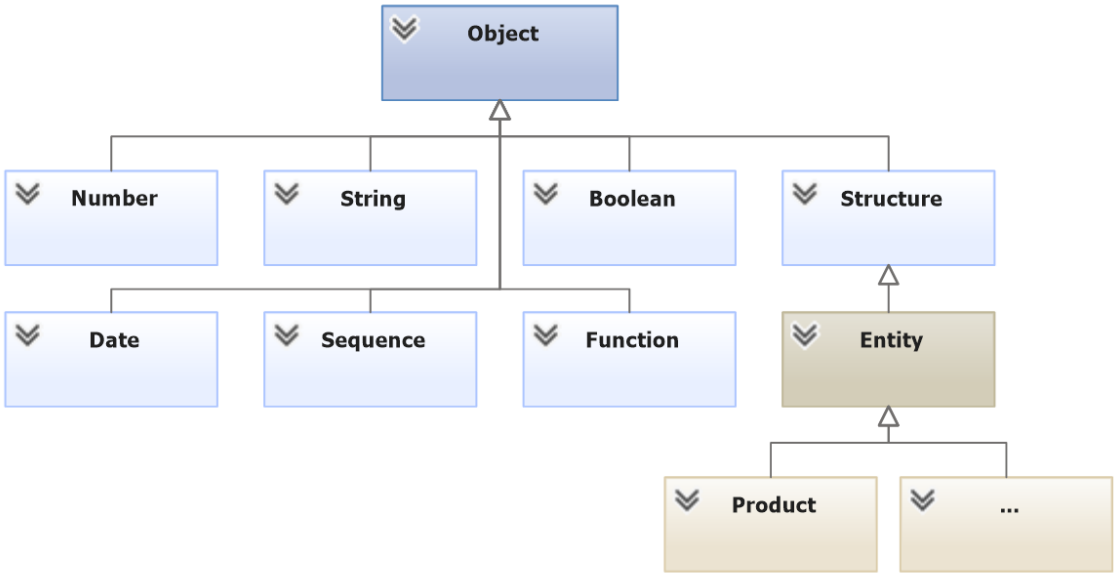


Figure 2.8: An excerpt of the type hierarchy used by SocioCortex's implementation of the model-based expression language (MxL). Adapted from [28].

2.3.2 Model-Based Expression Language (MxL)

The *model-based expression language* (MxL) is a functional, statically type-safe language, which can be used for query formulation, view definition, data transformation, data analytics, integrity rules and temporal reasoning within SocioCortex.

Originally designed to support the definition and calculation of metrics in the enterprise architecture domain [28], the language supports common query operators such as projection (*select*) and selection (*where*), aggregations, as well as common arithmetic operators. To support more complex use cases, MxL additionally offers features such as type checking, lambda expressions and higher-order functions. Figure 2.8 provides an overview of MxL's type hierarchy, which, apart from basic types such as `Number` and `String`, also includes functions, and types from the dynamic information model, as pictured by the type `Product` in the example. Furthermore, composite types like `Sequence` and `Structure` can be parameterized to specify the types of contained elements:

```
[3, 14, 15] is Sequence<Number> /* true */
[{Price: 3}, {Price: 14}] is Sequence<Structure<Price: Number>> /* true */
```

Due to the language's static type-safety, MxL expressions can be validated and analyzed at compile-time by resolving identifiers and checking their types. As a result, the language obtains all elements the expression refers to, thus enabling the construction of computation graphs. In SocioCortex, these capabilities are used to automatically adapt and validate stored expressions when referenced model elements are changed.

Figure 2.9: A screenshot of the in-browser code editor for the model-based expression language (MxL), showcasing syntax highlighting and code completion.

Expression Examples

```
find Product /* returns Sequence<Product> */

find Product
  .select(Price) /* returns Sequence<Number> */

find Product
  .select(Price)
  .sum() /* returns Number */

Sequence<Product> is Sequence<Entity> /* true */
Sequence<Product> is Sequence<Object> /* true */
Sequence<Product> is Sequence<Structure<Price: Number>> /* true */
```

In-Browser Code Editor

To facilitate the use of the language in web applications, the research group developed an in-browser code editor based on *CodeMirror*[9], an open source code editor implemented in JavaScript. Leveraging MxL's static type-safety, the component provides syntax highlighting, code completion, code navigation, error localization, and integrated documentation, as discussed in [28] and depicted in Figure 2.9.

2.3.3 Application Programming Interface

SocioCortex provides an HTTP-based application programming interface (API) that allows external applications to interact with the system. The API is split in two parts: a REST-based endpoint for creating, reading, updating, and deleting content (including types, entities, and workspaces), and an endpoint that supports the validation and execution of MxL statements.

[9] CodeMirror: `http://codemirror.net/`

3 Concept

In response to the questions raised in section 1.2, this chapter introduces fundamental models to drive a dashboard system for visualizing data in a traceable way.
Section 3.1 presents a generic model for reusable visualization types, and section 3.2 discusses how these types can be instantiated to compose dashboards. Building on this, section 3.3 sketches a holistic network model for data assets, visualizations and people, and discusses ways to extract knowledge from it.

3.1 Visualization Types

This section develops a generic meta-model for the definition of reusable, web-based visualization types, to serve as a foundation for a generic visualization system.

3.1.1 Key Requirements

VT1: Information Model Flexibility

Information models vary across data domains, use cases, and organizations:
A sales professional might think in customers and products, a financial analyst might work primarily with tick-based time series data, and a process analyst might focus on throughput times between process state.
To support the implementation of both generic and specific visualization types, ranging e.g. from simple line charts to interactive stock charts and graph visualizations, a generic schema should provide sufficient flexibility with respect to the specification of input parameters.

VT2: Adaptability

There are plenty of motives for end-users to adapt visual aspects of a visualization type—be it to conform to an organization's brand guidelines, or to emphasize a certain point. Depending on the type, such aspects might range from adapting colors, font types and the like, to adapting the parameters of a layout algorithm.
In consequence, creators of visualization types should be able to provide end-users with variability points, allowing them to adapt the visual representation as needed.

VT3: Technological Flexibility

As web-based visualization libaries, frameworks, and technologies in general, evolved significantly over the years, developers rejoice at an ever-growing range of tools available

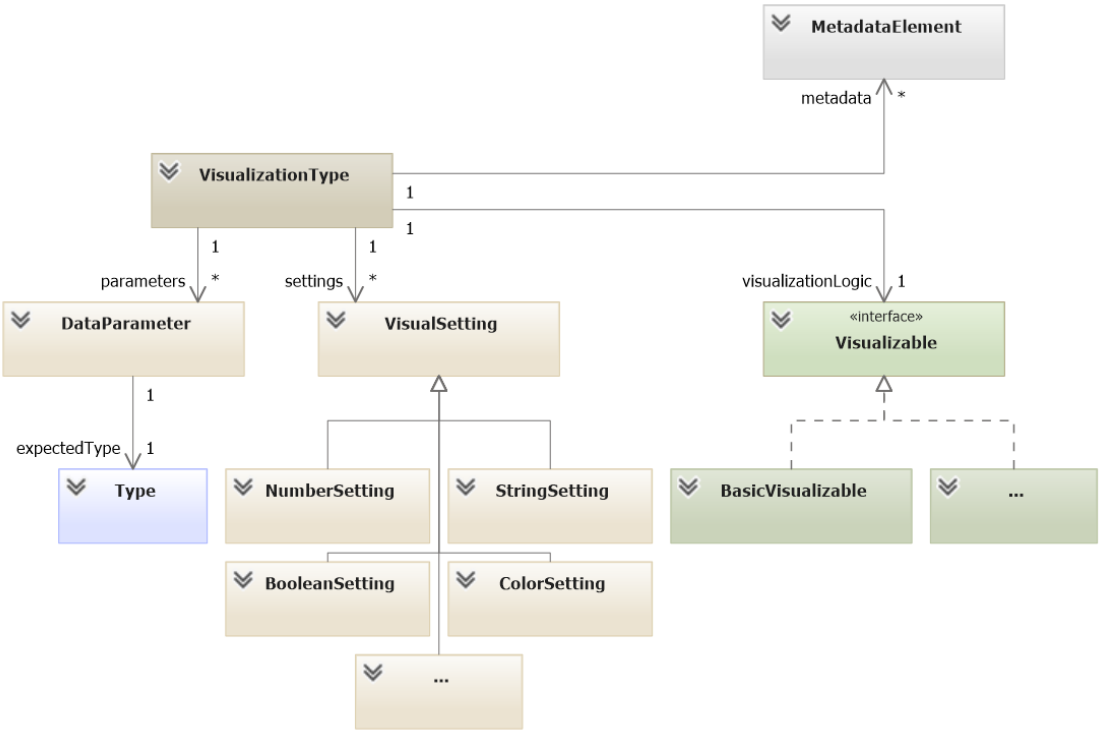


Figure 3.1: A meta-model for the definition of reusable, web-based visualization types.

for implementing visualizations. As discussed in section 2.1.3, contemporary tools range from plain JavaScript libraries like *D3.js* to visual configuration environments like *Lyra*. In order to act as an enabler rather than a barrier, a generic schema for specifying visualization types should embrace the constant change, and empower developers to adopt new tools as they emerge.

VT4: Discoverability

To facilitate the identification, discovery, and exchange of visualization types, specifications should include suitable metadata as discussed in section 2.2.3.

3.1.2 Model

In response to the previously discussed requirements, this section introduces a generic model to support the specification of visualization types.
As depicted in Figure 3.1, the model specifies a `VisualizationType` as a composition of *data parameters* (**VT1**, **VT4**), *visual settings* (**VT2**), *a visualization logic* (**VT3**), and descriptive *metadata* elements (**VT4**).

Data Parameters

The DataParameters specify a VisualizationType's information demands with the help of a type system, thus providing a sound interface to guide the instantiation of corresponding visualizations.
The heart of each DataParameter is a Type attribute named expectedType. Based on a given type system, this attribute specifies the type of data that should be bound to the parameter when using the VisualizationType. Using the syntax and type hierarchy of *MxL*, as introduced in section 2.3.2, possible values of expectedType could include String, Sequence<Number>, or Sequence<Structure<name: String>>. When using *TypeScript*[1] syntax instead, the previous examples would turn out as String, Number[], and { name: String }[].
To facilitate usage by both end-users and visualization type developers, each parameter should have a unique name, as well as a description text. To support more sophisticated parameter definitions, one could add an attribute for marking parameters as optional, add support for value restrictions, and allow overloading by changing the single expectedType to a list of allowedTypes.

The DataParameters provide high flexibility with respect to the specification of input parameters (**VT1**). Additionally, the VisualizationType's type signature, as spanned by its expectedTypes, can be used to support identification and discovery, thus supporting **VT4**. An example for the latter is *Hoogle*[2], an API search engine, which allows users to search standard *Haskell* libraries by approximate type signature.

Visual Settings

VisualSettings encompass the graphic variablility points of a VisualizationType, allowing end-users to e.g. adapt the aesthetics of a visualization at run-time (**VT2**).
To facilitate form generation and validation, different types of VisualSettings have been introduced, including ColorSetting, NumberSetting, StringSetting, BooleanSetting. Each setting should come with a suitable default value, as well as a unique name to facilitate its use.

Visualization Logic

The visualization logic is the essential part of a VisualizationType, defining the *visualization tranformations* and *visual mapping transformations* that transform data into views (cf. section 2.1.1).
In an effort to empower developers to adopt new tools and technologies as they emerge (**VT3**), the model does not enforce a specific set of technologies for defining visualization logics. Instead, it employs an interface called Visualizable, which defines a minimal set of lifecycle methods to control rendering. BasicVisualizable represents a straight

[1] TypeScript: http://www.typescriptlang.org
[2] Hoogle: https://www.haskell.org/hoogle/

forward implementation of the interface, driven by plain HTML, CSS, and JavaScript code. Such a low-level implementation allows experienced developers to reuse existing visualization libraries, and build custom visualizations using low-level frameworks like D3.js (cf. section 2.1.3). Other implementations might f.i. adapt higher-level visualization specification languages such as the visualization grammar *Vega*[3]—a declarative format for creating, saving and sharing visualization designs.

A minimal definition of the `Visualizable` interface could be comprised of a single function to be called after evaluating the bindings of all `DataParameters` at run-time, with a signature as follows:

```
render: ($element, data, settings, callback) => void;
```

`$element` : A container for the visualization to render in. This parameter could take the form of a *jQuery*[4] wrapped `div`-container, managed by the visualization system at run-time.

`data` : An object that contains the results of evaluating each `DataParameter`'s binding.

`settings` : An object that contains the values of each `VisualSetting`'s binding.

`callback` : A callback function for passing errors, if any, and notifying the visualization system once the rendering procedure has been finished.

More sophisticated visualization systems may expect implementations of `Visualizable` to implement more fine-grained sets of functions. An exemplary set could be comprised of an `init`-function, which is called before evaluating the `DataParameter`'s bindings, and a range of event handlers to deal with data updates and window resizings:

```
init: ($element, settings, callback) => void;
onDataChange: ($element, data, callback) => void;
onResize: ($element, callback) => void;
```

Metadata

As discussed in section as 2.2.3, metadata can play an important role in discovering relevant information, and organizing electronic resources. To facilitate the discovery and management of visualization types, the model can be enriched by descriptive and administrative metadata. For a start, one could add a selection of the elements specified by the previously introduced *Dublin Core* schema: e.g. title, description, dates, as well as references to the author and collaborators. As usage data and change histories may facilitate importance assessments and root cause analyses, adding both could further support decision making.

[3] Vega: `http://trifacta.github.io/vega/`

[4] *jQuery* is the most popular JavaScript library to date, and has been designed to simplify client-side scripting—see `https://jquery.com/`

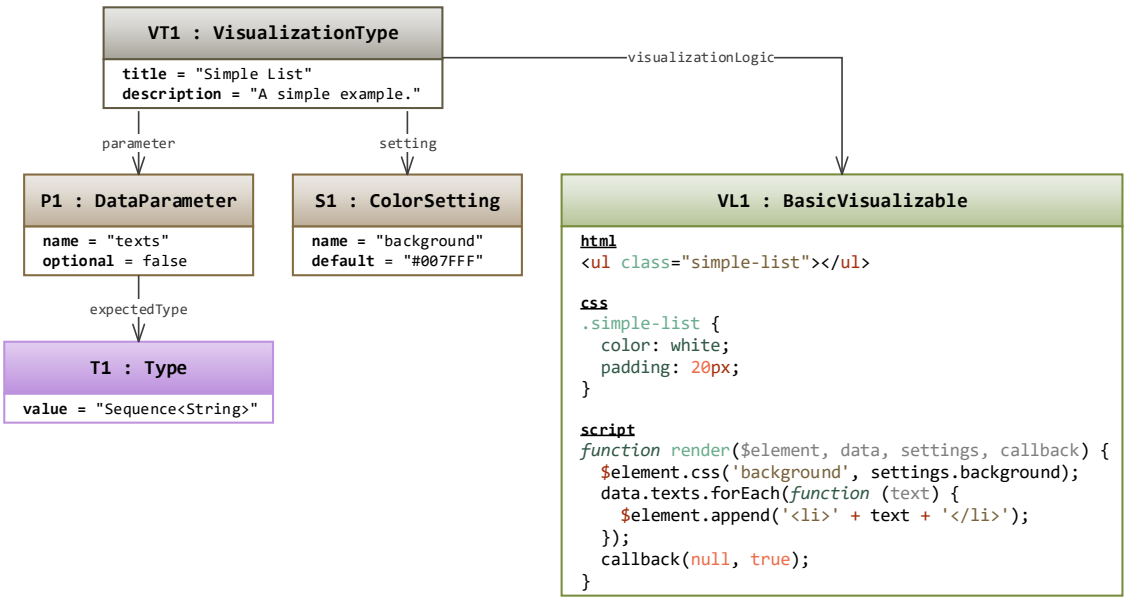


Figure 3.2: An examplary instantiation of the meta-model for reusable, web-based visualization types, depicted as a UML object diagram.

3.1.3 Example

Figure 3.2 shows an exemplary instantiation of the meta-model to illustrate the introduced concepts. The depicted model describes a *simple list* based on `BasicVisualizable`, with a single parameter and a single setting.
The instance of `DataParameter` is has a `name` attribute of value `"texts"`, and expects a return type of `Sequence<String>`. The setting is an instance of `ColorSetting`, and is supposed to allow end-users to adapt the background color. The `visualizationLogic` is split into `html`, `css`, and `script` attributes, containing the respective code snippets. The script assumes that the `data` parameter passed to the `render` function would contain an iterable property named `texts`, matching name and structure of the corresponding `DataParameter`—the same logic applies for `settings` and `background`.

3.2 Visualization Environment

This section introduces a model for binding instances of `VisualizationType` to data, yielding `Visualizations`, as well as a concept for composing `Dashboards` of multiple visualizations.

3.2.1 Key Requirements

VE1: Resilient Bindings

As instances of `VisualizationType` may evolve, and even disappear over time, a visualization's bindings to `DataParameters` and `VisualSettings` should be resilient with respect to change.

VE2: Responsiveness

In his popular book *Information Dashboard Design: Displaying data for at-a-glance monitoring*, Few emphasizes that a dashboard should fit a single screen, stating that "if the viewer must scroll around to see the all information, it is no longer a dashboard" [10] (cf. section 2.1.2). However, screen sizes, aspect ratios, and pixel densities vary across different device categories and user types, as illustrated by the results of Steam's recent *Hardware & Software Survey*[5] among its large host of desktop users[6], depicted in Figure 3.3.
As a result, a visualization environment offering dashboards should respond to the given device's screen size and display resolution, resizing labels and visualizations accordingly.

VE3: Flexible Composition

As expressed by Few, the information on a dashboard must be tailored specifically to the task at hand, otherwise it will not serve its purpose [10]. The previously introduced meta-model for reusable, web-based visualization types can be used address a broad range of different use cases.
To this end, the visualization environment should provide users with sufficient flexibility with respect to positioning and scaling the visualizations when composing a dashboard.

[5] Steam: Hardware & Software Survey: `http://store.steampowered.com/hwsurvey` (Accessed 2015-09-09)
[6] Steam is a leading computer gaming platform with up to 10 million concurrent users.
See `http://store.steampowered.com/stats` (Accessed 2015-09-09)

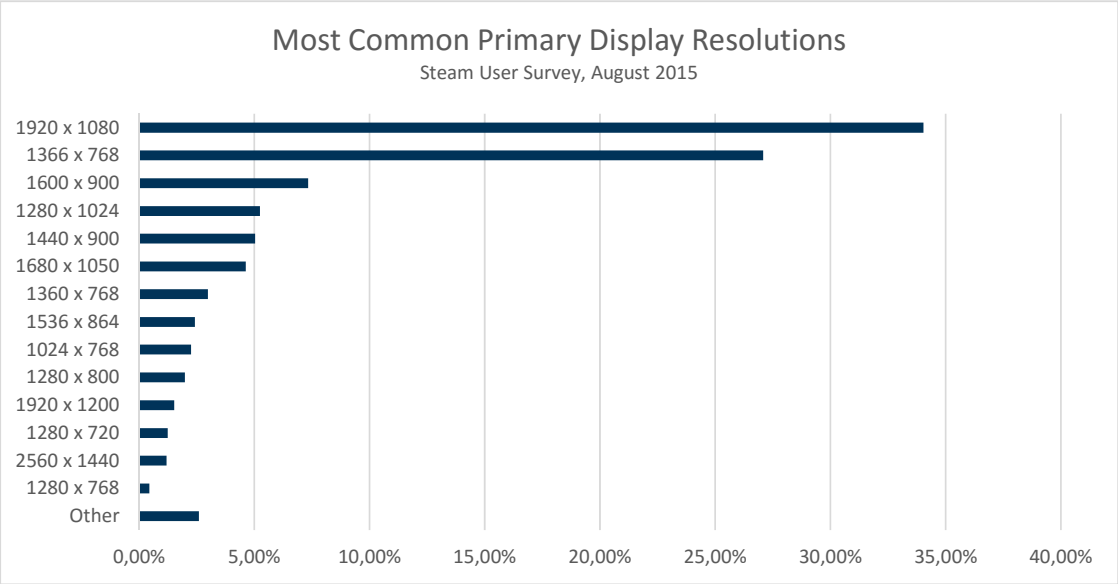


Figure 3.3: The most common primary display resolutions among participants in Steam's *Hardware & Software Survey* conducted in August 2015. Data extracted from `http://store.steampowered.com/stats` on September 9, 2015.

3.2.2 Visualizations

Following the conceptual framework for generating visualizations of arbitrary information models introduced in section 2.1.1, this section develops an approach for instantiating `Visualizations` based on `VisualizationTypes`—or, using the framework's terminology, *Viewpoints* based on *Abstract Viewpoints*.
In order to employ a given instance of `VisualizationType`, one has to bind its data parameters, and visual settings, as discussed in section 3.1.2. To this end, a `Visualization` has been modeled as a composition of data parameter bindings and visual setting bindings (**VE1**), complemented by a set of `MetadataElements` to support identification and discoverability, as depicted in Figure 3.4.

Data Parameter Bindings

The `DataParameterBindings` of a visualization make use of typed queries to bind the data parameters of its associated visualization type to a given information model.
As each parameter specifies an expected type, the corresponding bindings should be typed as well—it facilitates consistency checking in the face of possible changes to the visualization type at run-time (**VE1**). By employing a statically-typed query language, such as the *model-based expression language* (MxL) introduced in section 2.3.2, a visualization system can validate the queries at compile-time, and check whether the

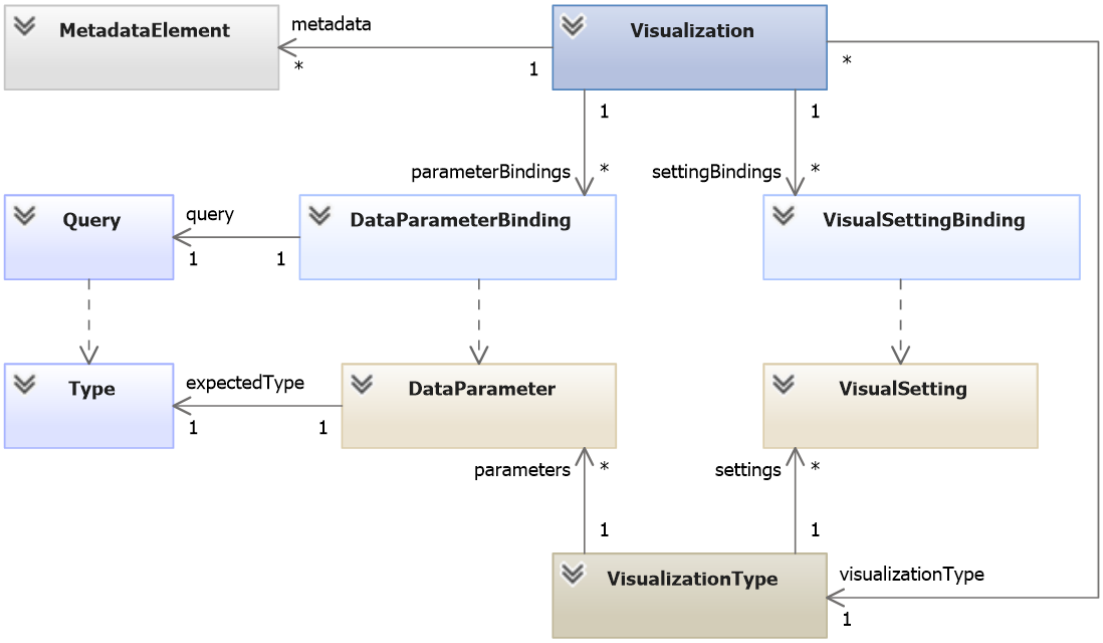


Figure 3.4: An expanded meta-model that describes the relationship between visualizations and visualization types. The dashed arrows describe dependencies.

return types match the ones expected by their associated parameters.

Apart from allowing end-users to assess whether their queries yield suitable data types, the type system creates additional opportunities for improving the user experience: By matching the information demands of a `VisualizationType`, embodied by the `expectedTypes` of its parameters, with the given information model, a visualization system can support its end-users at the formulation of suitable queries—e.g. by using structural pattern matching as discussed in [16].

Visual Setting Bindings

The `VisualSettingBindings` of a visualization allow end-users to override the default values of the associated visualization type's visual settings, thus enabling them to adapt visual aspects according to their needs. A visualization system implementing the meta-model may leverage the type attributes of settings to present users with suitable form elements, e.g. a color picker for adapting `ColorSettings`.

Example

Figure 3.5 provides an exemplary instantiation of the expanded meta-model, describing an instance of `Visualization` based on the previously modeled `VisualizationType` named *Simple List*. The visualization type employs an instance of `BasicVisualizable`, a

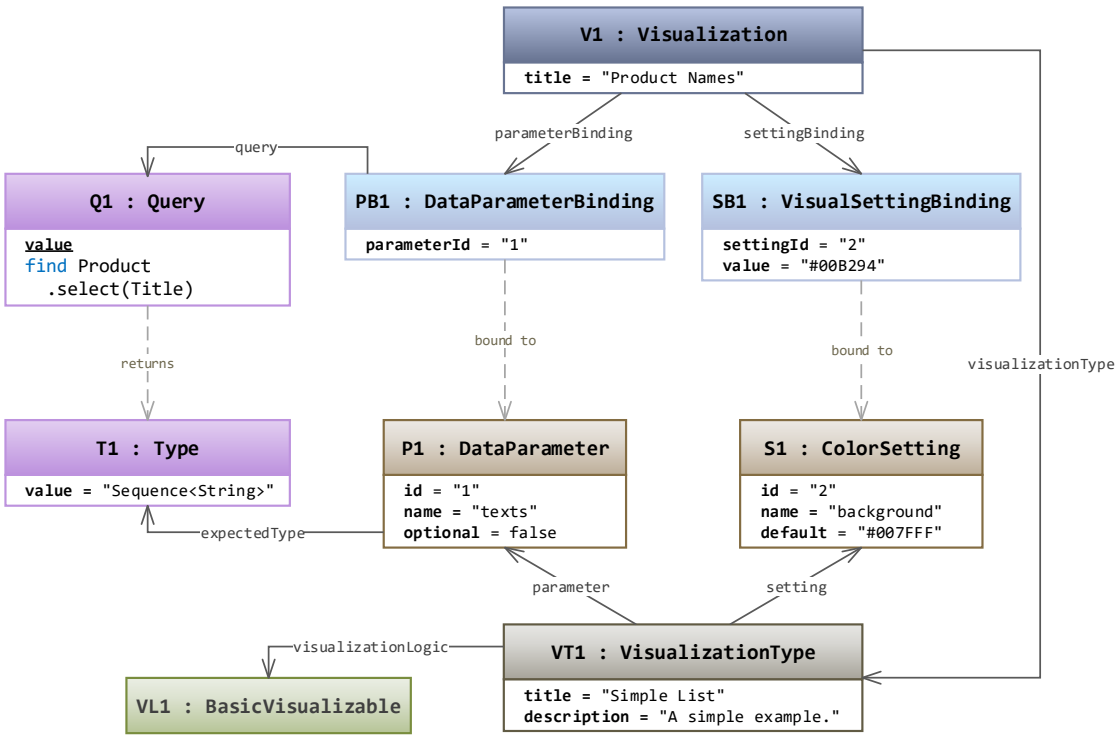


Figure 3.5: An exemplary instantiation of the expanded meta-model, showcasing the relation between `Visualizations` and `VisualizationTypes`.

parameter named *texts*, and a setting named *background*.
To bind the *texts* parameter with an expected type of `Sequence<String>`, the visualization features a `DataParameterBinding` with an *MxL*-based `Query` that requests a list of product titles—a sequence of strings. Furthermore, the visualization overrides the default value of *background* with `"#00B294"`.

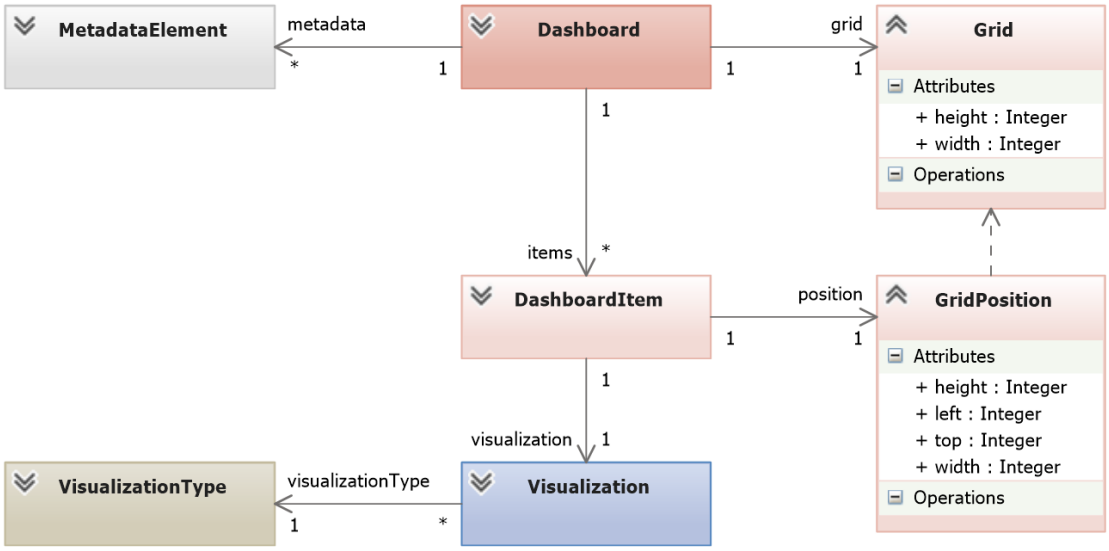


Figure 3.6: A further extension of the meta-model, specifying `Dashboards` as enriched collections of `Visualizations` placed on a `Grid`.

3.2.3 Dashboards

Based on the previously introduced meta-model comprised of `Visualizations` and `VisualizationTypes`, this section provides a further extension by introducing dashboards, as discussed in section 2.1.2.
Taking a liberal stance, a `Dashboard` can be modeled as a collection of one or more `Visualizations`, displayed together on a single screen. As motivated in the requirements **VE2** and **VE3**, the model should allow for environments that enable users to flexibly move and resize visualizations, while offering support for different screen sizes and display resolutions at the same time. As these requirements render pixel-based positioning infeasible, the model employs percentage-based positioning instead.
Figure 3.6 depicts the resulting model, which describes a `Dashboard` as a collection of *dashboard items*, each one comprised of a visualization and its position on a percentage-based *grid*, complemented by metadata to support identification and discoverability.

Grid System

In order to support flexible moving and resizing of visualizations across different viewport sizes, the `Dashboards` employ a percentage-based grid system, rather than a pixel-based set of positions and sizes.
In this system, the viewport is split into a virtual $m \times n$-grid, comprised of m rows and n columns. By acting as a universal abstraction layer, the system allows for seamless positioning across different environments, without ever violating Few's definition, which states that a dashboard should not exceed a single screen (cf. **VE2**). However, in extreme

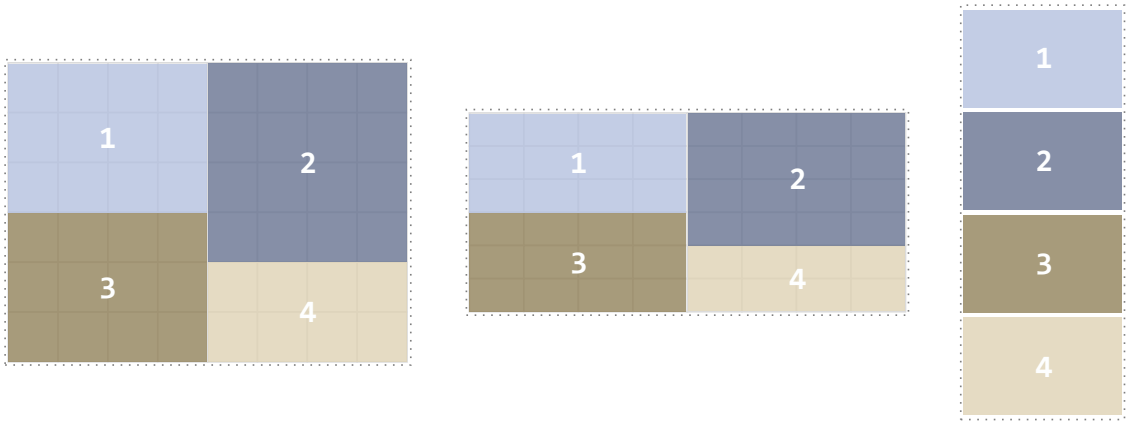


Figure 3.7: An exemplary 6×8-grid with four items, adapting to different viewport sizes. In the third case (from left), the grid layout has been abandoned to avoid highly skewed item containers, as the given viewport is comparatively narrow.

cases, e.g. when rendering a dashboard on a small widescreen-display in portrait-mode, one might abandon the grid layout and resort to scrolling, to ensure that the individual visualizations remain readable, as illustrated in Figure 3.7.

The Figure shows an exemplary 6×8-grid with four items. When denoting the position of a cell at row i and column j by (i, j), the third item could be described as a rectangle of height 3 and width 4, positioned at $(4, 1)$. Applying established web terminology and *JavaScript object notation* (JSON) instead, the `GridPosition` can be described as:

```
{ top: 4 left: 1, height: 3, width: 4 }
```

3.2.4 Example

Figure 3.8 provides a final example, illustrating a simple instantiation of the entire meta-model developed over the previous sections. In addition, the Figure shows an exemplary rendering of the `Dashboard`, its 6×8-grid, and the contained visualization of a type named *Simple List*, employing the visualization logic that has been introduced in Figure 3.2 on page 19.

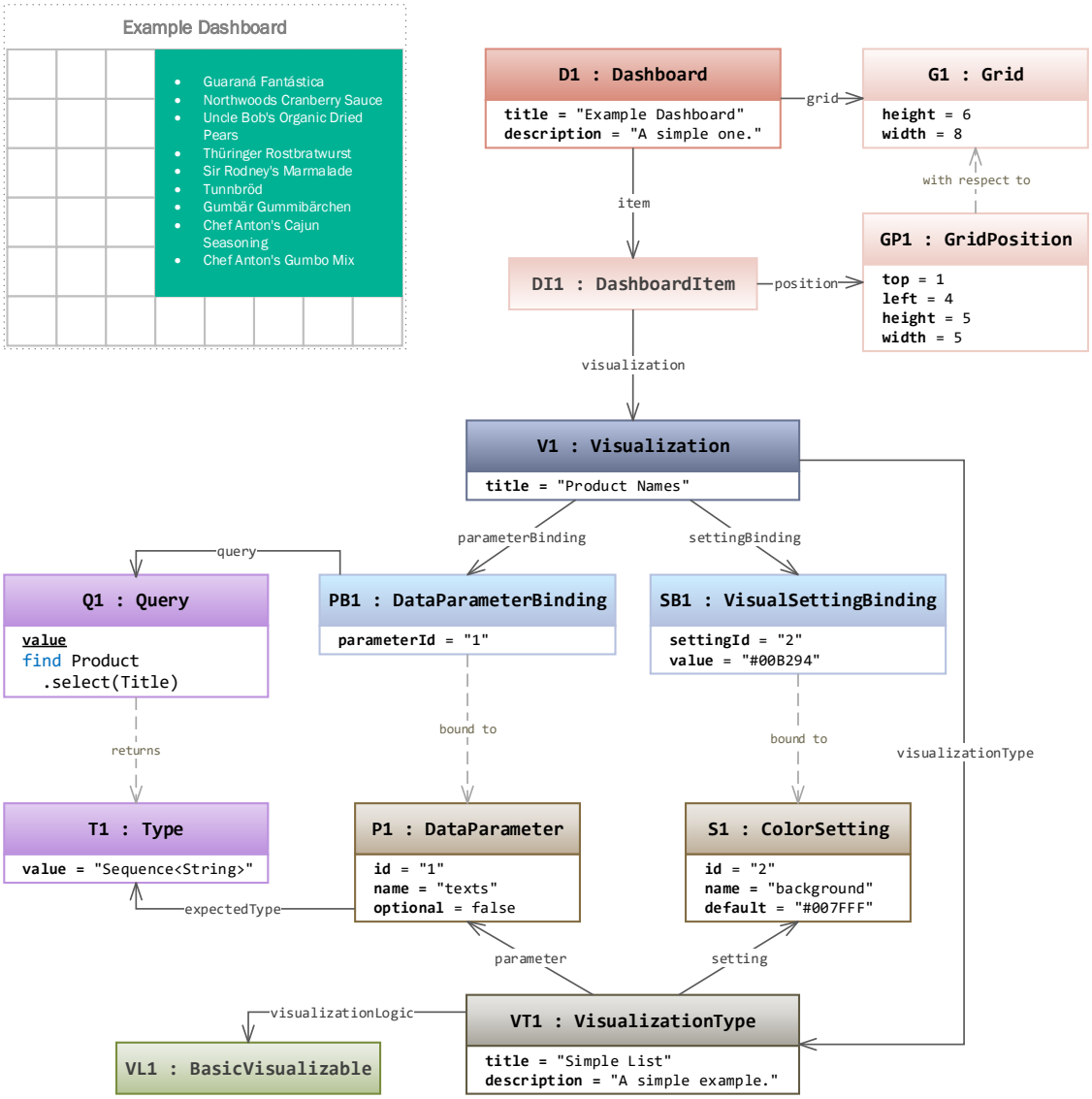


Figure 3.8: An exemplary instantiation of the introduced meta-model, showcasing a `Dashboard` based on previously shown instances of `Visualization` and `VisualizationType`. Depicted as an enriched UML object diagram.

3.3 Traceability Environment

Building on the previously introduced model for composing dashboards based on reusable visualization types, this section investigates how organizations can be supported in understanding and analyzing the networks spanned by their data assets, visualizations and people.

The section is split into two parts:

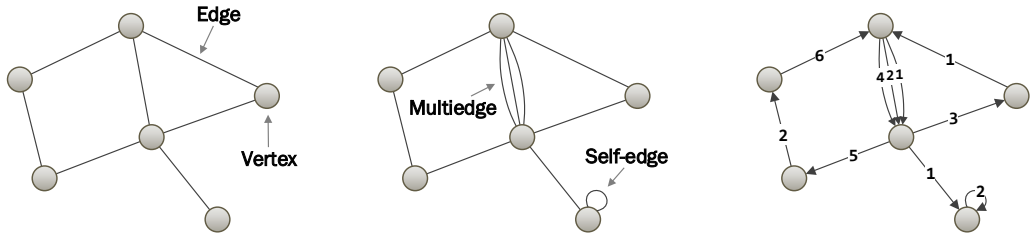
1. How to model the described network?
2. Given a suitable network model, how to analyze it?

3.3.1 Network Construction

Terminology

Formally speaking, a *network*, also known as *graph*, can be defined as a collection of *vertices* (nodes) connected by *edges* (links). As introduced e.g. by Newman in [24, Chapter 6], vertices and edges may describe a broad range of different concepts, ranging from relationships between persons, to data flows between systems.

If a network allows for multiple edges between the same pair of nodes, so-called *multiedges*, it is called a *multigraph*. If an edge connects a node to itself, it is called a *self-edge*, and if it carries a weight, e.g. the intensity of a friendship relation between two persons, it is said to be *weighted*. Furthermore, a graph is said to be *directed*, if its edges have a direction, thus pointing from one vertex to another. Figure 3.9 provides an illustration of the aforementioned concepts:



(a) (b) (c)

Figure 3.9: A simple graph (a), a multigraph with self-edges (b), and a weighted, directed multigraph with self-edges (c). Examples replicated from [24].

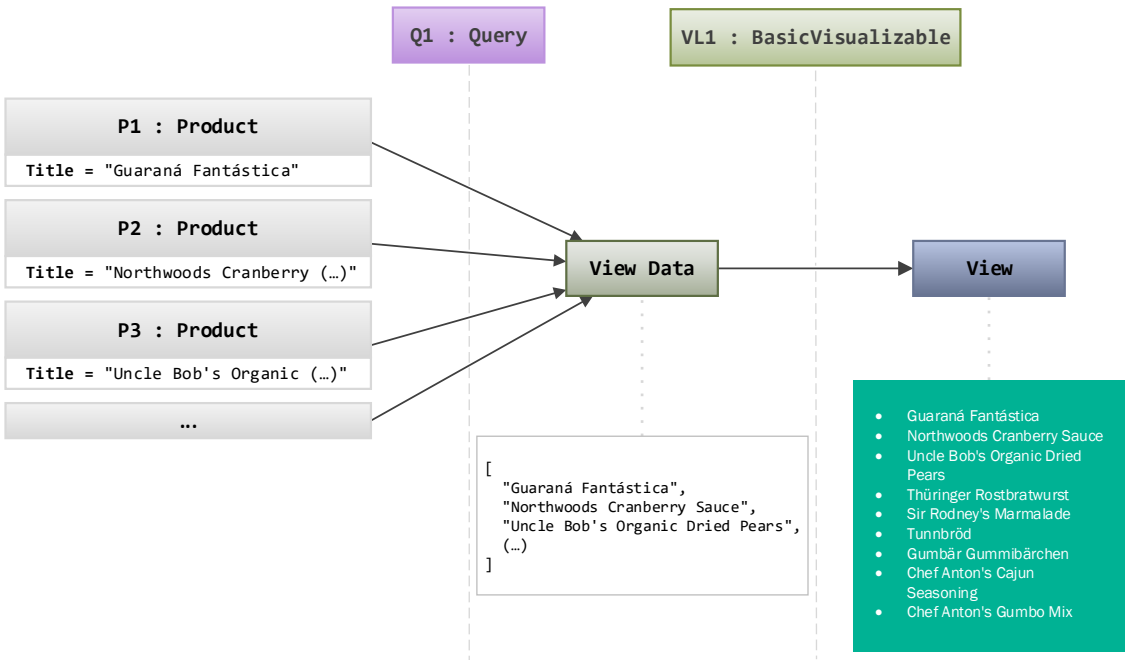


Figure 3.10: An exemplary pipeline representation of the visualization example presented in Figure 3.8 on page 27, employing a `VisualizationType` named *Simple List*.

Overview

The information visualization pipeline, as introduced in section 2.1.1, embodies a network of data flows, mapping data to views through a series of cascaded transformations. Figure 3.10 illustrates a basic pipeline representation of the visualization example presented in Figure 3.8 on page 27.

The described connections between data and views at run-time only account for a small part of the full network spanned by data assets, visualizations and people within a sophisticated visualization system. To serve as a foundation for building a traceability environment, the remainder of this section discusses a three-step approach for constructing a holistic network, resulting in a directed and potentially weighted graph with self-edges.

Step 1: Trace the Extended Visualization Pipeline

The classic information visualization pipeline can be extended by reusable visualization types, as described in section 2.1.1, and implemented by the meta-model introduced in sections 3.1 and 3.2, which adds instances of `Visualization`, `VisualizationType`, and `Dashboard`.

Given that the pipeline is supposed to represent a mapping from data to views, a

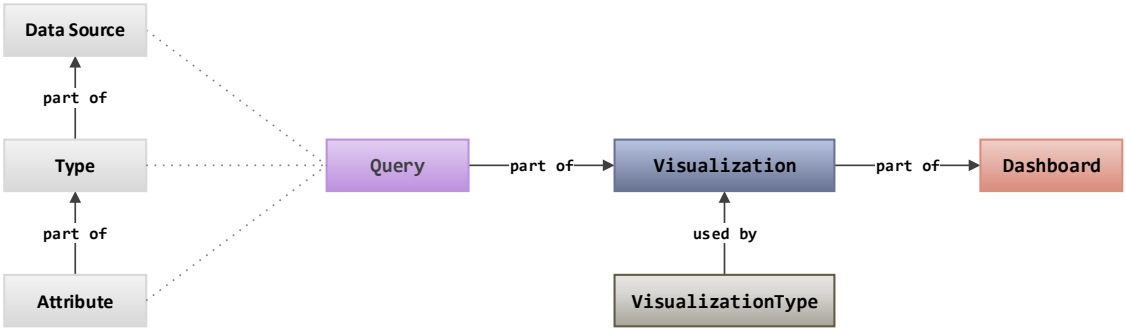


Figure 3.11: A simple network model comprised of *data sources* on the one hand, and *dashboards* on the other, as well as their associated parts.

reasonable way to start constructing the network, is to add vertices for each *dashboard*, each *visualization type*, and each *data source*. Afterwards, one can add vertices for each *visualization* referred by a dashboard, and each *query* contained in the respective parameter bindings. To provide more depth on the data side, one can add further vertices for each data type offered by the different data sources, as well as vertices for all attributes offered by the different types. Adding directed edges to express the given *part-of* and *used-by* relations, this procedure yields in the model depicted in Figure 3.11.

Step 2: Analyze the Queries

At this stage, there are only implicit relations between the queries and the data depicted on the left, as indicated by the dotted lines. To make these relations explicit, one has to analyze the queries, and recursively resolve all the references that they make.
If the queries are based on a statically-typed language such as *MxL*, this can be done in a straight forward fashion at compile-time, otherwise one has to execute the queries and perform more sophisticated analyses. Either way, Figure 3.12 illustrates the accordingly extended network model, featuring a new set of edges to represent reference-relations, as well as a new class of vertices to describe reusable functions for transforming data.

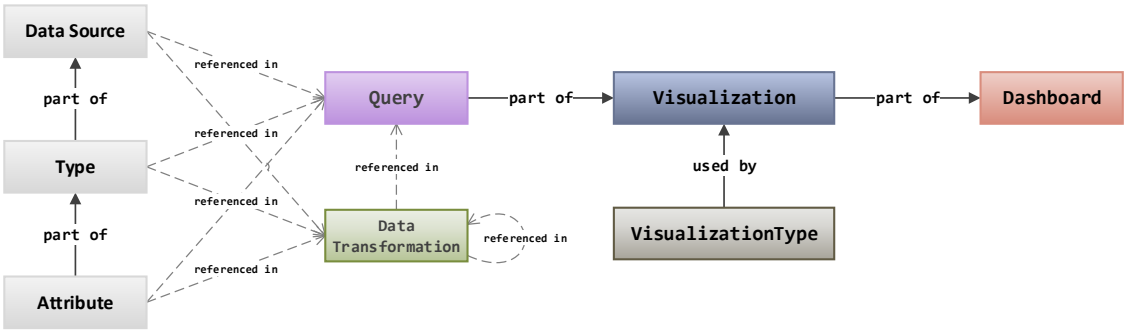


Figure 3.12: An extension to the model presented in Figure 3.11, connecting data-vertices to query-vertices by analyzing queries and recursively resolving references.

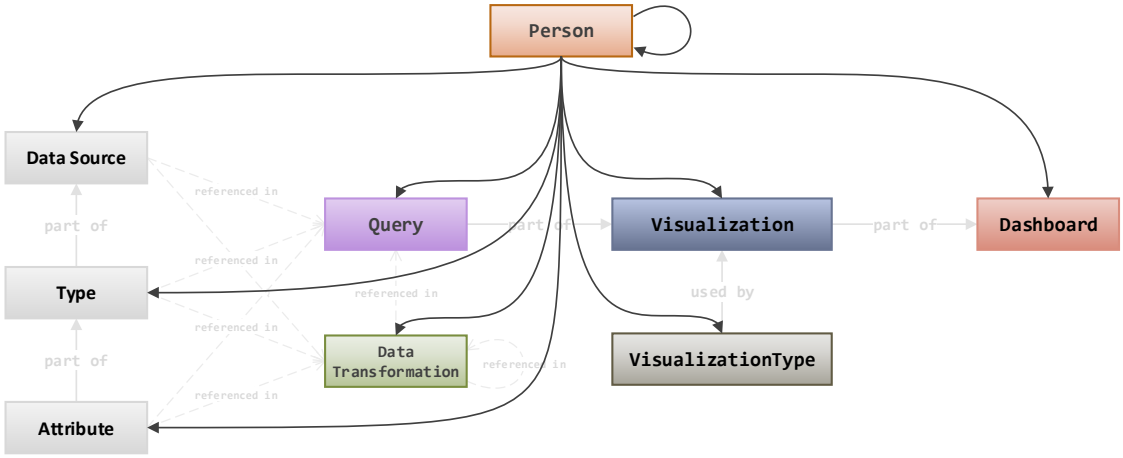


Figure 3.13: A further extension of the network, adding *person–item* and *person–person* relations from the visualization system's environment.

Step 3: Add Person–Item and Person–Person Relations

While the previous steps operate exclusively on parts of the extended visualization pipeline, thus addressing data assets and visualizations, this step adds a new dimension: persons.

To this end, the item-based network is extended by a new set of vertices, which describe all the persons that can be connected to the visualization system—e.g. by mining its user profiles. Given sufficient data, persons can be connected to other persons, for example through a mutual group membership, or a directed *follows*-relation.
There are multiple ways how a person can be tied to an item from the pipeline, including usage relations (e.g. *changed*, *clicked*, *commented*, *follows*, *liked*, ...), and references in complementary metadata attributes of pipeline-items, marking e.g. a person as the *owner* of a given item. Figure 3.13 illustrates the resulting network, highlighting the newly added edges that represent arbitrary *person–item* and *person–person* relations.

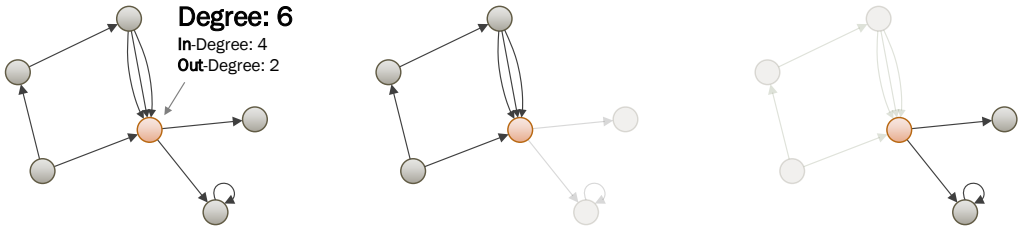
3.3.2 Network Analysis

The previously introduced network model can be used in a variety of ways to support end-users in making decisions, e.g. by facilitating impact analysis and stakeholder identification.

In order for this to work, the network data has to be made available to end-users in a sufficiently comprehensible way. However, as the number of data sources, transformations and visualizations increases, the graph can grow very complex—especially when further extending it to also cover the entities of data types. To this end, the remainder of this section discusses strategies to help end-users extract insights from the graph, most importantly via *calculating measures and metrics*, *complexity reduction*, and *visual exploration*.

Terminology

Following Newman [24], the *degree* of a given vertex is the number of edges connected to it. In a directed graph, one can distinguish between *in-degree* and *out-degree*, where the former is the number of incoming edges, and the latter is the number of outgoing edges, as exemplified in Figure 3.14a.
Given a vertex A, its *in-component* is the set of all vertices from which there is a (directed) path to reach A, including the vertex itself, and its *out-component* is the set of all vertices that can be reached via (directed) paths starting from A, as illustrated in Figures 3.14b and 3.14c.



(a) (b) (c)

Figure 3.14: Based on the selected vertex marked orange, the subfigures illustrate the different degrees (a), the in-component (b), and the out-component (c).

Calculating Measures and Metrics

The use of measures and metrics is widespread across domains, as they provide means to assess complex situations by analyzing simple numbers, thus facilitating decision making. In the realm of networks, measures and metrics allow to quantify network structure by capturing particular features of a network's topology, as discussed in [24, Chapter 7].

In the given context of networks spanned by an organization's data assets, visualizations and people, *centrality measures* can prove to be very useful. Those measures aim to describe the importance of a given vertex, thus helping end-users to identify notably relevant and irrelevant items.

A simple way to assess the *centrality* of a vertex is to compare its degree to the degrees of other vertices. In the current environment, the validity of this so-called *degree-centrality* depends on the type of vertex at hand: If a vertex represents a *data source*, the number of incoming references, as expressed by its out-degree, arguably provides an indication of its importance. The same goes e.g. for *visualization types* and their numbers of associated *visualizations*, as well as for *dashboards* and their numbers of associated *people* that expressed their interest i.a. by clicking, editing, or following. The degree of a *query* however, has no immediate significance with respect to centrality.

By construction, each vertex that represents an item of the extended visualization pipeline depends on all items in its *in-component*, while being depended on by all items in its *out-component*. Consequently, a more meaningful approach for assessing centrality in this network is to employ a recursive strategy, and express the importance of an item by the sum of its transitive inbound and outbound dependencies.

Other potentially useful measures include *similarity assessments*, facilitating e.g. the identification of duplicates and content recommendation, as well as general measures such as the number of contributors associated with a given dashboard.

Complexity Reduction

As previously discussed, the networks at hand can grow very complex. In order to facilitate their analysis, their complexity can be reduced by merging different classes of vertices, e.g. combining *dashboards*, their associated *visualizations* and all related *queries*, to composite vertices, as illustrated in Figure 3.15.
While this process may significantly reduce the complexity of the network, it may also cause a loss of information, as the relations between children of different composite elements can no longer be traced in detail—unless one employs *multiedges* that preserve the information.

Visual Exploration

Interactive visual representations of data can amplify cognition and support decision making (cf. section 2.1.1). As further discussed in section 2.2.1, visualizing the networks spanned by system artifacts and their relationships can help users understand complex systems, and may support communication between stakeholders by generating a shared holistic perspective. Possible visualization approaches in this regard include lists, dependency matrices, and node-link diagrams. With that said, a visual graph exploration environment could prove very useful to help end-users extract insights from the network model. To this end, the remainder of this section sketches cornerstones for such an

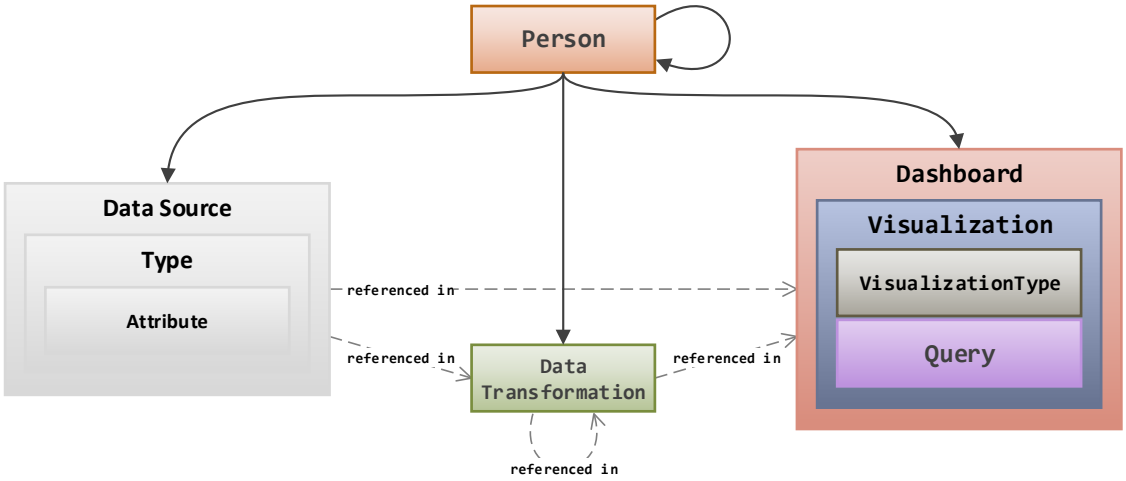


Figure 3.15: By grouping vertices according to their part-of-whole relations, the network complexity can be reduced significantly—at the cost of a loss of information.

environment, acting as a foundation for prototypically implementing a traceability environment in the subsequent chapter.

Visualization Style: The information visualization pipeline provides a comprehensible foundation for illustrating a network spanned by data assets and visualizations. By adopting its inherent layout constraints when rendering a node-link diagram, a traceability system empowers end-users to trace both inbound and outbound dependencies of each artifact in a straight forward fashion.

Complexity Reduction: As previously discussed, merging related vertices to reduce the complexity of the network may come at the cost of a loss of information. A traceability environment might avoid this trade-off by allowing its users to interactively *drill-down* on merged vertices as needed, revealing detailed relations of child elements on demand.
To allow users to browse the aggregated information for a given vertex, the system may provide a modular side panel, displaying e.g. available metadata and details about the child elements of an aggregate vertex.

Measures and Metrics: The system may additionally provide its users with measures and metrics—either displayed in a distinct panel, or integrated directly into the node-link diagram—e.g. by color coding the vertices, or adapting the edge-thickness to represent a given measure.

Figure 3.16 presents an exemplary mockup of a traceability environment based on the concepts introduced in this chapter, highlighting the in-component of the selected vertex, allowing users to drill-down on merged vertices, and displaying available metadata in a sidebar.

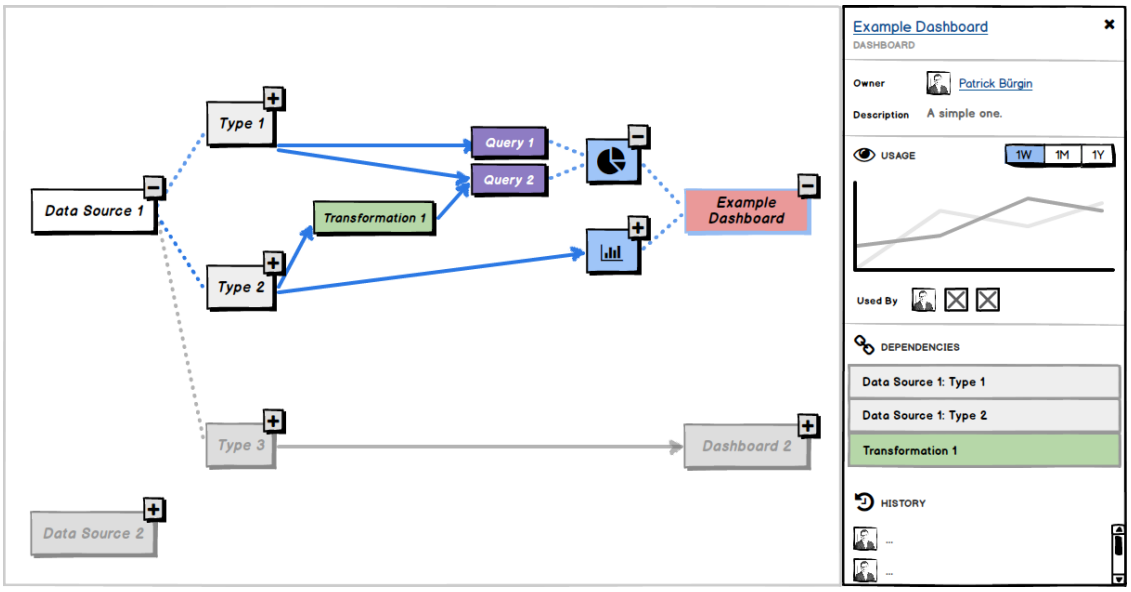


Figure 3.16: A mockup of an environment for visually exploring data from the network model introduced in section 3.3.1. The system allows users to *drill-down* and *roll-up* on the part-to-whole relations in the model, as signified by the plus- and minus-icons attached to eligible vertices.

4 Implementation

This chapter presents a prototypical implementation of the models and concepts introduced in the sections 3.1 (visualization types), 3.2 (visualizations and dashboards), and 3.3 (network model).
To provide an overview of the implemented system, section 4.1 presents its high-level architecture and its view hierarchy. Afterwards, section 4.2 shows its interface for defining visualization types, and section 4.3 shows those used to create visualizations and dashboards thereof. Finally, section 4.4 presents a first implementation of the previously sketched environment for visually exploring data from the network model.

4.1 Overview

4.1.1 Key Technologies

The prototypical implementation is a web application written entirely in

- *TypeScript*[1], a typed superset of JavaScript that compiles to plain JavaScript,
- *Sass*[2], a CSS extension language, and
- *Jade*[3], a terse language for writing HTML templates.

It is based on the open JavaScript web application framework *AngularJS*[4], and makes use of numerous other open source libraries, most notably:

Ace: An embeddable code editor written in JavaScript.[5]

Cytoscape.js: A graph theory library for analysis and visualization.[6]

Dagre: A directed graph renderer for JavaScript.[7]

D3.js: A JavaScript library for manipulating documents based on data, as introduced in section 2.1.3.[8]

[1] TypeScript: `http://www.typescriptlang.org/`
[2] Sass: `http://sass-lang.com/`
[3] Jade: `http://jade-lang.com/`
[4] AngularJS: `https://angularjs.org/`
[5] Ace: `http://ace.c9.io/`
[6] Cytoscape.js: `http://js.cytoscape.org/`
[7] Dagre: `https://github.com/cpettitt/dagre/wiki`
[8] D3.js: `http://d3js.org/`

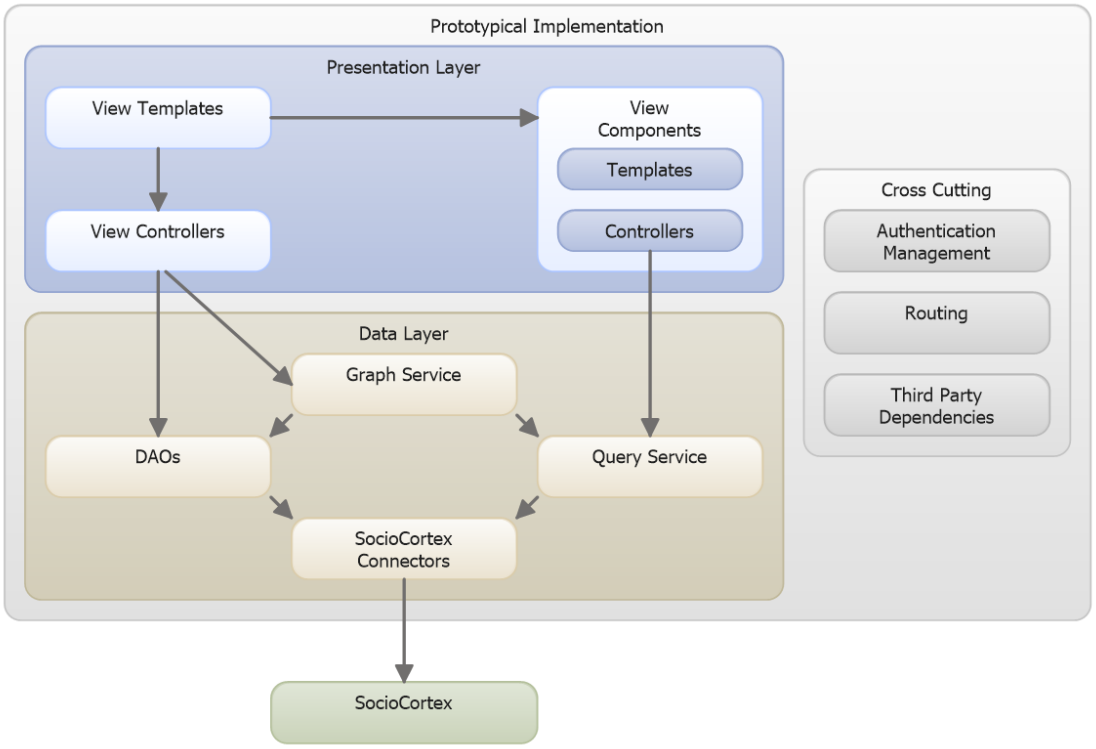


Figure 4.1: An informal layer diagram of the prototype's high level structure.

4.1.2 Architecture

The prototypical implementation is a thick, client-side web application. It attaches to the API of *SocioCortex*, the platform for social content and model management introduced in section 2.3, and uses the system as a persistence interface for the model elements introduced in chapter 3, thus profiting from its versioning capabilities. Additionally, the prototype uses SocioCortex also as a data source for visualizations in its dashboard environment, thus allowing the use of the model-based expression language (MxL) as introduced in section 2.3.2. Figure 4.1 provides a high-level overview of the prototype's architecture.

Following the general structure of the employed application framework, the prototype is split into a presentation layer (controllers, directives), and a data layer (factories, services), as well as a set of cross-cutting capabilities. Apart from data access objects (DAOs) for the model elements, the data layer contains a *query service*, and a *graph service*, designed to drive dashboards and the traceability environment.

4.1.3 Design Principles

The design of the prototypical implementation has been guided by the principles discussed by Norman in his well-cited, revised and expanded edition of *The design of*

everyday things [25]. He states that when people use a system, they face two gulfs, the *gulf of execution*, where they try to figure out how to use it, and the *gulf of evaluation*, where they try to figure out the results of their actions and the state of the system, as depicted in Figure 4.2. Arguing that the role of a designer is to "help people bridge the two gulfs", he develops a series of design principles (cf. [25, p. 72ff]), including:

Discoverability: "It is possible to determine what actions are possible and the current state of the device."

Feedback: "There is full and continuous information about the result of actions and the current state of the product or service. After an action has been executed, it is easy to determine the new state."

Signifiers: "Effective use of signifiers ensures discoverability and that the feedback is well communicated and intelligible." In this context, the term *signifier* refers to perceivable indicators that communicate appropriate behavior to users—e.g. a special cursor icon signifying that a given item can be resized.

Figure 4.2: An illustration of the gulfs of execution and evaluation, as discussed by Norman. Diagram replicated from [25, p. 39].

Visual Design

In terms of its design language, the prototype is based on the corporate design of the research group, and employs an extended color palette built using *Adobe Color CC*[9]—a tool for creating color themes with presets based on color theory.
The visual aspects have been implemented with the help of the popular front-end framework *Bootstrap*[10], and a considerable amount of custom CSS.

[9] Adobe Color CC: `https://color.adobe.com/create/color-wheel/`
[10] Bootstrap: `https://github.com/twbs/bootstrap`

4.1.4 View Hierarchy

The view hierarchy follows the three-part structure of the concept described in chapter 3, and offers three major branches, as signified by the cards on the start view: *Visualization Type Management*, a *Visualization Environment*, and a *Traceability Environment*.

Figure 4.3 provides an overview of the hierarchy, omitting peripheral views such as the login page:

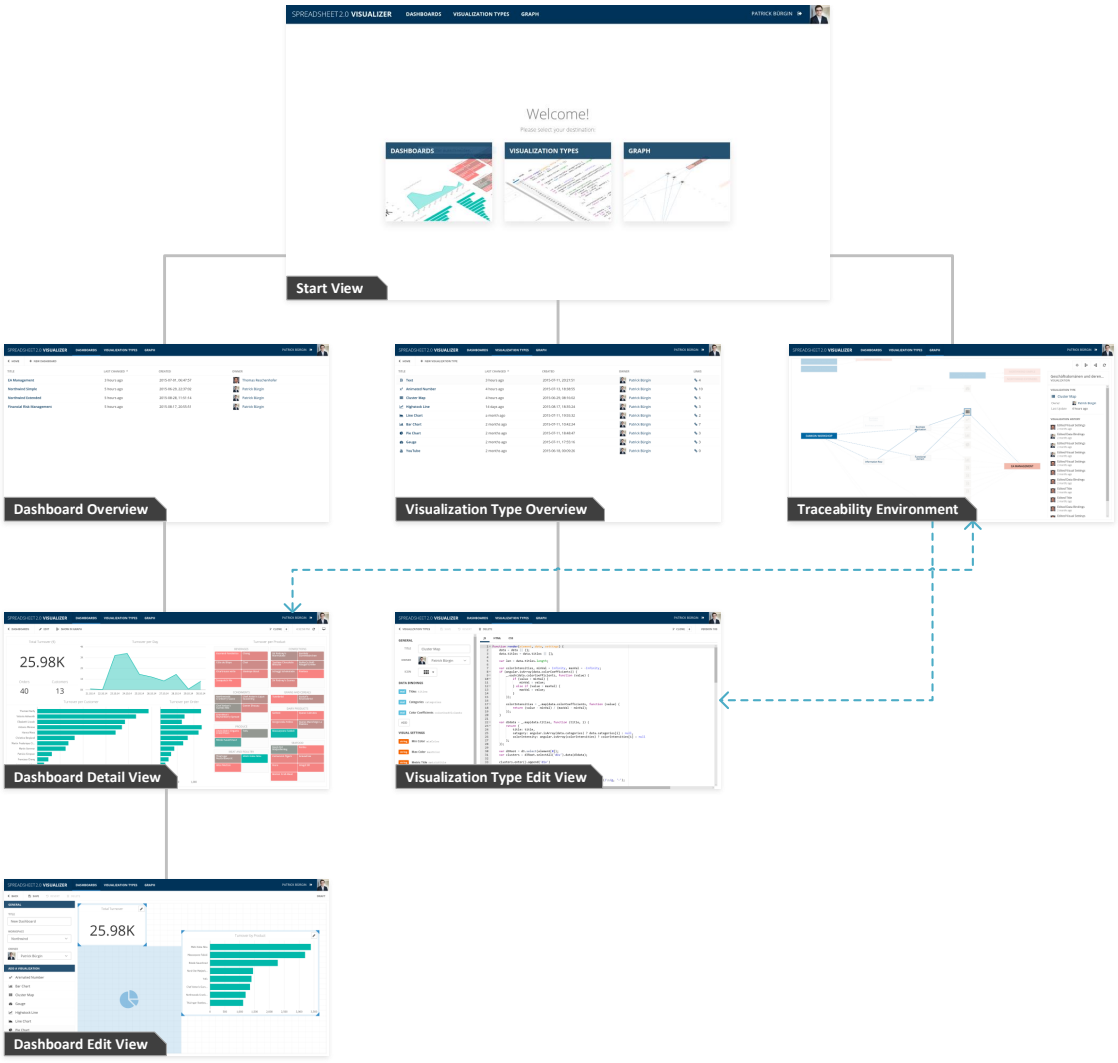


Figure 4.3: A simplified view hierarchy of the prototypical implementation.

4.1.5 Sample Data

The subsequent sections in this chapter refer to a demo data set called *Northwind*, which is represented as a *workspace* with *types* and *entities* in the prototype's underlying instance of SocioCortex (cf. section 2.3).

The setup is based on sample data for *Microsoft SQL Server*, and features sales data for a fictitious company, which imports and exports specialty foods from around the world.[11]

The resulting schema is depicted in Figure 4.4, and Figure 4.5 shows a selection of exemplary entities based on it.

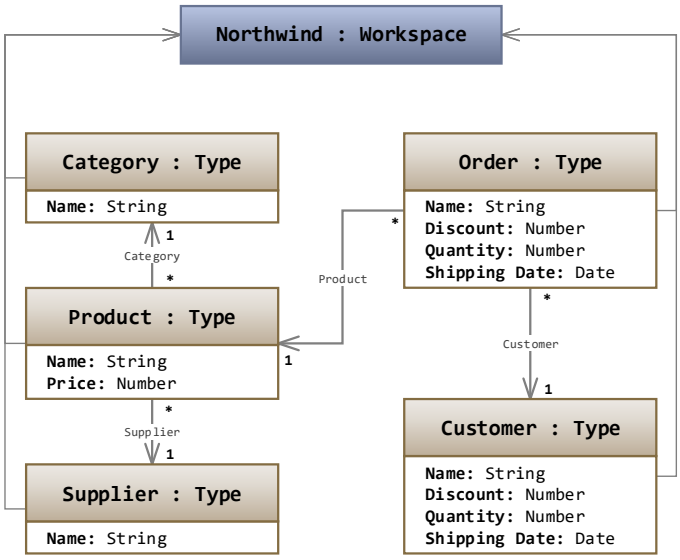


Figure 4.4: The basic schema of the sample data set.

Product	Price	Category	Supplier
Guaraná Fantástica	4.5	Beverages	Refrescos Americanas LTDA
Chang	19	Beverages	Exotic Liquids
Côte de Blaye	263.5	Beverages	Aux joyeux ecclésiastiques
Chai	18	Beverages	Exotic Liquids
Chartreuse verte	18	Beverages	Aux joyeux ecclésiastiques

Figure 4.5: A selection of exemplary *entities* based on the *type* `Product` introduced in Figure 4.4, as they are presented in the web interface of SocioCortex.

[11] Northwind Sample Data: `http://northwinddatabase.codeplex.com/`

Derived Attributes

SocioCortex allows users to extend a type's set of attributes by so-called *derived attributes*, whose values are generated at run-time based on MxL-expressions.

As depicted in Figure 4.6, the sample data set has been extended by a set of such attributes, e.g. `Total Price` in the case of the type `Order`, which multiplies the `Price` of its associated `Product` with the given `Quantity`, and subtracts the `Discount`.

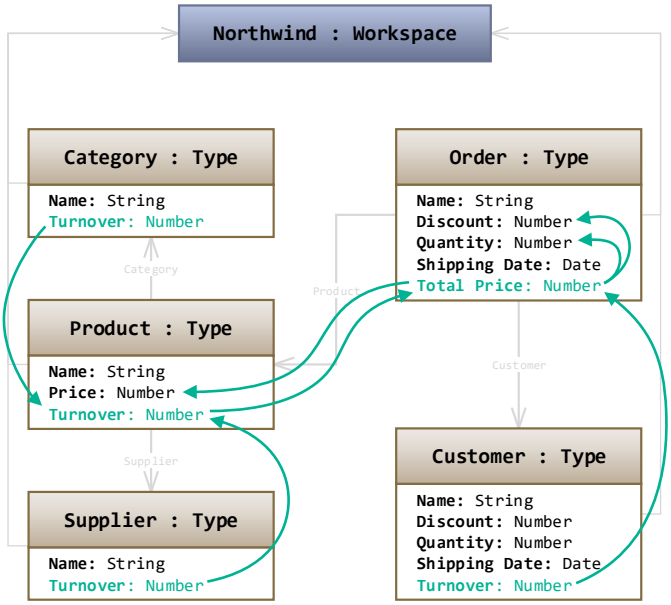


Figure 4.6: The extended schema of the sample data set, to be used in subsequent sections. The diagram highlights added *derived attributes* and their respective attribute dependencies.

4.2 Visualization Type Management

This section describes the prototypical implementation of the generic meta-model for the generation of reusable, web-based visualization types introduced in section 3.1.

The implemented `VisualizationType` model features three manually managed metadata elements, namely a title field, an icon name (based on *Font Awesome* icon toolkit[12]), and a reference to a user marked responsible for this item (labeled *Owner*). Furthermore, each `VisualizationType` is versioned, thus allowing for the inference of a creation date and a last changed date.

4.2.1 List View

When a user enters the visualization type management environment (labeled *Visualization Types*), she is presented with a table of all currently existing visualization types and their previously described metadata attributes. To support users at identifying *relevant* visualization types, and assessing their criticality, the table additionally lists the number of *used-by* references associated with each entry, as illustrated in Figure 4.7.
From this view, users may proceed to create a new visualization type, or edit an existing one.

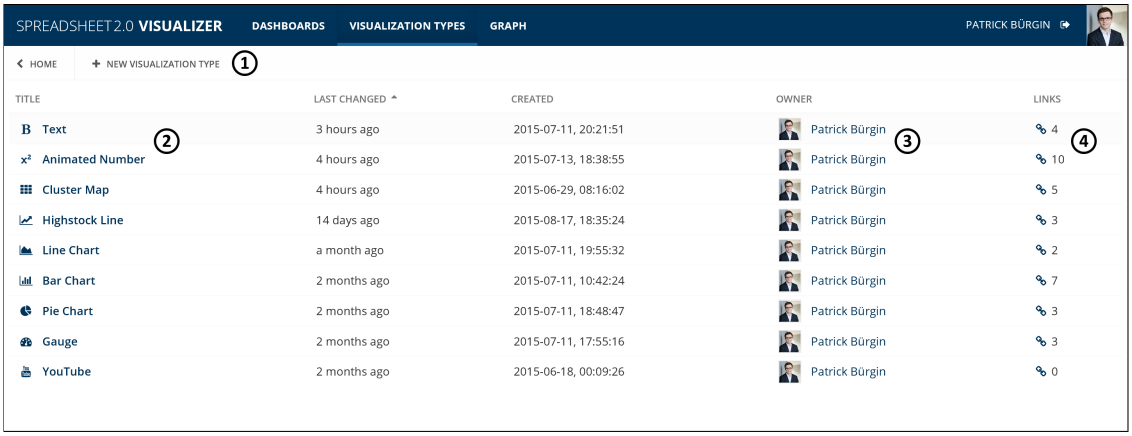


Figure 4.7: The overview page provides links to the *visualization type edit view* (1), existing visualization types (2), and profiles of the respective owners (3), as well as the number of *used-by* references (4), which serves as an indicator for popularity and criticality.

[12]Font Awesome: `http://fortawesome.github.io/Font-Awesome/`

4.2.2 Visualization Type Creation

As introduced in section 3.1.2, a visualization type can be modeled as a composition of descriptive metadata elements, data parameters, visual settings, and a visualization logic. This model is reflected in the view for creating new visualization types presented in Figure 4.8.

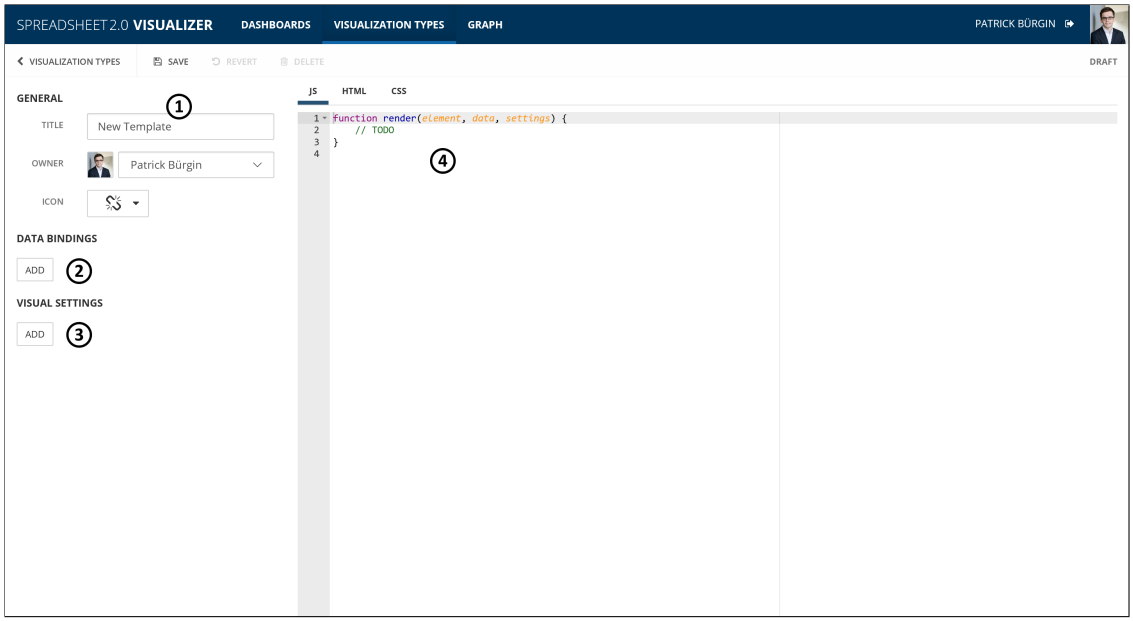


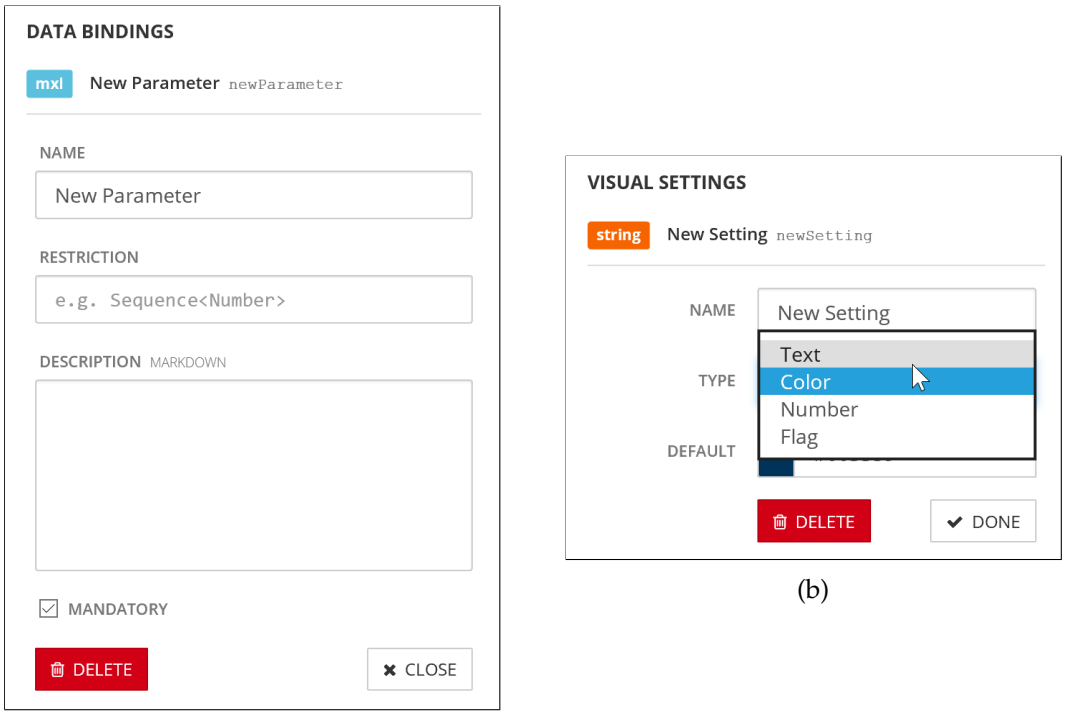
Figure 4.8: The *visualization type edit view*, as presented when a user creates a new visualization type.

Metadata Elements

In the *General* section (1) of the visualization type edit view, users may specify a title, select an owner from a list of users registered on the underlying instance of SocioCortex, and select an icon.

Specifying Data Parameters

The data parameters section (labeled *Data Bindings*) (2) enables users to create, update, and delete instances of `DataParameter` to specify the information demands of the visualization type. Figure 4.9a provides a more detailed example, showcasing the different form elements. Each parameter has a *Name* field, which is automatically transformed in to a *lower camel case* string to be used when defining the visualization logic, transforming e.g. *New Parameter* to `newParameter`. Furthermore, the interface allows users to specify an expected type, based on MxL's type hierarchy as introduced in section 2.3.2, as well as a description text and an *is mandatory* flag.



(a) (b)

Figure 4.9: The view components for configuring the `DataParameters` (a) and `VisualSettings` (b) of a given `VisualizationType`.

Specifying Visual Settings

In the *Visual Settings* section (3), users may add, modify, and remove graphic variablility points of a visualization type. As showin in Figure 4.9b, users may select from four different setting types, namely *Text*, *Color*, *Number*, and *Flag*, which correspond to the different types of `VisualSetting` introduced in the model section. As with data parameters, each setting has a *Name* field, which is automatically transformed in to a *lower camel case* string to be used when defining the visualization logic.

Adding Visualization Logic

As discussed in the model section, the visualization logic is the essential part of a visualization type, defining the *visualization transformations* and *visual mapping transformations* that transform data into views.
The prototype allows users to specify visualization logic in plain HTML, CSS, and JavaScript, in line with the previously introduced `BasicVisualizable`. To this end, the view features a set of code editors (4).

4.3 Visualization Environment

This section presents the prototypical implementation of the concepts for creating *visualizations* by binding *visualization types* to data, and composing *dashboards*, introduced in section 3.2.

4.3.1 List View

As with visualization types, a user entering the visualization environment (labeled *Dashboards*) is presented with a table of all existing dashboards and their complementary metadata attributes, namely the creation date, last changed date, and information about the owner, as shown in Figure 4.10.

From this view, a user may proceed to open an existing dashboard, or create a new one.

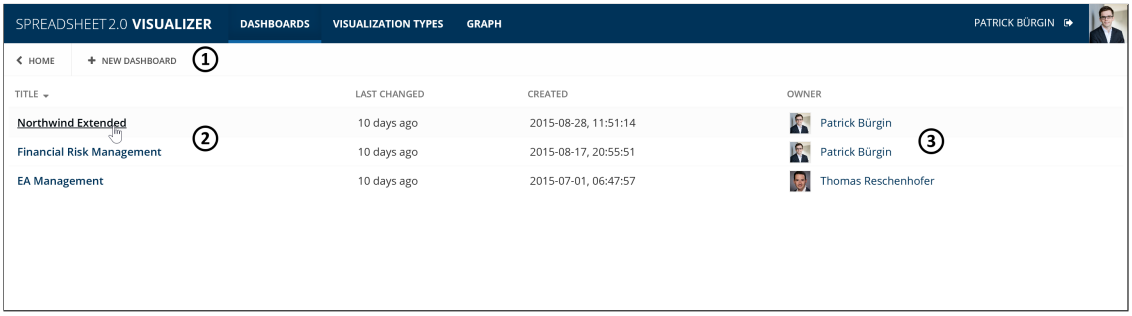


Figure 4.10: The overview page provides links to the *dashboard edit view* (1), existing dashboards (2), and profiles of the respective owners (3).

4.3.2 Creating Dashboards

Responsive Grid

According to Few's definition, a dashboard should not exceed a single screen (cf. section 3.2.1). In order to support flexible moving and resizing of visualizations regardless of screen size and display resolution, the visualization environment implements the grid system sketched in section 3.2.3, equipping its dashboards with a 36×28-grid.

The corresponding view component allows users to move and resize items on the grid in a non-overlapping fashion, as signified by drag handles and special cursor icons on hover—Figure 4.11 provides an illustration of the corresponding interactions.
The view component has been isolated from the prototype, and made available on the open source platform *GitHub* under the MIT license.[13]

[13] angular-widget-grid on GitHub: `https://github.com/patbuergin/angular-widget-grid`

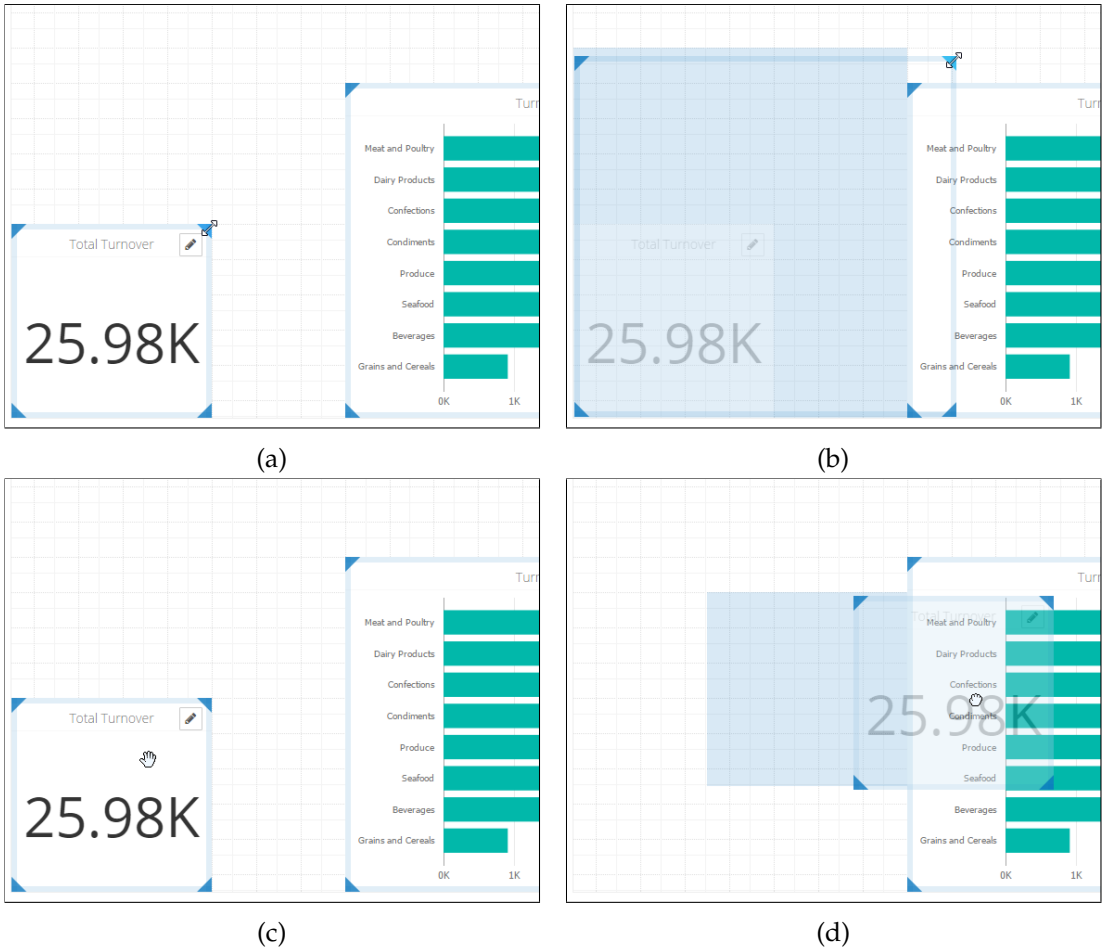


Figure 4.11: When moving and resizing items on the grid, the user is presented with suitable signifiers and immediate feedback for her actions, shortening the gulfs of execution and evaluation.

Creating a New Dashboard

When creating a new dashboard, the user is directed to the *dashboard edit view*, as shown in Figure 4.12. There she is presented with affordances to configure metadata elements (1), select from the palette of visualization types (2), and an empty, grid-based canvas (3). As with visualization types, dashboards are versioned, starting in a *draft mode* (4).

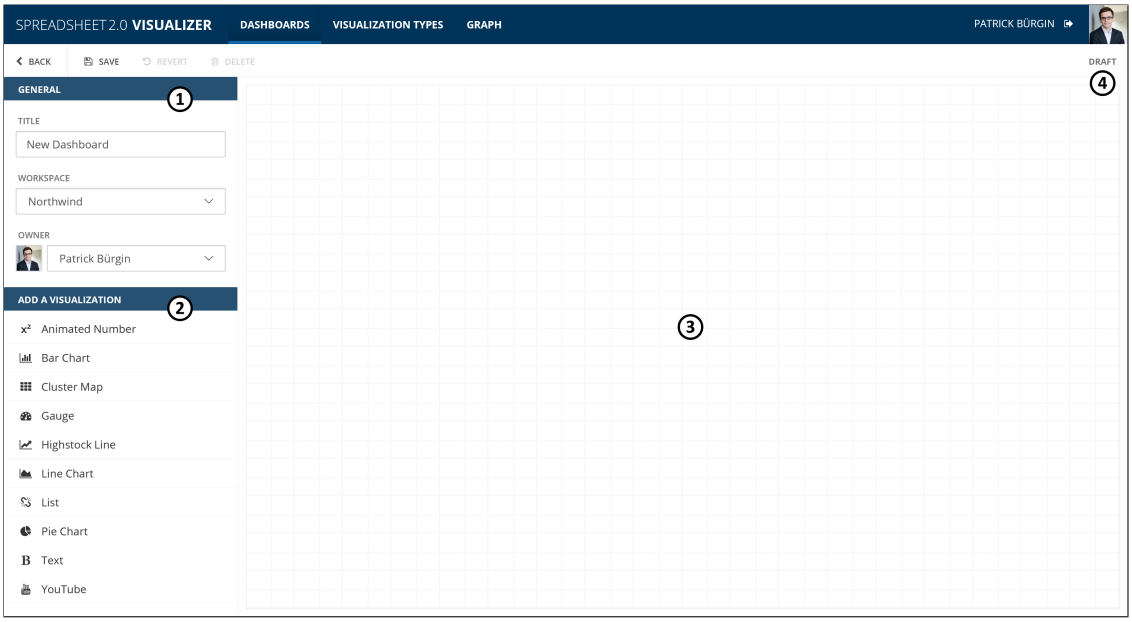


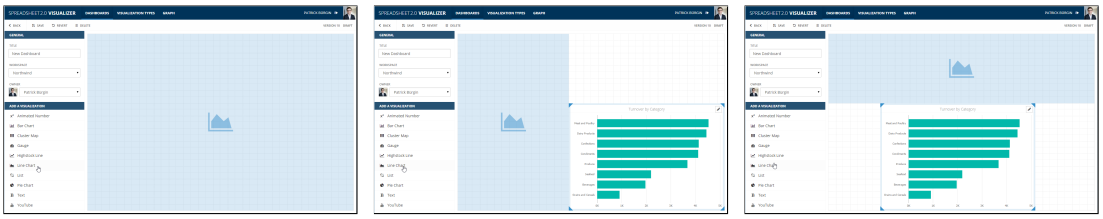
Figure 4.12: The *dashboard edit view*, as presented when a user creates a new dashboard.

Adding Visualizations to the Grid

When a user selects a visualization type from the palette, the system creates a new instance of `Visualization`, adds a reference to the respective `VisualizationType`, and places it at the largest available, rectangular area in the grid—the respective controls are disabled when there is no space left.
To make this mechanism more apparent, the system shortens the gulf of execution by providing immediate feedback when a user hovers an item of the palette, highlighting the area that will be assigned to the visualization on selection, as illustrated in Figure 4.13.

Configuring Visualizations

After creating a new visualization, or selecting the edit button of an existing one, the system opens a *visualization configuration panel*, as illustrated in Figure 4.14. The panel allows users to configure metadata elements (1), data parameter bindings (2), and visual setting bindings (not visible; shown in Figure 4.17b) of the selected instance of



(a) (b) (c)

Figure 4.13: When hovering items of the visualization type palette in the sidebar, the user is presented with a preview of the area where the respective visualization will be added when she confirms the selection.

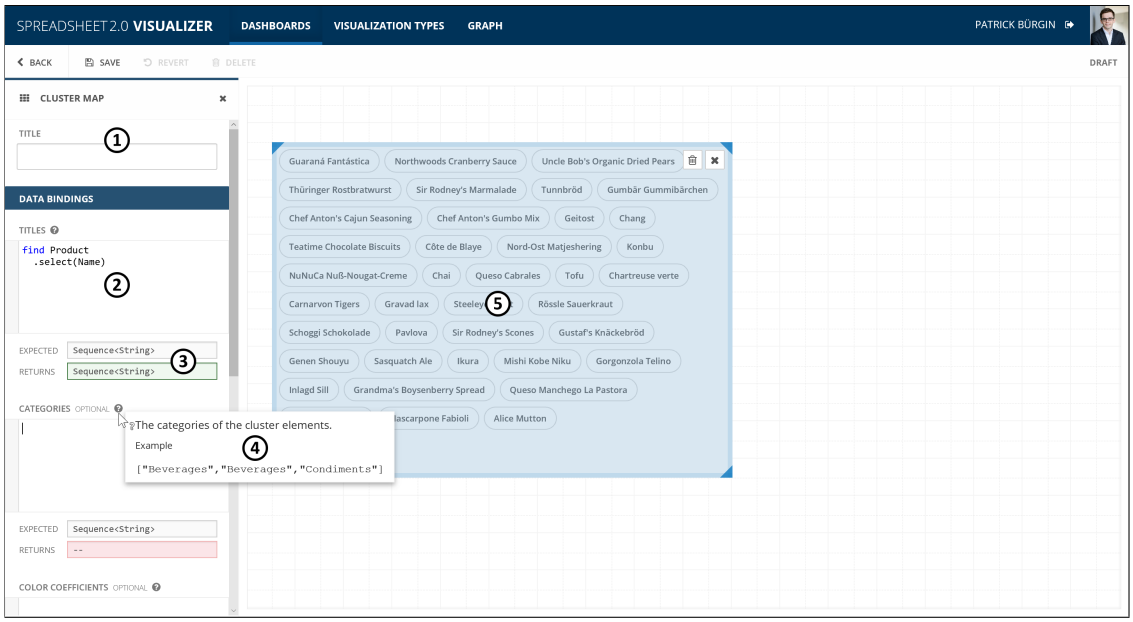


Figure 4.14: The visualization configuration panel.

`Visualization` (5). Changes to any of these properties are reflected back immediately, causing the visualization to re-render, thus providing the user with feedback to shorten the gulf of evaluation.

As modeled in section 3.2.2, *visualizations* bind *visualization types* to data by addressing their information demands, as expressed by the expected types of their *data parameters*, with corresponding *data parameter bindings* and queries.
As mentioned, the prototype uses an instance of SocioCortex as its data source, and employs the *model-based expression language* (MxL, cf. section 2.3.2) for the definition of queries. Marker (2) shows an exemplary MxL statement entered into the view component for configuring data parameter bindings, selecting the *Name* attributes of all data entities of type *Products*.

To support users at defining queries, the system embeds the in-browser code editor introduced in section 2.3.2, which provides e.g. syntax highlighting and code completion, as illustrated in Figure 4.15. To aid users in meeting the demands of the visualization type, the system validates the entered queries and matches their current return types with the expected ones (3), and provides access to the description texts defined in the visualization type (4), if any.

To further illustrate the discussed concepts and view components, Figures 4.16 and 4.17 illustrate sets of data parameters resp. visual settings and matching bindings next to each other.

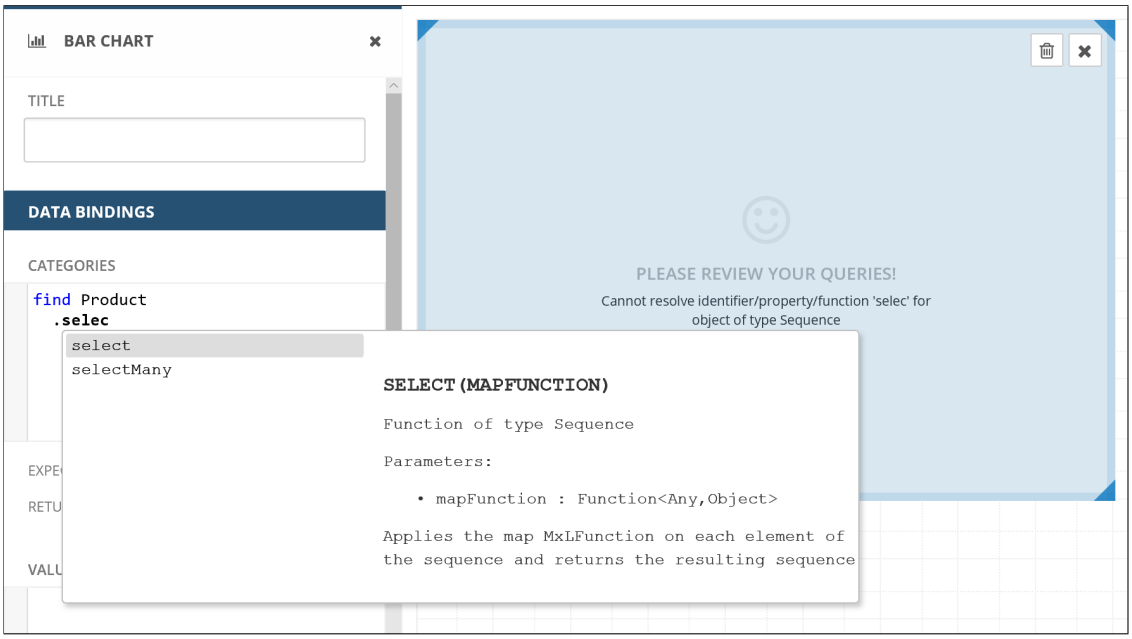
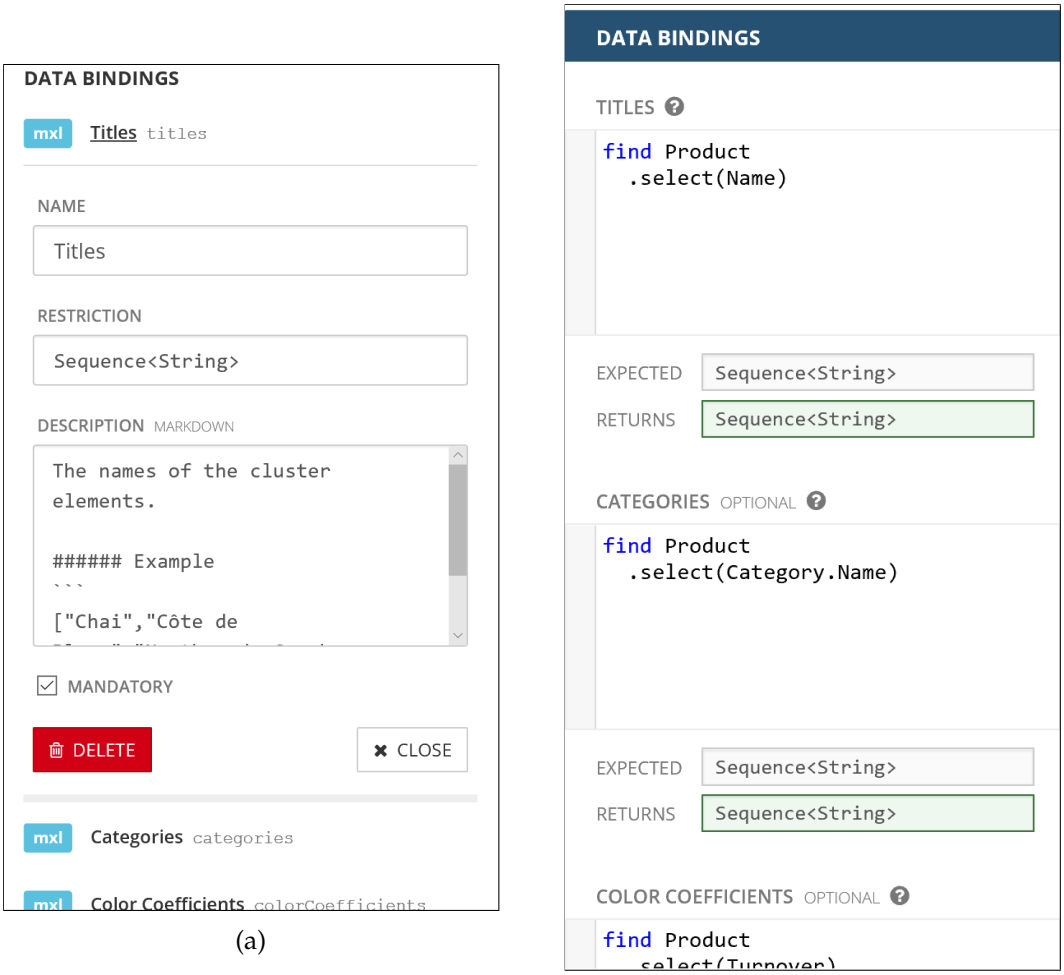


Figure 4.15: To aid users in configuring data parameter bindings with *MxL*, the prototype embeds a suitable code editor, providing syntax highlighting and code completion. Changes to the queries trigger a re-rendering of the selected visualization, thus helping users figure out the results of their actions.



(a) (b)

Figure 4.16: An exemplary set of `DataParameters` (a), and a corresponding set of `DataParameterBindings` (b), as they are represented in the prototypical implementation.

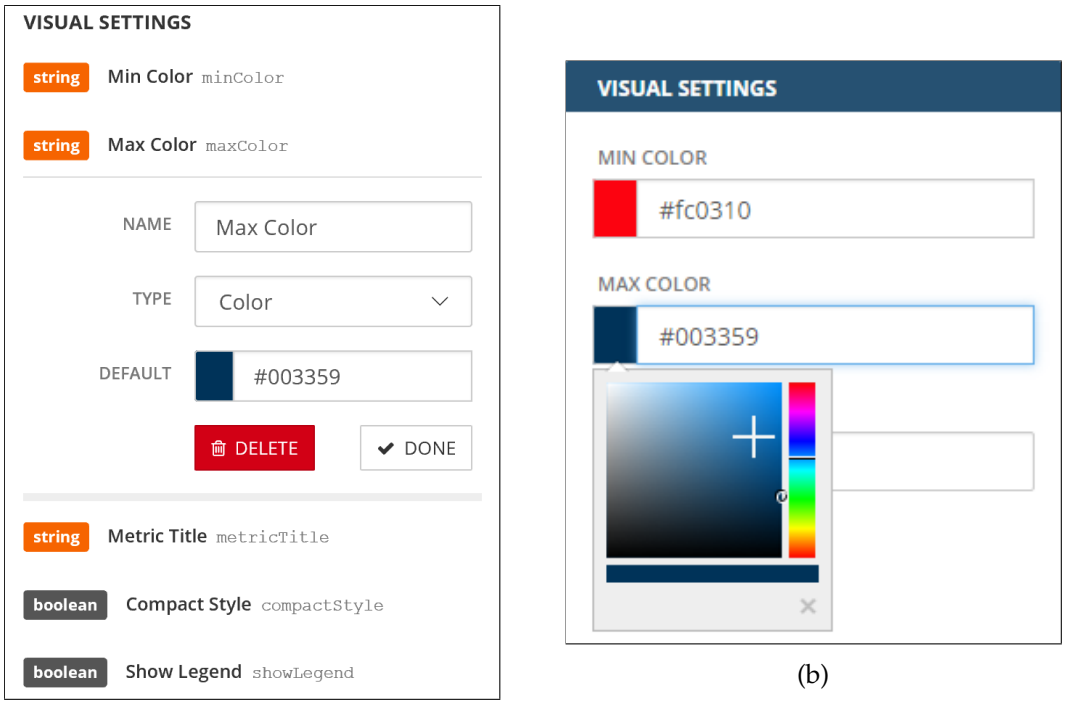


Figure 4.17: An exemplary set of `VisualSettings` (a), and a corresponding set of `VisualSettingBindings` (b), as they are represented in the prototypical implementation.

4.3.3 Viewing Dashboards

When opening an existing dashboard, e.g. via the list view introduced in section 4.3.1, the user is directed to the *dashboard detail view*. Figure 4.18 shows an exemplary dashboard called *Northwind Extended*, which is based on the *Northwind* data set introduced in section 4.1.5. For reference, Figure 4.19 provides a screenshot of the same dashboard in the *dashboard edit view*.

The dashboard makes use of four different visualization types: *Animated Number*, *Bar Chart*, *Line Chart*, and *Cluster Map*. The first three are adapters to *D3.js*-based visuals provided by the team behind *Power BI*[14], a BI tool introduced in section 2.1.3; the fourth one will be discussed in section 5.2.

From the detail view, a user may proceed to edit the dashboard, examine it in the traceability environment which will be introduced in the next section (labeled *Show in Graph*) (1), create and edit a copy (2), or enter full-screen mode (3):

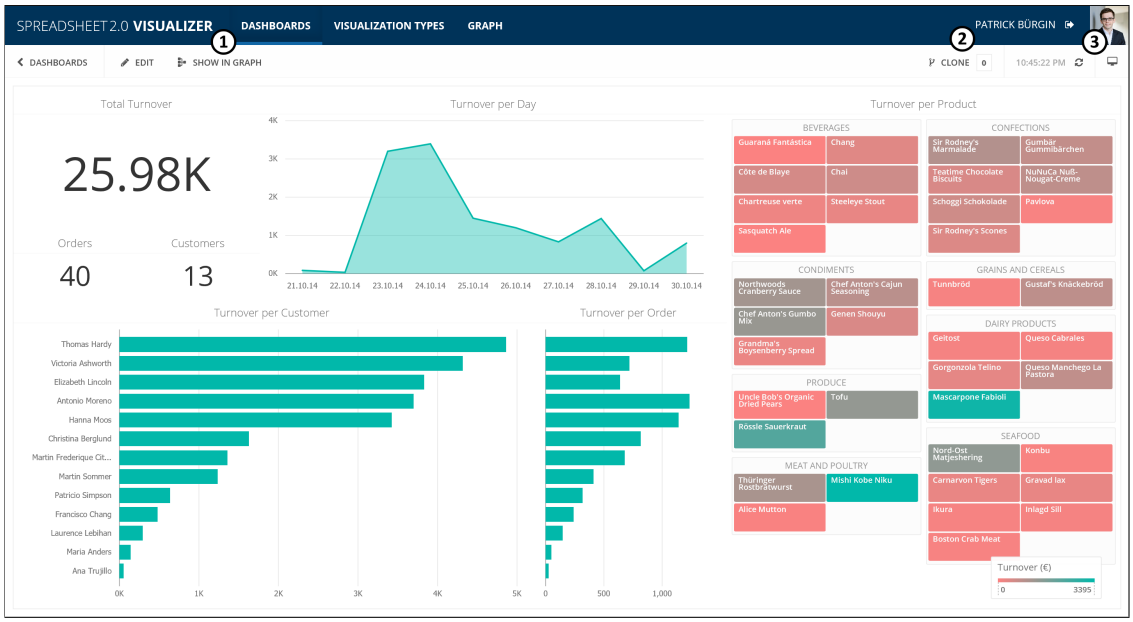


Figure 4.18: An exemplary dashboard based on the data set introduced in section 4.1.5, presented in the *dashboard detail view*.

[14]Power BI Visuals on Github: `https://github.com/Microsoft/PowerBI-visuals`

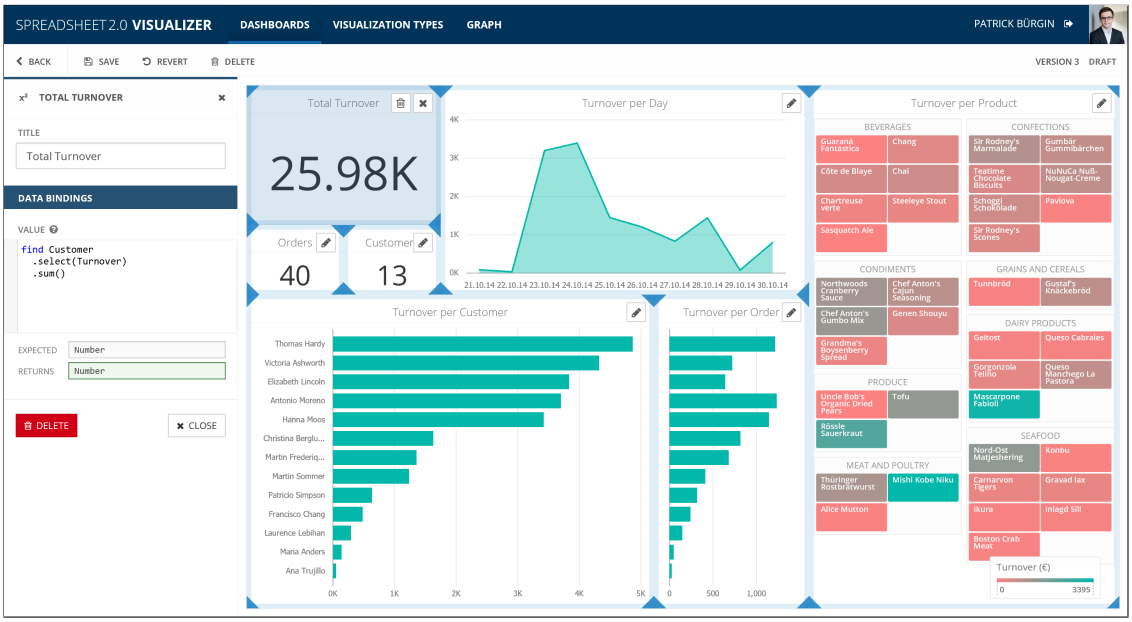


Figure 4.19: The exemplary dashboard *Northwind Extended* from Figure 4.18, shown in the *dashboard edit view*. The selected visualization named *Total Turnover* is of type *Animated Number*, and references the derived attribute `Turnover` of `Customer` as introduced in section 4.1.5.

4.4 Traceability Environment

This section presents a first implementation of a traceability environment for supporting end-users in understanding and analyzing the networks spanned by their organization's data assets, visualizations and people, as introduced in section 3.3.

The prototype enables users to visually explore a subset of the network model introduced in section 3.3.1, featuring visualization types, visualizations, and dashboards, as well as data sources (⇒ workspaces), types, and people (⇒ registered users) from the underlying instance of SocioCortex.

4.4.1 Network Generation

The network that drives the traceability environment is generated on-demand and in the client, causing a cascade of back end calls whenever a user first enters the view, or selects its *refresh* button. Albeit suffering from obvious scalability issues, this approach worked sufficiently fast in the small-scaled test environment, and ensures networks that accurately represent the status quo.

The network generation process is based on the first two steps of the three-step procedure proposed in section 3.3.1, and yields the model depicted in Figure 4.20:

1. Fetch all instances of `Dashboard`, `VisualizationType`, and `Visualization`.
2. Fetch all instances of `Workspace` and `Type`.
3. Drop the workspaces and types used to persist items from the prototype.
4. Add a vertex for each remaining instance of workspace, type, and dashboard, and attach information from available metadata elements.
5. Add a vertex for each instance of visualization, and attach available information from both the visualization and its referenced visualization type.
6. Add directed edges from workspaces to their associated types, and from visualizations to their associated dashboards.
7. Statically analyze the queries of each visualization, and add directed *referenced-in* edges to reflect the network of references which they span.

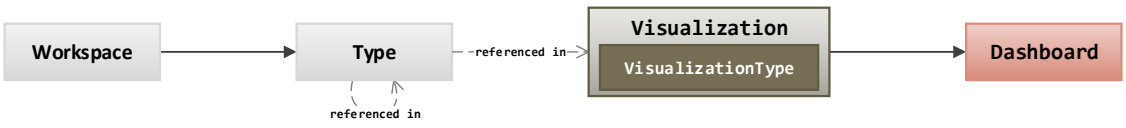


Figure 4.20: The network model used within the prototypical implementation. As SocioCortex' types can be enriched with *derived attributes* (cf. section 4.1.5), types may reference other types, including themselves.

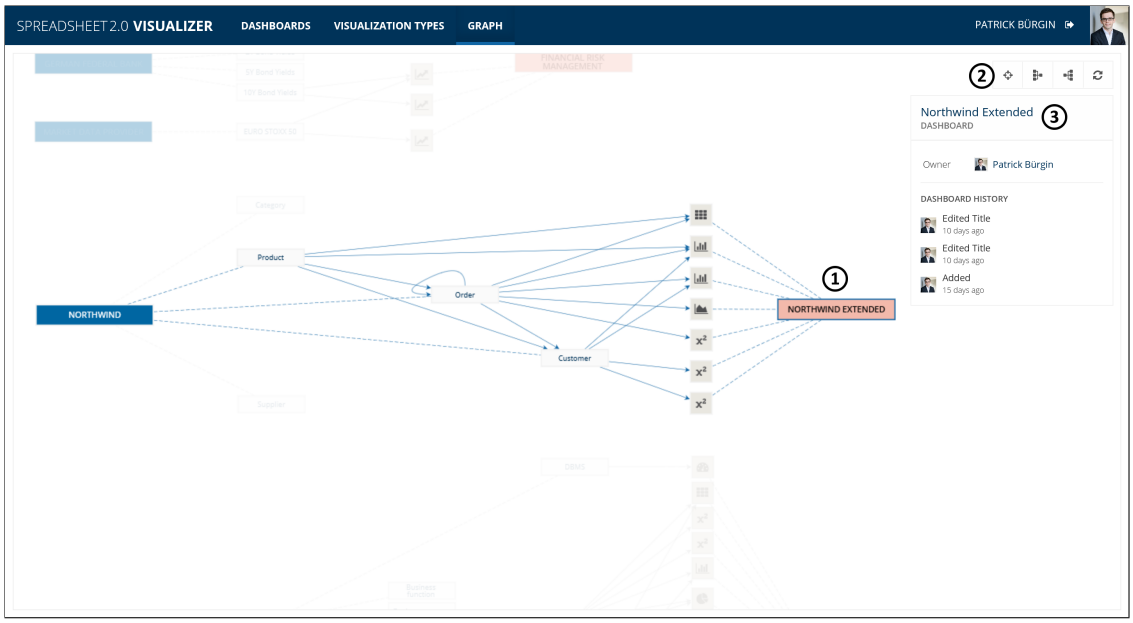


Figure 4.21: The *traceability environment*, as presented when a user selects *Show in Graph* in the *dashboard detail view* (cf. section 4.3.3). In this case, the system preselects the associated dashboard-vertex (1), highlighting all associated vertices, and providing more details in a side panel (3).

The button bar (2) provides affordances to reset the view, collapse/expand all eligible vertices, and refresh the network.

4.4.2 Visual Exploration

The *traceability environment* (labeled *Graph* in the user interface) can be accessed from the *dashboard detail view*, and via the navigation bar. Upon entering the view, the client performs the previously described network generation process, computes a layout with the help of *Dagre*[15], whose core is based on Gansner et al.'s *Technique for Drawing Directed Graphs* [12], and presents the user with a visual exploration environment.

The view consists of two major parts:

1. An *interactive node-link diagram* based on *Cytoscape.js*[16]), allowing users to pan and zoom the canvas, and to drag and select vertices.

2. A *modular side panel*, providing informations about the currently selected node.

Figure 4.21 shows the initial state of the environment after selecting *Show in Graph* in the *dashboard detail view*—in this case for the dashboard *Northwind Extended* presented in section 4.3.3.

[15]Dagre: `https://github.com/cpettitt/dagre/wiki`
[16]Cytoscape.js: `http://js.cytoscape.org/`

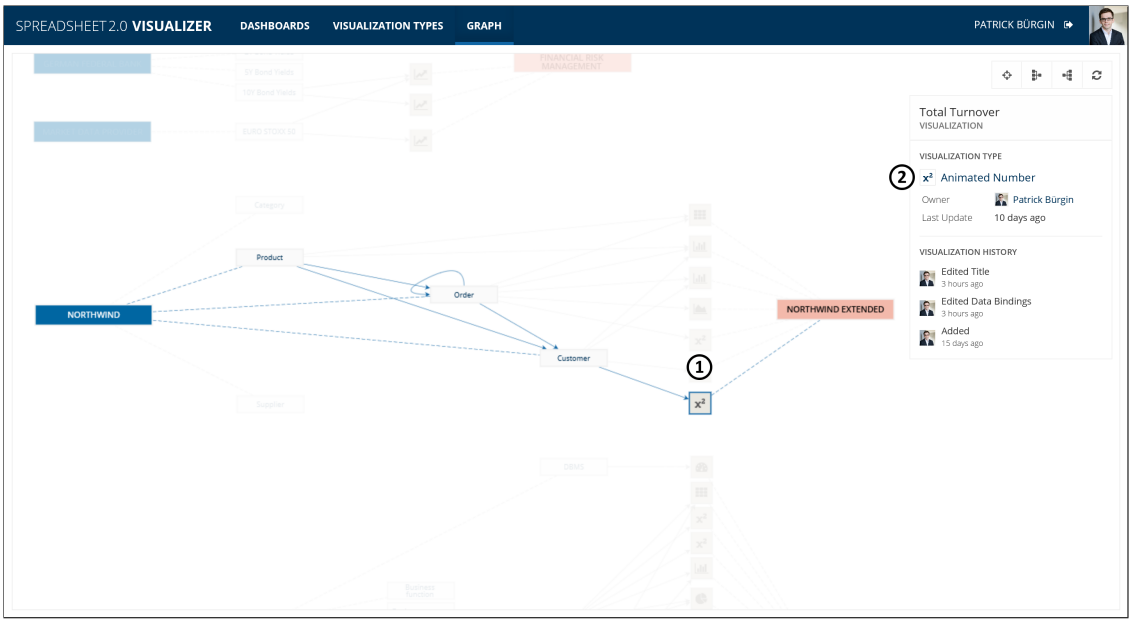


Figure 4.22: The same scene as in Figure 4.21, but with a different vertex selection. In this case, the visualization *Total Turnover* from Figure 4.19 has been selected (1), revealing e.g. its reference to `Customer`, its associated visualization type (2), and information about its change history.

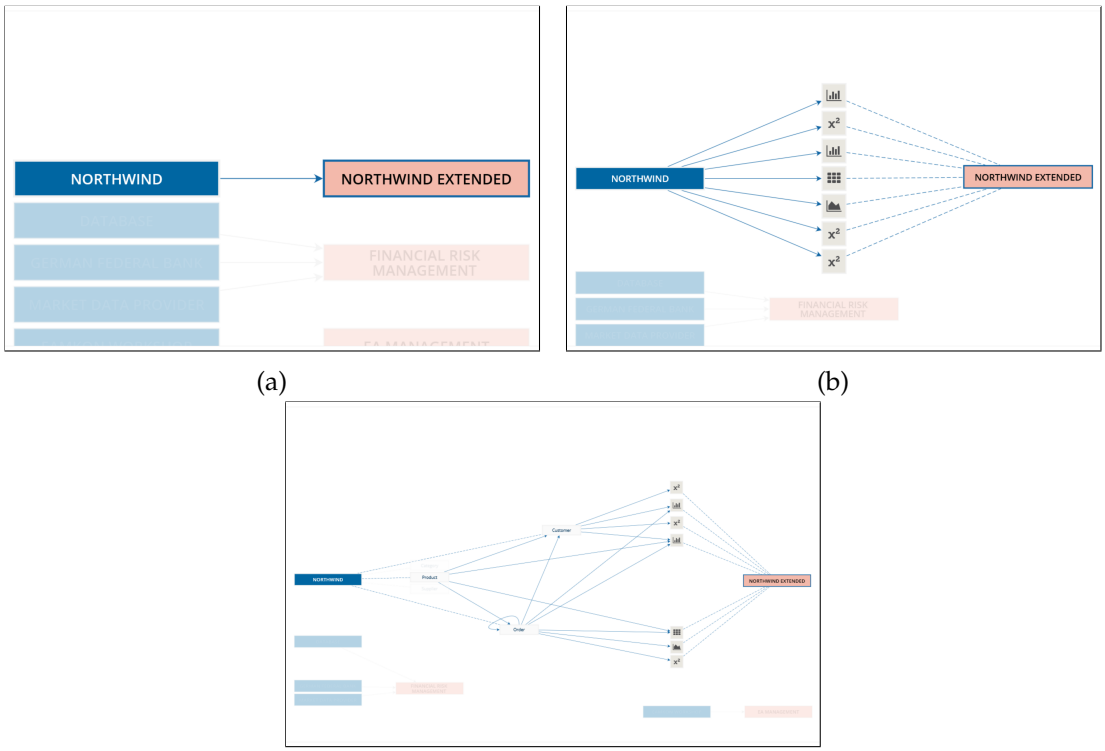
Features to Support Exploration

To facilitate the exploration of the network, the environment offers a selection of supportive features, including:

Highlighing Relevant Items: When a user selects a vertex, the environment highlights all edges on paths from and to the item, and fades vertices that cannot be reached, thus helping users to identify the items that depend on the selected one, as well as items that the selected one depends on. By presenting two different selection states, Figures 4.21 and 4.22 may serve as an illustration for this behavior.

Drill-Down: By double-clicking resp. tapping an eligible item, users can drill down on a vertex, reveal more detailed relations, as presented in Figure 4.23.

Fit Selection: At any time, users may select the *Reset View*-button in the top right corner, or hit the space bar, to fit pan and zoom to the current selection—be it a selected vertex and its in-/out-components, or the entire network.



(a) (b)

(c)

Figure 4.23: As envisioned in Figure 3.16, the traceability environment allows users to drill-down on data sources and dashboards. By double-clicking resp. tapping an eligible item, users can reveal more detailed relations, allowing them to go from (a) to (c).

5 Evaluation

Following the design and prototypical implementation of a platform for traceable data visualization from chapters 3 and 4, this chapter aims to shed light on the viability of the discussed concepts, and play a formative role for its further development.
Section 5.1 describes the methodology of the evaluation: an exploratory case study with a "two-case" design. Sections 5.2 and 5.3 evaluate and discuss the concepts from the perspectives of two different industry domains: enterprise architecture management and financial services. Finally, section 5.4 provides a cross-case synthesis.

5.1 Methodology

In their information systems research framework, Hevner et al. describe design as a "search process to discover an effective solution to a problem"—an iterative generate/test cycle that alternates between *Develop/Build* and *Justify/Evaluate* phases [17].

With this in mind, first assessments of the design artifacts have been performed within the research group, taking the form of informal interviews, as well as monitored usage of the prototypical implementation. These preliminary evaluation steps yielded valuable feedback, especially with respect to the prototype's user interface, resulting in refinements such as adding dialogs to prevent accidental discarding of unsaved changes.

In a first effort to get feedback from business professionals, we decided to perform an exploratory case study evaluation with a "two–case" design, following Yin [36, Appendix B]. This evaluation stage is driven by the following set of goals:

1. Lay the foundation for refinement and reassessment in future work.
2. Assess the validity of the developed artifacts with respect to practitioner demands.
3. Assess the utility and usability of the prototypical implementation.

Case Study Design

The case study is based on the assumption that organizational needs with respect to custom visualizations and network analysis differ across data domains and organizations. To play a formative role for the further development of the artifacts, the evaluation employs a replication logic, and assesses the developed artifacts in the light of organizations in different data domains, namely *enterprise architecture management*, and *financial services*.

At the current state of development, an integration of the prototypical implementation within the technical infrastructure of third parties is not yet feasible. To this end, the individual cases are guided by domain-themed scenarios constructed around the artifacts, and employ expert interviews as their sole source of evidence. Consequently the cases may suffer from weaknesses such as a bias due to poorly articulated questions, response bias, and reflexivity [36, p. 106].

Expert Interview Design

The interviews were held in a prolonged fashion (about two hours each), and based on the qualitative style described by Rubin and Rubin, which revolves around employing open-ended queries, and following up on any arising new idea or perspective to examine its relevance for the study [30].

Albeit responsive, the interviews both followed the same multi-tier structure, designed to reduce response bias by discussing underlying issues before introducing the corresponding design artifact:

1. Introduction of myself and the goals of the interview, without yet disclosing details about the designed artifacts.
2. Easy, general questions, addressing e.g. the role of data visualization and collaborative usage of data at the interviewee's organization.
3. Introduction of the visualization type concept discussed in section 3.1, accompanied by questions addressing custom visualization demands.
4. Introduction of the general concepts driving the traceability environment, as discussed in section 3.3, accompanied by questions addressing possible benefits by analyzing different types of relations in the network spanned by data assets, visualizations and people.
5. Introduction and discussion of the prototypical implementation and its three major concepts, namely dashboards, the traceability environment, and visualization types.

5.2 Case 1: Enterprise Architecture Management

The first case evaluates the developed artifacts in an *enterprise architecture management* (EAM) setting, where EAM describes "a continuous and iterative process controlling and improving the existing and planned IT support for organizations" [8].

The interview partner for this case is an *Enterprise Architect* with over nine years of professional experience in the role, currently managing the internal architecture of an IT services provider (5000-10000 employees).

5.2.1 Scenario Design

Exemplary Visualization Type

According to survey results from the *Enterprise Architecture Visualization Tool Survey 2014*, *cluster maps* are one of the five most commonly used visualization types among EA professionals [29, p. 371]. As described by Ernst et al., a cluster map is characterized by its use of nesting to visualize a relationship between cluster elements and nested elements—e.g. of type *used-in* or *belongs-to* [9].

Figure 5.1 shows an exemplary instance of the `VisualizationType` *Cluster Map*, which has been implemented in the prototypical implementation to serve the EAM case. The implemented type has three data parameters, namely *Titles*, *Categories*, and *Color Coefficients*, and a set of visual settings to control visual aspects, such as the padding of clusters.

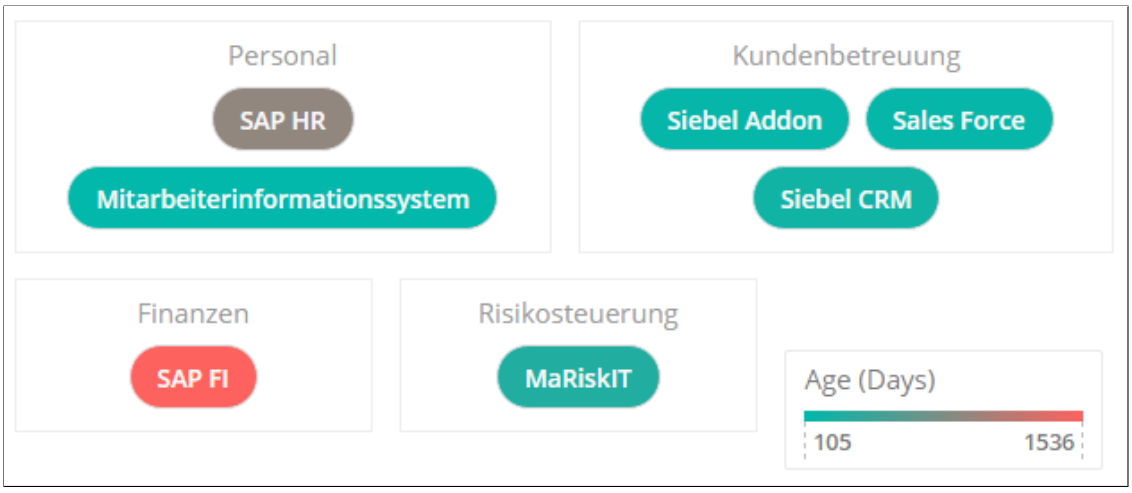


Figure 5.1: An instantiation of the exemplary visualization type for the *Enterprise Architecture Management* case: *Cluster Map*. The visualization illustrates different business applications deployed in an organization, grouped by the functional domain which they belong to, and colored by their age.

Dashboard Design

The EAM-based scenario is based on a previously created demo data set called *EAMKON Workshop*, which is represented as a *workspace* in the prototype's underlying instance of SocioCortex. Figure 5.2 shows an illustration of the types contained in the workspace:

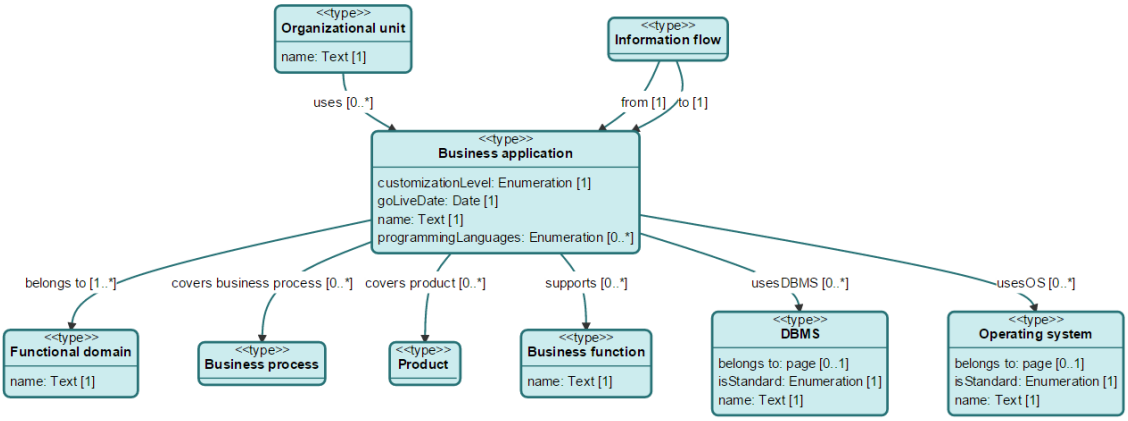


Figure 5.2: The set of types contained in *EAMKON Workshop* workspace, which drives the dashboard shown in Figure 5.3. The diagram has been generated by the web interface of SocioCortex.

As seen in the Figure, the schema is centered around a type called `Business Application`, modeling i.a. the information flow between applications with the help of a support type (`Information Flow`), and connecting applications to the `Functional Domain` which they belong to.

With the help of the model-based expression language (MxL, cf. section 2.3.2), this schema allows e.g. for the following queries, which are used to bind data to the cluster map depicted in Figure 5.2:

```
find 'Business Application'
  .select((Today - goLiveDate))
/* Returns the age of each instance of Business Application */

find 'Business Application'
  .select('belongs to'[0].name)
/* Returns the first Functional Domain associated with each Business
    Application */
```

Building on this, Figure 5.3 presents the dashboard created to support the case, providing users with a set of visualizations that describe the current state of a fictional companies' application landscape—Figure 5.4 presents the corresponding view in the *traceability environment* (via *Show in Graph*, cf. section 4.3.3).

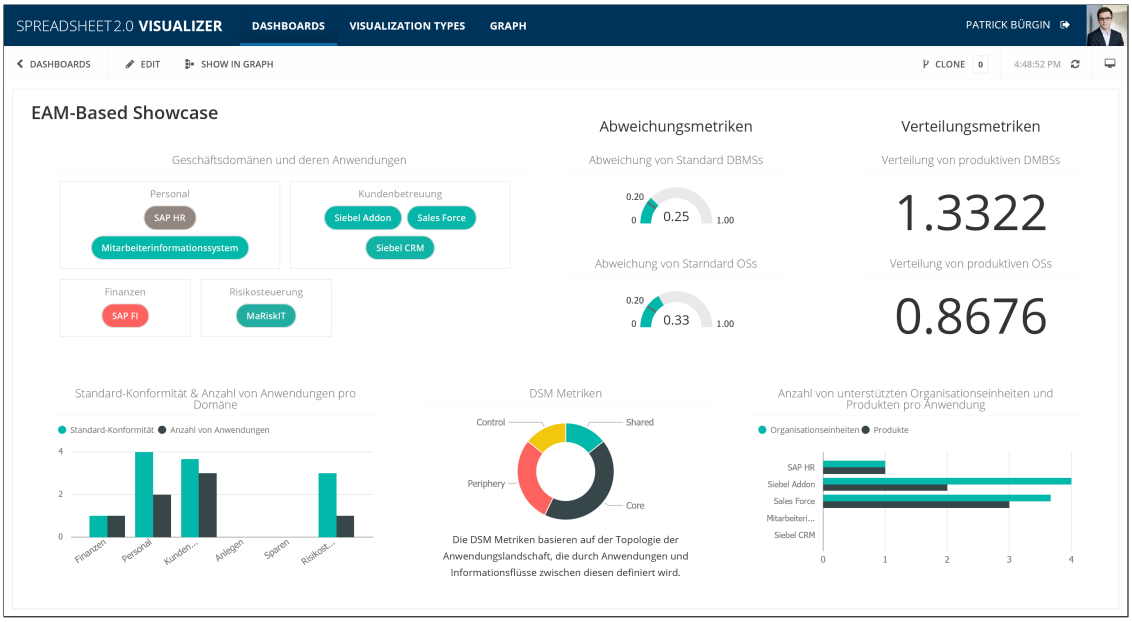


Figure 5.3: The dashboard built to support the *Enterprise Architecture Management* scenario, presented in the *dashboard detail view* (cf. section 4.3.3).

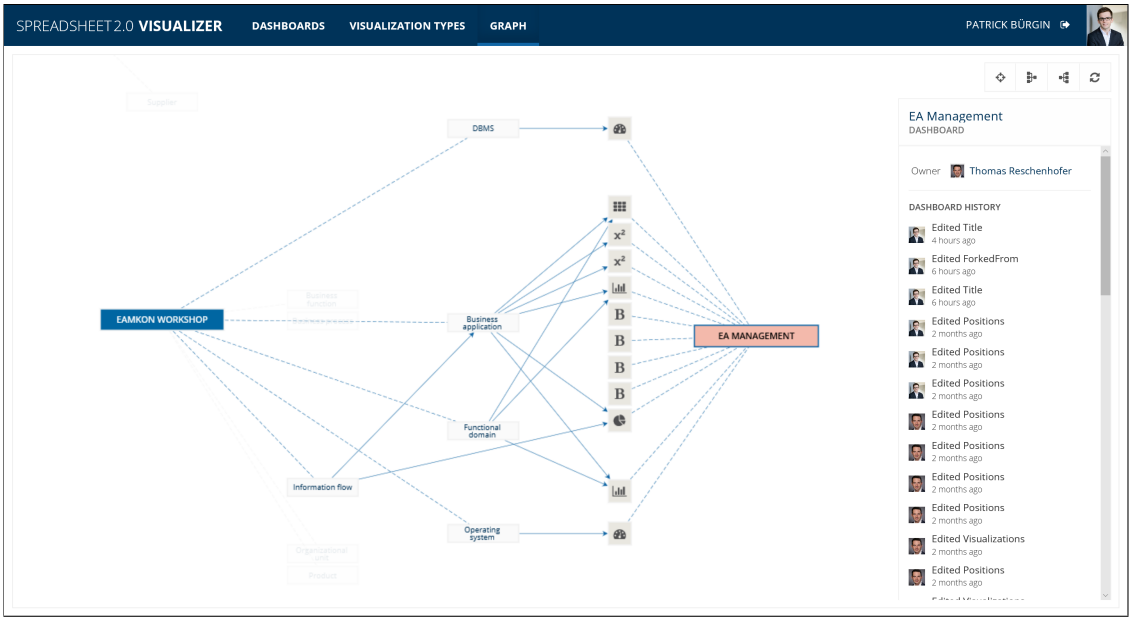


Figure 5.4: The *Enterprise Architecture Management* dashboard, as shown in Figure 5.3, presented in the *traceability environment* (cf. section 4.4).

5.2.2 Interview Report

Data Sources

In the environment of the interviewee, the main data sources are manual assessments, and a configuration management database (CMDB) maintained by third parties.
The manually gathered and/or imported data is managed within a special purpose, model-based EA system, employing a conceptually similar but significantly more complex schema than the one depicted in Figure 5.2.

Visualization Types

Apart from using common EA visualizations such as the cluster map introduced before, the interviewee stated that he has developed an interactive custom visualization based on *D3.js* (cf. section 2.1.3), which allows users to browse the network spanned by different entities of the model. On a related note, he stated that he participates in an exchange program for EA-related custom visualizations which connects EA professional across different companies.

Collaboration

In the EAM environment, collaboration with respect to data arises from different sources. First off, according to the interviewee, a key role of an enterprise architect is to serve as a link between business and IT, gathering, updating and integrating data from both sides into the model-based EA system.
This data is then used to support decision making, i. a. driven by information visualization or the calculation of EA metrics. These measures and metrics are then shared with different stakeholders on the quest to drive architectural change—a core use case being the use of visualizations as a basis for discussion in workshops.

Use Cases for the Traceability Environment

Usage Analysis: Looking at the network showing the flow from data sources to dashboards, as depicted in Figure 5.4, the interviewee stated that he "wished it would go one step further: to the users". Stating that "the collection of data induces the highest cost in our daily business", he argued that "the more I can learn about who uses certain visualizations [etc], the more I can infer about the importance of model-elements", thus enabling architects to optimize models, e.g. by stopping to maintain never/rarely referred attributes.

Stakeholder Identification: The interviewee stated that the traceability environment could be used by architects to identify stakeholders, enabling them to i. a. proactively contact the users that expressed their interest in a given dashboard (e.g. via following/subscribing), ask them "whether they want to participate", and "discuss possible changes".

Impact Analysis: The architect believes that less tech-savvy users are deterred by the "model-based nature" of EA system deployed in the organization, causing them to rather ask him by mail to adapt elements of the model, than to do it themselves, as they are afraid to "break something". Faced with the traceability environment, the interviewee expressed that "this would certainly help to make the models and relations more transparent to them".

Feedback for the Prototype

Traceability Environment: The interviewee was particularly impressed by the drill-down feature of the traceability environment (cf. section), which allows users to get access to detailed information on demand. However, as stated in the *use cases* section, he would have liked to see more information about people in the network, and their interactions with visualizations and model elements.

Binding Data Parameters: Faced with the *visualization environment*, the interviewee expressed his praise, calling the environment "good" and "management-compatible". However, he remarked that less tech-savvy end-users would be deterred by the current approach of configuring data parameter bindings (via query specification), due to lacking knowledge of the query language and the underlying model.

By employing a "visual approach" for specifying queries instead, he thinks one could empower these users to interact with the system as well.

Professional Source Code Management: Referring to his personal experiences with developing complex custom visualizations in JavaScript, the architect expressed a need for professional source code management, e.g. driven by *Subversion* (SVN) or *Git*, as well as a test environment.

Access Control: By empowering users to analyze the networks between data assets, visualizations and people, individuals who would not have noticed before, might get implicit notifications about planned architectural changes. To avoid an unwanted spread of possibly planning informations which are subject to change, the system should provide sufficient access control capabilities.

5.3 Case 2: Financial Services

The second case evaluates the developed artifacts in the light of the financial industry—an environment where organizations may have to deal with large amounts of data, regulatory demands, and high risks.

The interview partner for this case is the *IT Infrastructure Manager* of a financial services company, which is part of a major investment and insurance group (10,001+ employees).

5.3.1 Scenario Design

Dashboard Design

As depicted in Figure 5.5, the dashboard provides users with a fictional customer's risk preference factor, as well as performance data from members of two different asset classes, namely stocks and bonds (which entail different risks). The core idea of the dashboard is that an analyst may use the data to determine a suitable allocation strategy for the customer. Asked whether the scenario makes any sense, the interviewee answered in the affirmative, stating "Yeah, of course!".

The two classes are represented by *EURO STOXX 50*, a stock index based on 50 of Europe's leading companies, and the *German 10-Year Federal Bond*. The corresponding time series data (index values and daily yields) has been extracted from *Yahoo Finance*[1] resp. the *German Federal Bank*[2], and imported into the prototype's underlying instance of SocioCortex, where the data is represented as *workspaces*, *types*, and *entities*.

Figure 5.6 illustrates the resulting data flow.

Exemplary Visualization Type

As an exemplary visualization type to suit given setting, an adapter to the *interactive financial line chart* from the library *Highstock*[3] has been implemented. As the underlying charting library supports the visualization of multiple time series, the corresponding data parameter (labeled *Series*) expects a list of structures containing a timestamp and a value, as expressed by the *MxL*-based type statement:

```
Sequence<Sequence<Structure<Timestamp: Number, Value: Number>>>
```

Figure 5.7 presents the corresponding data parameter binding for the visualization that shows the yields of the federal bond, showcasing the type checking capabilities of the prototype and its underlying expression language.

[1] Yahoo Finance: `http://finance.yahoo.com/`
[2] German Federal Bank: `http://www.bundesbank.de/`
[3] Highstock: `http://www.highcharts.com/products/highstock/`

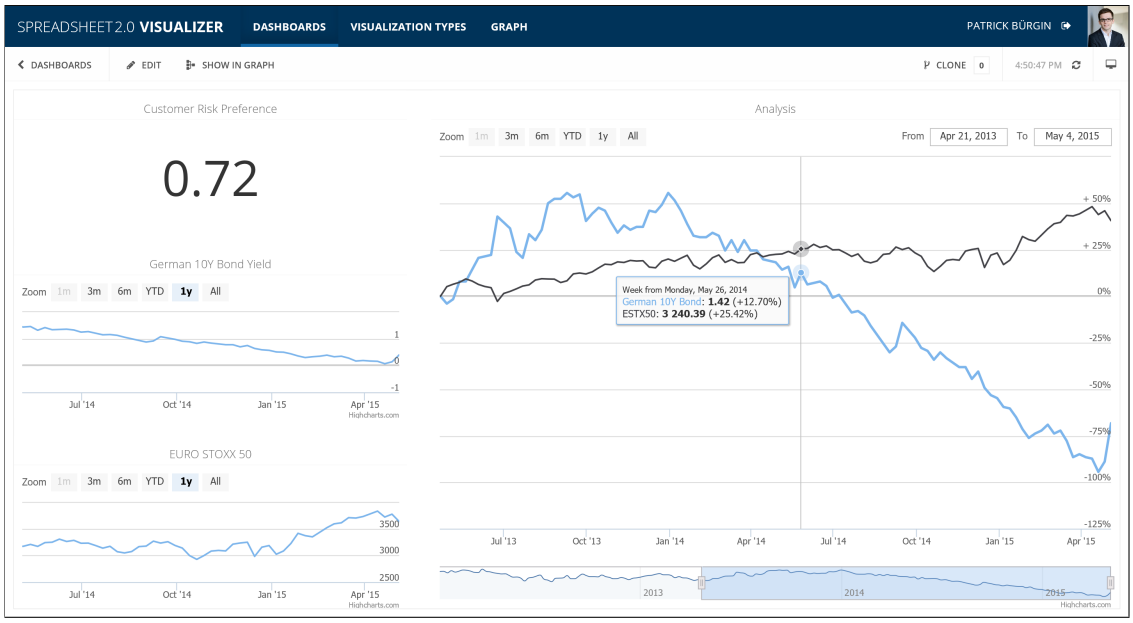


Figure 5.5: The dashboard built to support the *Financial Services* scenario, presented in the *dashboard detail view* (cf. section 4.3.3).

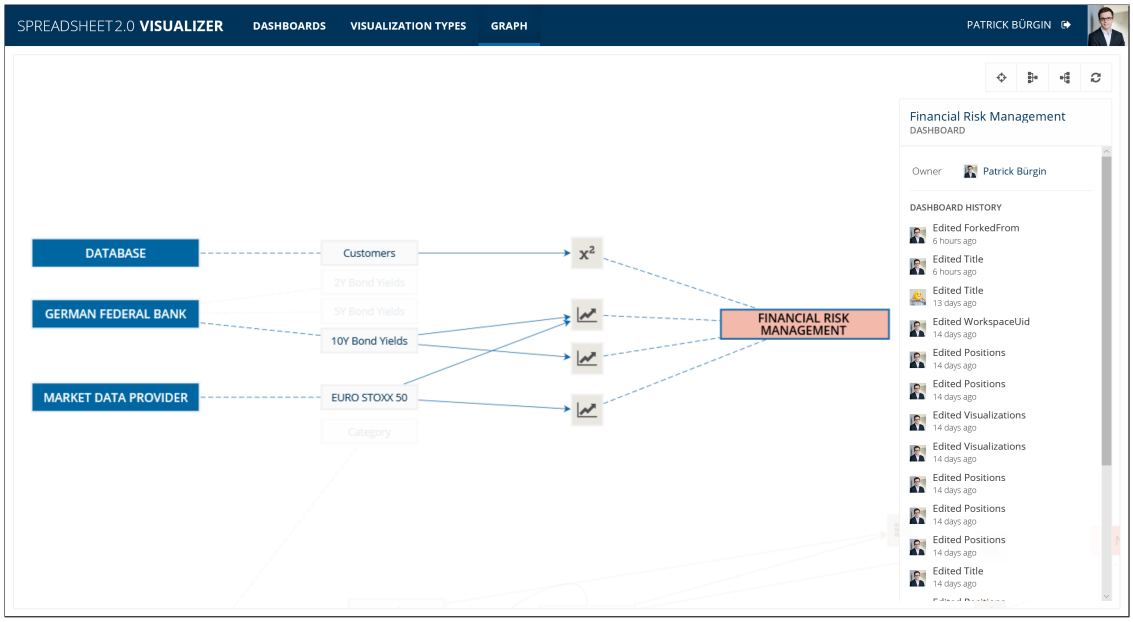


Figure 5.6: The *Enterprise Architecture Management* dashboard from Figure 5.3, presented in the *traceability environment* (cf. section 4.4). The view allows users to trace back the data depicted in the *dashboard detail view* to the three data sources.

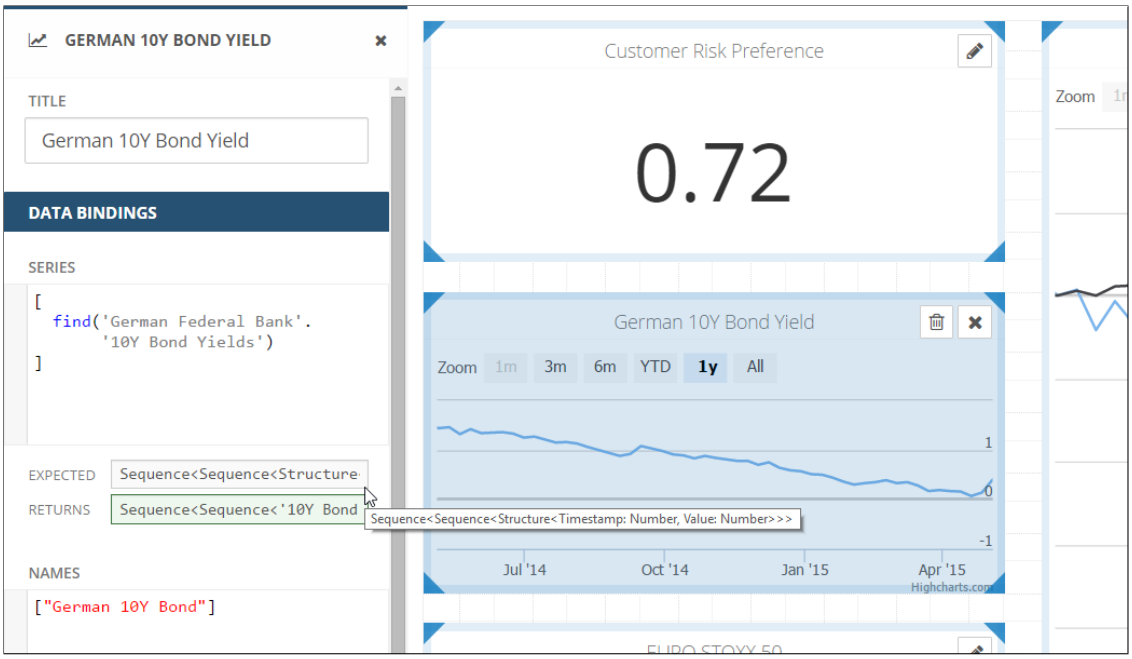


Figure 5.7: A clipping of the *dashboard edit view*, showcasing the data parameter binding to the visualization type's key parameter: *Series*.

5.3.2 Interview Report

Data Sources

The organization works with "very large amounts of high-dimensional data", e.g. analyzing time series data, sometimes updated by the second, and simulating future economic states with a variety of parameters. In addition, they receive data from customers, where they e.g. state their investment horizon, and risk budget.

Visualization Types

According to the interviewee, the most popular visualization types in daily work of the organization's financial analysts are *tables*, *bar charts*, *line charts*, *scatter plots*, *box plots*, and *tree maps*.
Asked about the importance of custom visualizations, he expressed skepticism with respect to the daily business, referring to Few, who describes visualization types like *donut charts*, *radar charts*, and *funnel charts* as "silly graphs that are best forsaken", stating that they "fail miserably at data presentation, even though their popularity is growing" [11, p. 271]. However, he also stated that they have recently adopted *D3.js* (cf. section 2.1.3) to create custom visualizations for newly emerging financial products.

Collaboration

As part of their daily work, analysts share their analyses with colleagues, and build on the results of others, e.g. by using them as input for further analyses. In that regard, a common challenge analysts face is to "find the right data for the task at hand, and access it very fast". To this end, it is "important to know where the data originated from, where it has been used, what it depends on, and wheter it is suited for the job." Asked why the choice of a market data provider matters, the interviewee responded that different providers may e.g. employ different algorithms to calculate the allegedly same measure.

Use Cases for the Traceability Environment

Support Data Exchange: As motivated in the previous section, sharing data and transformations among colleagues is "part of the daily business". Thus it is "interesting to know what the story behind a given data artifact is", raising questions such as "Where does the data come from? Who worked on it? When was the last change and by whom?".

By providing analysts with an environment to visually explore the network between data assets, visualizations, and people, a traceability environment may reduce the "significant operative cost" induced by analysts struggling to identify co-workers that faced similar problems in the past, and assess the lineage of a given data set.

Impact Analysis: In addition to supporting data exchange, a traceability environment may warn an analyst e.g. of incoming references to a script for data transformation, allowing her to get in touch with affected colleagues before committing critical changes.

Address Compliance Demands: Driven by internal (avoid reputational risk) and external (financial regulation) compliance demands, financial services organizations may have to be able to reveal the full calculation process behind a given measure or visualization on demand. A system that allows analysts and auditors to explore the network of data flows may help to meet these demands. Furthermore, the increased level of transparency can be used when introducing new hires to the environment.

Feedback for the Prototype

The interviewee was subject to a sales pitch by a major business intelligence vendor a couple of days before the interview, and thus frequently compared features of the prototypical implementation to concepts in the presented solution.

Query Flexibility: The infrastructure manager praised the system's flexibility with respect to formulating queries (using *MxL* expressions to bind data parameters to

data, cf. section 4.3.2), stating that the compared, drag-and-drop-based BI product "isn't as advanced". Asked about the increased technical barrier induced by requiring end-users to write expressions in a typed, functional language, he explained that "even the least tech-savvy individuals in our group know *R* and *MATLAB*", concluding that the added complexity would not be a disqualifier but actually an enabler.

Grid Component: Praising the signifiers and feedback provided by the grid component (cf section 4.3.2), the interviewee stated that the solution is "more intuitive" than the one provided by the popular commercial product.

Traceability Environment: Faced with the *traceability environment* and its features to support exploration (cf. section 5.2.2), as depicted in Figure 5.6, the interviewee expressed his praise, rating it as "cool". However, he would have liked to see transformations, which are not explicitly covered in the presented state of the prototypical implementation, and more details in general.

While discussing strategies to support organizations in understanding and analyzing the networks spanned by their data assets, visualizations, and people, the infrastructure manager expressed that the "view should depend on the question at hand". Asked to comment the possibility to allow users to create their own views of the network with the help of a *graph query language*, he stated that this approach would be "more powerful than providing a static perspective".

Integration & Scalability: Given that the organization employs comparatively large amounts of data, and routinely performs sophisticated data transformations and simulations with the help of scripting languages like *R*, sufficient integration and scalability are "knock-out criteria" in the given environment. As the prototypical implementation is far from meeting these demands, significant development effort would be necessary to allow for a deployment within the technical infrastructure of the organization.

5.4 Cross–Case Synthesis

In the previous sections, the developed artifacts have been assessed in the light of organizations from different data domains, using the methodology described in section 5.1. Driven by the developed scenarios, the qualitative expert interviews yielded comprehensive feedback for the prototypical implementation and domain-specific use cases for the *traceability environment*.

The prototypical implementation and its underlying concepts were received well by the interviewed experts. Both interviewees liked the general user experience and aesthetics of the prototype, and both expressed increased interest in the traceability environment, seeing possibilities to enhance the daily business in their organizations.

The remainder presents a selection of strongly overlapping findings from the two cases:

Depth of Analysis

Both interviewees expressed their praise for the *traceability environment* and its features to support exploration. At the same time, both would have liked to see more information in the view. However, their demands differed:

While in the EAM case more detailed usage information would enable architects to reduce costs and increase impact by optimizing their EA models, analysts in the financial case may benefit primarily from more detailed informations about the data lineage and applied transformations.

Impact Analysis

In both cases, impact analysis has been identified as one of the *traceability environment*'s core use cases. In the EAM case, the main appeal has been that the environment may aid less tech-savvy individuals to grow confident in using a model-based EA system. In the financial services case, collaborative work on data and scripts for data transformations could be enhanced.

Features Known from Prevalent BI Solutions

When discussing the *dashboard detail view*, the experts remarked the lack of filtering and drill-down capabilities known from prevalent business intelligence solutions as the ones introduced in section 2.1.3. Furthermore, both interviewees expressed remarks with respect to scalability and data integration capabilities of the prototype—key selling points of prevalent BI solutions, not addressed by the presented artifacts.

6 Summary and Outlook

Summary

Environment

The goal of this thesis was to create artifacts that combine the flexibility of web-based visualization toolkits with the ease of use of self-service visualization tools prevalent in today's organizations, as well as to support individuals in understanding and analyzing the networks spanned by the data assets, visualizations and people in their organization.

Knowledge Base

To this end, a broad knowledge base, comprised of current trends in related fields such as information visualization, traceability, and data governance has been spanned, driving the subsequent development of artifacts.

Develop/Build

First off, fundamental models to drive a dashboard system for visualizing data in a traceable way have been introduced:

- A generic model for reusable visualization types, and a concept how these types can be instantiated to compose dashboards.
- A holistic network model for data assets, visualizations and people in an organization, as well as ways to extract knowledge from it.

Based on these concepts, a prototypical implementation has been presented, providing modern, web-based interfaces for defining visualization types, creating and viewing dashboards, as well as visually exploring data from the network of data flows that connect data sources to dashboards.

Justify/Evaluate

In an exploratory case study with a "two-case" design, the artifacts have been subject to a first evaluation designed to play a formative role for the further development in future research endeavors. The prototypical implementation and its underlying concepts were received well by practitioners from different data-domains, and constructive feedback to drive future work could be extracted.

Outlook

The feedback and use cases that have been extracted from the exploratory case study presented in chapter 5 may play a formative role in the further development of the introduced artifacts, offering a broad range of possible areas for further investigation.

The remainder presents a selection of promising areas for future work:

Network Expansion

As discussed in the cross-case synthesis (cf. section 5.4), both interviewees would have liked to see more details of the network spanned by data assets, visualizations and people within the *traceability environment*, suggesting different paths for further expansion of the model.

The drill-down feature of the graph view, as praised by both experts, provides a good foundation for the expansion of the network, and could be combined with lazy-loading to increase the scalability of the *traceability environment* dramatically.

Furthermore, the environment may better serve its users by adopting a selection of the measures and metrics sketched in section 3.3.2.

Integration & Scalability

As discussed in section 5.1, the prototypical implementation is not yet ready for a deployment within the technical infrastructure of most real-world organizations.

By improving the artifacts with respect to data integration and scalability, such a deployment may become possible, thus allowing for more rigorous evaluations in organizational environments.

Facilitate the Binding of Data Parameters

As remarked by the interviewee in the enterprise architecture management case presented in section 5.2), less tech-savvy users may struggle to configure data parameter bindings in the prototype.

To make the solution accessible to a broader audience of users, thus allowing for more diverse feedback, one could design an optimized set of view components that allows end-users to visually compose queries without programming.

Furthermore, one could employ structural pattern matching between the information model and the demands of visualization types to further facilitate the creation of bindings, as sketched in section 3.2.2.

Bibliography

[1] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. "Model traceability." In: *IBM Systems Journal* 45.3 (2006).

[2] G. Beier. "Verwendung von Traceability-Modellen zur Unterstützung der Entwicklung technischer Systeme." PhD thesis. 2013.

[3] M. Bostock, V. Ogievetsky, and J. Heer. "D3: Data-Driven Documents." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* 17.12 (2011).

[4] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.

[5] E. H.-H. Chi and J. Riedl. "An Operator Interaction Framework for Visualization Systems." In: *Proceedings IEEE Symposium on Information Visualization (Cat. No.98TB100258)* (1998).

[6] J. Cleland-Huang, C. K. Chang, and M. Christensen. "Event-Based Traceability for Managing Evolutionary Change." In: *IEEE Transactions on Software Engineering* 29.9 (2003).

[7] B. Dinter, C. Schieder, and P. Gluchowski. *A Stakeholder Lens on Metadata Management in Business Intelligence and Big Data – Results of an Empirical Investigation*. 2015.

[8] A. M. Ernst, J. Lankes, C. M. Schweda, and A. Wittenburg. "Tool Support for Enterprise Architecture Management - Strengths and Weaknesses." In: *The Tenth IEEE International EDOC Conference (EDOC 2006)* (2006), pp. 13–22.

[9] A. M. Ernst, J. Lankes, C. M. Schweda, and A. Wittenburg. "Using Model Transformation for Generating Visualizations from Repository Contents." In: November (2006).

[10] S. Few. *Information Dashboard Design: Displaying data for at-a-glance monitoring*. 2nd ed. Analytics Press, 2013.

[11] S. Few. *Show Me the Numbers*. 2nd ed. Analytics Press, 2012.

[12] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. "A Technique for Drawing Directed Graphs." In: *Software Engineering, IEEE Transactions on* 19.3 (1993), pp. 214–230.

[13] Gartner Inc. *Gartner IT Glossary: Business Intelligence (BI)*. 2015. URL: `http://www.gartner.com/it-glossary/business-intelligence-bi/` (visited on 08/30/2015).

[14] O. C. Gotel and A. C. W. Finkelstein. "An Analysis of the Requirements Traceability Problem." In: (1994), pp. 94–101.

[15] M. Hauder and S. Roth. "A Configurator for Visual Analysis of Enterprise Architectures." In: *ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems* (2013), pp. 1–5.

[16] M. Hauder, F. Matthes, S. Roth, and C. Schulz. "Generating dynamic cross-organizational process visualizations through abstract view model pattern matching." In: *Architecture Modeling for Future Internet enabled Enterprise (AMFInE 2012)* (2012), p. 6.

[17] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research." In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[18] V. Khatri and C. V. Brown. "Designing Data Governance." In: *Communications of the ACM* 53.1 (2010), p. 148.

[19] J Kunze and T Baker. *The Dublin Core Metadata Element Set*. Tech. rep. IETF, 2007. URL: https://tools.ietf.org/html/rfc5013.

[20] F. Matthes, C. Neubert, and A. Steinhoff. "Hybrid Wikis : Empowering Users To Collaboratively Structure Information." In: *6th International Conference on Software and Data Technologies ICSOFT* (2011).

[21] K. Moreland. "A Survey of Visualization Pipelines." In: *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), pp. 367–378.

[22] T. Munzner. "Process and pitfalls in writing information visualization research papers." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4950 LNCS (2008), pp. 134–153.

[23] National Information Standards Organization. "Understanding Metadata." In: *National Information Standards* NISO Press (2004), p. 20. arXiv: 4.

[24] M. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[25] D. A. Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[26] S. I. O'Donoghue, K. S. Sabir, M. Kalemanov, C. Stolte, B. Wellmann, V. Ho, M. Roos, N. Perdigão, F. a. Buske, J. Heinrich, B. Rost, and A. Schafferhans. "Aquaria: simplifying discovery and insight from protein structures." In: *Nature Publishing Group* 12.2 (2015), pp. 98–99.

[27] M. C. F. de Oliveira and H Levkowitz. "From visual data exploration to visual data mining: A survey." In: *IEEE transactions on visualization and computer graphics* 9.3 (2003), pp. 378–394. DOI: 10.1109/TVCG.2003.1207445.

[28] T. Reschenhofer, I. Monahov, and F. Matthes. "Type-Safety in EA Model Analysis." In: *Trends in Enterprise Architecture Research Workshop (TEAR)* 9 (2014).

[29] S. Roth, M. Zec, and F. Matthes. *Enterprise Architecture Visualization Tool Survey 2014*. Tech. rep. sebis, Technische Universität München, 2014.

[30] H. J. Rubin and I. S. Rubin. *Qualitative Interviewing: The Art of Hearing Data*. SAGE Publications, 2012.

[31] R. L. Sallam, B. Hostmann, K. Schlegel, J. Tapadinhas, J. Parenteau, and T. W. Oestreich. *Magic Quadrant for Business Intelligence and Analytics Platforms*. Tech. rep. Gartner Inc., 2015.

[32] A. Satyanarayan and J. Heer. "Lyra: An interactive visualization design environment." In: *Computer Graphics Forum* 33.3 (2014), pp. 351–360.

[33] N. Shedroff. "Information interaction design: A unified field theory of design." In: *Information design* (1994), pp. 267–292.

[34] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. "A Metadata Catalog Service for Data Intensive Applications." In: *ACM/IEEE SC 2003 Conference (SC'03)* (2003).

[35] L. Wilkinson. *The Grammar of Graphics*. 2nd ed. Vol. 2. Springer, 2005.

[36] R. K. Yin. *Case Study Research: Design and Methods*. 5th ed. SAGE Publications, 2013.