Visualization of Tree-structured Data using Web Service Composition

Willy Scheibel[1], Judith Hartmann[2], Daniel Limberger[1], and Jürgen Döllner[1]

[1] Hasso Plattner Institute, Faculty of Digital Engineering, University of Potsdam
willy.scheibel@hpi.de, daniel.limberger@hpi.de, juergen.doellner@hpi.de
[2] Futurice, Berlin
judith.hartmann@futurice.com

Abstract. This article reiterates on the recently presented *hierarchy visualization service* HiViSer and its API [51]. It illustrates its decomposition into modular services for data processing and visualization of tree-structured data. The decomposition is aligned to the common structure of visualization pipelines [48] and, in this way, facilitates attribution of the services' capabilities. Suitable base resource types are proposed and their structure and relations as well as a subtyping concept for specifics in hierarchy visualization implementations are detailed. Moreover, state-of-the-art quality standards and techniques for self-documentation and discovery of components are incorporated. As a result, a *blueprint for Web service design, architecture, modularization, and composition* is presented, targeting fundamental visualization tasks of tree-structured data, i.e., gathering, processing, rendering, and provisioning. Finally, the applicability of the service components and the API is evaluated in the context of exemplary applications.

Keywords: Visualization as a Service · Hierarchy Visualization · Tree Visualization · RESTful API Design · Web-Based Visualization.

1 Introduction

Information visualization has become the prevalent way for interacting with the growing complexity, volume, and ubiquity of data. It provides the means for "bridging the two quite distinct fields of data science and human-computer interaction to help scientists and domain experts handle, explore and make sense of their data."[3] It allows users to absorb information, detect relationships and patterns, identify trends, and interact and manipulate data directly. Visualization facilitates accurate communication by aligning the mental image of and cultivating effective images to its consumers. At the same time, "web services [...] are becoming the backbone of Web, cloud, mobile, and machine learning applications" [66] in which visualization is playing a vital part. Hence, visualization services – designed, implemented, and operated based on a Software-as-a-Service,

[3] L. Micallef: "AI Seminar: Towards an AI-Human Symbiosis Using Information Visualization", 2018. https://ai.ku.dk/events/ai-seminar-luana-micallef

by means of a software delivery model – are becoming an indivisible part of said web services, providing interactive, high-quality visualization techniques across all domains [10]. To this end, we focus on *hierarchy visualization*: the visualization of hierarchically-structured – strictly speaking tree-structured – multi-variate data. Within the past three decades, over 300 hierarchy visualization techniques and variations have been proposed [54]. One reason for this diversity seems to be that tree-structured data is omnipresent in almost all application domains, e.g., demographics [29], business intelligence [47], health [14], and software development [69], but no single technique suits all tasks and applications.

This article reiterates on the design of the recently presented hierarchy visualization service HiViSer [51]. For its API specification, the *Representational State Transfer* (REST) paradigm is employed. Structure-wise, a three-tier architecture is be implemented, comprising "a client layer which provides the user interface; a stateful web service middleware layer which provides a published interface to the visualization system; and finally, a visualization component layer which provides the core functionality of visualization techniques" [72]. To avoid a rather monolithic service design and specification, we exert a separation of concerns analogously to the well-known visualization pipeline [48] into data analysis, filtering, mapping, and rendering components. With this decomposition we anticipate various benefits, namely:

- elevated unambiguity of component interfaces and capabilities,
- simplified extensibility by means of integration of new components,
- increased re-usability of components through service composition,
- high degree of data privacy and protection [32], and
- reduced complexity and increased maintainability of components.

Supplementary, this article elaborates on relevant design decisions and incorporates technical details, eventually providing blueprints that can aid implementation of future visualization components. Finally, we ensure the visualization service's API quality by adopting the Richardson Maturity Model [70], i.e., introduce the strict use of resources, remove unnecessary variation, and improve discoverability of all resources maintained by the service.

The remainder of this article is structured as follows. Section 2 provides a rough overview of related work concerned with the fundamental building blocks required for the implementation of a modern web service. The targeted scope, prevalent use cases, and applications for a hierarchy visualization service are identified and requirements for its implementation are derived and itemized in section 3. Subsequently, a concept for service orientation and service composition is derived and detailed in section 4. On this basis, the actual routes specification for a web-based hierarchy visualization API is presented in section 5 and evaluated in the context of exemplary applications in section 6. Finally, this article is concluded by a summary of its findings, an estimate of its impact, as well as an outlook into future work in section 7.

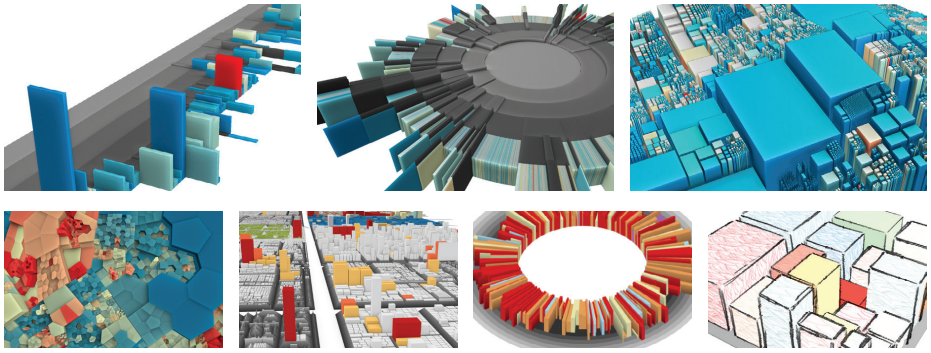


Fig. 1. Variations of hierarchy visualization techniques. From left to right and top to bottom: (1) 2.5D icicle plot, (2) 2.5D sunburst view, (3) 2.5D treemap, (4) 2.5D Voronoi treemap, (5) multiple 2.5D treemap arranged in a landscape, (5) 2.5D nested rings visualization, and (6) 2.5D treemap using sketchiness as a visual variable. Image used from "Design and Implementation of Web-Based Hierarchy Visualization Services" [51].

2 Related Work

This section references the fundamental building blocks and technical core ingredients that should be considered when designing and implementing modern services. Even though, these ingredients are most often interchangeable, at any time, there is only a small set of reliable, "industry-forged" options, so called de-facto standards. We do not cover solutions for deployment and storage since they should not interfere with the general design and implementation of any service: we expect that any given service built on below mentioned ingredients can most likely be ported to any technology stack and environment. Therefore, we focus on a brief discussion of the following implementation aspects: (1) image provisioning, (2) rendering frameworks, (3) visualization grammars and (4) Web API idioms and specification approaches.

As we follow a three-tier architecture for our service design [72], we propose data management and handling on a server, whereas a visualization client provides the graphical output and handles interaction. Image synthesis itself, however, can be executed either client-side or server-side, having the latter visualization pipeline fully executed on the server, i.e., all data processing, visualization mapping, and image synthesis.

Implementations of visualization services are usually not restricted by an environment and, thus, a broad range of implementation approaches are possible. However, since the hardware of the server is controllable and one server should serve visualization for multiple clients, there is a need for efficient data processing and image synthesis. The expectations towards rendering services are further heightened as they are the only hardware reliable enough for sophisticated rendering. This leaves visualization server implementations most likely to rely on hardware-accelerated image synthesis [50] using graphics APIs such as OpenGL or Vulkan (or other graphics library) for absolute control on the visual output [33],

or specialized rendering engines such as game engines as well as real-time visualization engines. In highly specialized cases, intermediate resources of the visualization service such as geometry artifacts or exchange formats might be highly specialized for subsequent API or engine-dependent image synthesis (e.g., using GPU-friendly attributed vertex clouds [50]). However, such specializations should be well-considered and should not lead to limitations of the visualization services capabilities. Employing service composition, in contrast, allows for arbitrary customizability and specialization of resources for a multitude of clients and visualization consumers.

On the client side, visualizations are usually displayed by Web browsers. With our service design, we strive to support a variety of different output formats not only for convenience, but also for the various application scenarios (interactive, non-interactive, etc.). Browsers can display static images with built-in techniques, e.g., the `img` tag supporting raster graphic as well as SVGs. SVGs can also be used for interactive visualizations, e.g., when manipulating the DOM with the D3 library [9]. Declarative 3D is an approach to apply the properties of SVGs to 3D visualizations. So far, declarative 3D approaches like X3DOM [5] or XML3D [62] are not supported natively by browsers and are implemented by polyfill layers [62]. It remains to be seen if these approaches can attract more popularity. Lately, the HTML5 canvas is the element of choice for the creation of hardware-accelerated graphics at run-time. It allows for visual display of 2D or 3D graphics using the Web Graphics Library (WebGL). Before the canvas was introduced, interactive visualizations were usually displayed using external plugins (Java applets, Java FX, or the Macromedia/Adobe Flash) – in a way, similar to polyfill based approaches. The use of such plugins cannot be recommended anymore, since they are widely considered to be legacy.

Taking advantage of aforementioned techniques, libraries facilitate the creation of visualization in the browser. These libraries can be distinguished into two different approaches: Imperative libraries, also called visualization toolkits, that require a developer to program the resulting visualization, and declarative libraries that allow the user to create visualizations by providing data and configuration on how to display the data. Examples of visualization toolkits are Prefuse [27], its successor Flare[4] as well as Protovis [8] and its successor D3[5]. Examples of declarative libraries are charting libraries like the SVG-based Google Visualization API[6] or the canvas-based Chart.js library[7]. For the implementation of several of our service components, we preferred more low-level, use-case agnostic rendering frameworks such as webgl-operate[8]. This has the benefit of simplifying almost always required, application-specific customizations in terms of specialized target device, offscreen rendering, advanced interaction and rendering, 3D labeling, etc.

[4] http://flare.prefuse.org/
[5] https://d3js.org/
[6] https://developers.google.com/chart/
[7] https://chartjs.org/
[8] https://webgl-operate.org

To ease the development of visualization techniques, there are multiple projects that allow to create visualizations by means of configuration. This is usually done using *visualization grammars* or domain specific languages. Examples for general visualization grammars are Vega Lite [49] and ATOM [45]. A grammar specialized for hierarchy visualization is the HiVE notation [60]. Providing a full server-client setup, the *Shiny*[9] project allows to deploy interactive visualization clients written in the R statistics language. For service composition we incorporated operations on data similar to HiVE operators, e.g., for preprocessing data or creating topology. Furthermore, multiple components (e.g., full visualization pipeline) can be invoked within a single query/request using nested objects.

The specification of possible interactions with a software component is called an *Application Programming Interface* (API). Thereby, a Web API is an API that is accessed using HTTP and related Web transmission technologies. Design approaches for Web APIs are classified as either action-based or resource-based APIs. The action-based approaches remote procedure call (RPC) protocols to trigger an action on the server. Formerly, these protocols encode calls with XML, e.g., the service-oriented architecture protocol (SOAP) [22] or XML-RPC[10]. More recent ones started to use JSON as well, e.g., JSON-RPC[11]. Although not as wide-spread, other notations are used as well by projects like gRPC[12] for example. The resource-based approaches mainly uses the architectural style of the *Representational State Transfer* (REST) for distributed hypermedia systems [18].

The specification of an API is usually provided using an *interface description languages* (IDL), describing the interfaces using a programming-language-agnostic syntax and semantics. Thereby, SOAP interfaces are commonly described in an XML-based IDL, the *Web Service Description Language* (WSDL) [15]. The RESTful equivalent to WSDL is the *Web Application Description Language* (WADL) [23]. A more recent RESTful API description language is *OpenAPI*[13], which comes with tooling that supports multiple programming languages for client libraries and IDE integration[14].

For hierarchy visualization, the HiViSer API [51] provides an entry point for hierarchy visualization as a service. However, the availability of APIs for server-side data processing for visualization and images is prevalent in other domains as well. For example, the OGC 3D Portrayal Service is a specification for web-based 3D geo-data portayal [24]. Likewise, processing and provisioning services are available for 3D point clouds [16] and image abstraction [46]. For this work, however, we predominantly implemented and tested service components that can be used for visualization of general business intelligence applications and visual software analytics applications for software system data and software engineering data.

[9] https://shiny.rstudio.com/
[10] http://xmlrpc.scripting.com/spec.html
[11] https://jsonrpc.org/specification
[12] https://grpc.io/
[13] https://www.openapis.org/
[14] http://openapi.tools/

3 Requirements of Hierarchy Visualization Techniques

The design and implementation for hierarchy visualization as a service should be aligned to prevalent usage scenarios. In order to increase flexibility, the services should support a broad range of prevalent visualization techniques and usage scenarios. These functional requirements are derived from available data processing and hierarchy visualization techniques (algorithms) as well as prevalent use cases (usage and integration scenarios of those algorithms). For the sake of brevity, this article includes a shortened analysis of techniques and use cases. However, we chose the subset to allow for generalization of the findings.

3.1 Scope of Processing and Hierarchy Visualization Techniques

We want the services to support a wide range of visualization techniques that operates on hierarchically-structured and tree-structured data. As data processing is highly domain-agnostic, usable algorithms are prelavent throughout the field of visualization [19]. The selection of the visualization techniques includes both *implicit* and *explicit* hierarchy visualization techniques [58]. In particular, we want to support the family of implicit hierarchy visualization techniques with their whole design space [57]. On the other hand, explicit hierarchy visualization should be supported as well [20]. Summarizing, the service design should allow for the techniques listed on treevis.net [54], i.e., over 300 tree visualization techniques.

Regarding the current state of this body of work, we want to limit the scope to visualization techniques of hierarchically-structured and tree-structured data. It is arguable if charts or unit visualizations [45] are a subset of hierarchy visualization techniques. However, for those categories, visualization systems are more widespread and visualization services are already available using sophisticated APIs. Further, we want to limit the supported techniques to exclude processing and depiction for additional relations among the data items (also known as *compound hierarchies*). Additionally, geo-referenced data attributes are not considered with their specific semantics but as plain data.

3.2 Use Cases of Hierarchy Visualizations

As exemplary use cases for hierarchy visualizations, we want to assess (1) the visual analysis of source code and software systems, (2) federal budget management, reporting, and controlling, (3) visual analysis on mobility data, and (4) visualization as navigational tool.

Visual Analysis of Source Code and Software. The process of creating software results in multiple sets of hierarchically-structured data. Examples are the structure of the source code modules, the team structure of developers and the association of commits to features and releases. Visualizations of these structures can facilitate conversation with stakeholders by depicting structure, behavior and evolution of the software system. As this use case is a whole field of research, we want to highlight some of the techniques. The software map [7] uses treemap

subdivision to derive its layout and has a range of extensions regarding visual variables [34] and metaphors [37]. An alternative approach is the CodeCity [71] that uses quantized node weights and a recursively packing layout algorithm. A more street-focused approach is the Software City [63] that iterate on computed layouts to handle data changes. All these approaches are 2.5D visualization techniques and are provisioned using 3D scenes. As a rather exotic visualization provisioning technique, we want to highlight the use of Minecraft worlds [2].

Federal Budget Management, Reporting, and Controlling. Government spending and income can be assigned to categories and subcategories – resulting in hierarchically-structured data. Visualizing this data allows to gain a quick overview of the categories and their impact on the federal budget and faciliates a comparison throughout the years. As an example, Auber et al. visualized the government spending of the USA using GosperMaps, squarified treemaps, and icicle plots [1]. Further, a research fund dataset was visualized using treemaps with a cascading layout postprocessing [41].

Mobility Data Visual Analysis. Slingsby et al. [59] use treemaps for a visual analysis of GPS data from delivery vehicles in central London. The data contains the vehicle position, the vehicle speed, the vehicle type and the collection time of the data. For every vehicle data is collected multiple times per minute. A visual analysis of this data can be used by the courier company to optimize the vehicle allocation, scheduling and routing. It may also be used by transport authorities to assess patterns of traffics to help set up policies to reduce congestion. Since the data includes no inherent hierarchy, Slingsby et al. use categorical values to superimpose a hierarchy; we call them variable-constrained subsets. The subsets are used in different treemap visualizations to depict different aspects of the dataset. Additionally, they derive a domain-specific language HiVE to describe hierarchy visualizations [60].

Hierarchy Visualization as Navigational Tool. By providing an overview about the structure of a data hierarchy, visualization can be used as a tool to facilitate navigation through this data hierarchy. Lui et al. [41] use the nested treemaps to navigate through 30 000 proposals for fellowships and education projects on the website of the National Science Foundation. Likewise, Guerra-Gómez et al. [21] use proportional photo treemaps as a tool to navigate through the online photo service Flickr.

3.3 Derived Functional Requirements

From the scope of targeted visualization techniques and use cases, a comprehensive list of requirements can be derived. Thereby, the main feature is the operation on tree-structured or hierarchically-structured data. For flexible use from a client perspective, the required results from the processing are (1) the processed data itself, (2) mapped data by means of geometries, scenes, and 3D models, and (3) synthesized images and videos. Following a generalized pipeline for hierarchy

visualization (cf. Figure 2), specific requirements are assigned to (a) datasets, (b) data preprocessing, (c) topology preprocessing, (d) layouting, (e) mapping and visual variables, (f) image synthesis, and, subsequent, (g) provisioning. A per-feature exposed parameterization allows for fine-grained control on the results.

Requirements on Datasets. The services should handle data from different application domains. This includes (f1) spatio-temporal, (f2) multi-variate, (f3) multi-modal, (f4) multi-run, and (f5) multi-model data [31]. Thus, a user should be able to provide heterogeneous data and the service should provide support to integrate the data. As part of dataset management, we want to support analysis on pre-existing data, e.g. train machine learning models [6], as well.

Requirements on Data Preprocessing. The services should support basic data preprocessing operations such as resampling, normalization, quantization [71], and filtering of outlier. More specific to hierarchy visualization, up-propagation and down-propagation of data is required, too. One particular metric for visualization that is computed as part of preprocessing is the degree-of-interest of nodes [68].

Requirements on Topology Preprocessing. Services should support basic operations on the topology – the relations between nodes – to specify the tree-structure for subsequent visualization. This includes subtree selection, node filtering, and subtree creation, e.g., through hierarchy operations [60]. More idiomatic, this step includes the extraction of a coarse-grained view on a hierarchy by means of aggregated views [17].

Requirements on Layouting. For tree layouting, all layout algorithms that operate on a tree data structure should be supported, i.e., all implicit and explicit tree layout algorithms. For explicit tree layouts, services should support node-link diagrams [44] and point-based depictions [56]. For implicit tree layouts, this includes rectangular treemap layout algorithms [69], as well as generalized algorithms [3]. Thereby, algorithms should not be limited in layout complexity and rather should allow for use of GosperMaps [1] and mixed polygonal layouts as well [25]. There are algorithms that use a mixed representation of implicit and explicit relations between nodes, but should be supported nevertheless. Examples include general rooted tree drawings [55]. As additional feature, the services should support for layout evolution, i.e., reusing previous layouts to compute the current one. This should work for implicit tree layouts [53], mixed representations [63], and general node-link diagrams [13].

Requirements on Visualization Mapping. Supported visual variables are target to the used visualization technique. Services should support to expose them to a client. As starting point, the standard set of visual variables has to be supported [12]. More advanced – but still considered – visual variables are sketchiness [73] and natural metaphors [74]. Next to specific visual variables, the services should support to encode two [67] and multiple points in time [63]. Additionally, labeling and glyphs should be supported, too [61].

Requirements on Geometries. The visualization service should support the geometries of implicit and explicit tree visualizations [57]. Existing techniques mainly use different types of primitive geometries (e.g., points, lines, rectangles, cuboids, polygons, or general triangle meshes). However, there are visualization techniques that use highly-specific types of scenes [2]. The type of geometry usually requires the use of specific renderers to synthesize the visualization images.

Requirements on Rendering. As renderer are highly specific, a fine-grained parameterization should be supported. Typical parameters include camera specification, enabling of rendering features (e.g., shadows, lighting, use of materials), visible semantic layers [24], or the rendering quality (e.g., number of intermediate frames for progressive rendering approaches [33]).

Requirements on Provisioning. For provisioning of results to a client, the service should be flexible as well. This means, that the services should support to provide final visualization images, but intermediate results as well (i.e., derived data [40], layouts, geometries, and models [38]). To this end, sophisticated renderer should not solely create the final image, but intermediate buffer with geometric semantics – the g-buffers – as well [36]. Typically, the visualization images and additional g-buffer can be thought of as intermediate results and can be used as input for postprocessing [46] or video creation.

4 Concept for Service Orientation and Composition

Based on the requirement analysis, we propose a service composition of multiple services to allow for the use cases and integration scenarios. Thereby, our focus is on two services that are specific to tree-structured data and its visualization. Further, an integration into a larger service landscape is outlined. The service landscape is aligned to a visualization pipeline model for tree visualization (cf. Figure 2). The internal data model of the services is separated into concepts on the underlying visualization implementations (capabilities) and concepts for user data (operational data).

4.1 Service Composition and Integration

Following the visualization pipeline model and the enclosing visualization process, the three technical phases to derive the visualization image are *preprocessing and filtering of data*, *mapping the preprocessed data*, and *rendering the mapped data* [48]. We propose a service landscape focusing on this pipeline model. Thereby, we introduce a separation between the data management and processing service and the data visualization service (cf. Figure 2). A thorough service integration has a service for problem data retrieval (data mining) as well. Services extending on the visualization images are postprocessing services or video services. Alternateively, external services may use intermediate results as subject of analysis

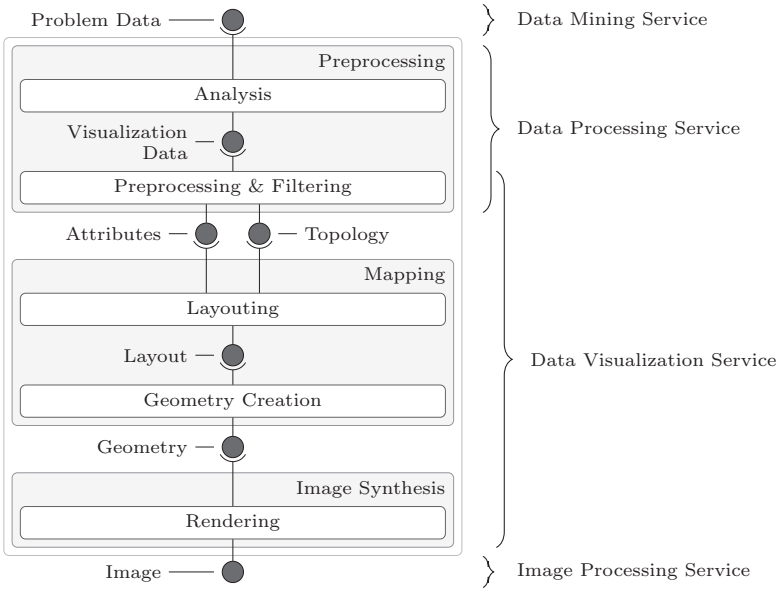


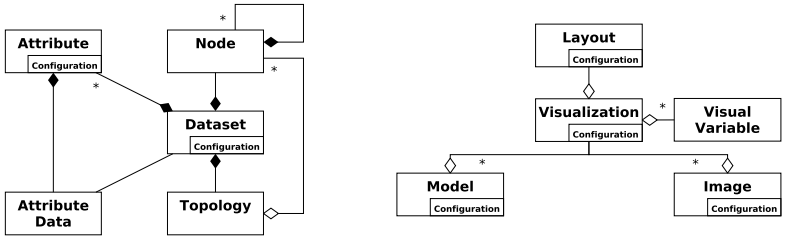
Fig. 2. The proposed tree visualization pipeline and a proposed service association. The specifics for tree visualization techniques are coined with the explicit model of attribute data and tree topology as separate inputs for the explicit layouting phase prior to geometry creation. In order to support a wide range of use cases and system integrations, having access to the final result (the image) as well as all intermediate results (attribute data, tree topologies, layouts, and geometries) is required as well.

(e.g., using layouts to derive layout metrics [69]). Although we propose a separation of data processing service and visualization service, we can imagine that an actual implementation can provide both functionalities and therefore exposes interfaces for both (e.g., HiViSer is designed this way).

4.2 Derived Operational Data Model

The operational data model represents the user data and derived data throughout visualization. For the data processing service, we propose the concepts *Dataset*, *Topology*, *Attribute*, and *Attribute Data*. For the visualization service, we propose the concepts *Visualization*, *Layout*, *Model*, and *Image*. For the concepts Dataset, Attribute, Layout, Visualization, Model, and Image, we include explicit configurations, each. The proposed concepts are the communication artifacts between services and clients and can be actively managed by the user, i.e., create, read, update, and delete (CRUD approach).

Dataset and Topology. A Dataset represents a set of nodes, their tree-structured Topology (their parent-child relationship by means of edges), and a number of Attributes with values – their Attribute Data. Although depending the Topology



a) Data Processing Operational Model b) Visualization Operational Model

Fig. 3. The operational models for (a) the data processing service and (b) the visualization service.

Format, we suggest that a topology contains information on all nodes, their ids (identifier), and parent node. Additionally, node meta information as node labels can be added, too. As the topology is a specific view on the Dataset and changes to the Dataset are promoted through use of Dataset Processing Techniques, a Topology cannot be created, updated, or deleted independently. When not provided directly, e.g., through a data mining service or through direct upload, a dataset can be derived from another dataset by use of a Dataset Processing Technique. This is encoded using a Dataset Configuration that can be re-used for multiple Datasets.

Attribute and Attribute Data. A measure that is associated with the nodes of a Dataset is represented by an Attribute. The Attribute Data allows to query a value for each node and Attribute. An Attribute Configuration allows to derive an Attribute from another Attribute, including implementation-specific computation and derivation of the per-node values.

Visualization and Layout. A Visualization represents the concept of a configured mapping from data to a Layout and Visual Variables. Thereby, a Layout represents a spatial position and possibly an extent for each node of a Topology. A Visualization Configuration allows to re-use previously defined configurations across datasets. Likewise, a Layout Configuration allows to re-apply layout techniques to other datasets.

Model and Image. A Model represents the required graphical primitives, geometries, or the virtual scene to render depictions of a Visualization. An Image thereby represents a server-rendered image of a Visualization. The Model Configuration and Image Configuration concepts allow to re-use a configuration across datasets.

4.3 Derived Capabilities Model

The capabilities model represents the exposed implementation specifics of a service. For the data processing service, we propose the concepts *Dataset Type*, *Dataset Processing Technique*, *Topology Format*, *Data Processing Technique*, and

Data Format. For the visualization service, we propose the concepts *Visualization Technique*, *Layout Technique*, *Layout Format*, *Visual Variable Mapping*, *Model Format*, *Rendering Technique*, and *Image Format*. The proposed concepts expose the available operations, data formats, and parameterization to other services and clients. These concepts are not meant to be actively managed by a user, i.e., they are designated to be read-only. As the content of these concepts is directly derived from implementation specifics, only a change the implementation should change the contents.

Dataset Capabilities. The Dataset Type encapsulates a set of implementation-specific properties of a Dataset. This includes associated Dataset Processing Techniques, Topology Formats, Data Processing Techniques, and Data Formats. A Dataset Processing Technique represents an implementation of a mapping of one Dataset to another, derived, Dataset. Likewise, a Data Processing Technique represents an implementation to derive Attributes and their data from other Attributes. For efficient transmission of results and intermediate results, a service can provide data using different transmission formats. The proposed services support this by use of Topology Formats for Topologies and Data Formats for Attribute Data.

Visualization Capabilities. A Visualization Technique encapsulates a set of implementation-specific properties of a Visualization. This includes associated Layout Techniques, Layout Formats, Visual Variable Mappings, Model Formats, Rendering Techniques, and Image Formats. Thereby, a Layout Technique is an implementation that maps a node to a spatial position and possibly an extent. A Layout Format represents a specific encoding of a Layout for provisioning. For Visual Variable Mapping, this concept encapsulates an Attribute that is mapped to a visual variable, i.e., a mapping to a spatial or otherwise visual encoding. A Rendering Technique represents an implementation that processes a Visualization and produces Images, i.e., color depictions and possibly with additional data buffers that encode additional per-pixel information. For efficient transmission of results and intermediate results, a visualization service can provide Models and Images using different encodings. To support this, a Model Format represents an implementation-specific encoding of a Model and an Image Format represents an encoding for Images, respectively.

5 Hierarchy Visualization API

Derived from the service concept, we propose a RESTful API that allows to access the services. This API is designed with an extended stakeholder model in mind as well as additional non-functional requirements. Essentially, each model from the service concept is mapped to a route, and thus a REST resource. These routes are meant to be general extensions points for specific hierarchy visualization services by specialization of requests and responses, rather than adding new resources or routes to the API. Specifics of visualization implementations are expected to extend on the base resources instead of adding new ones.

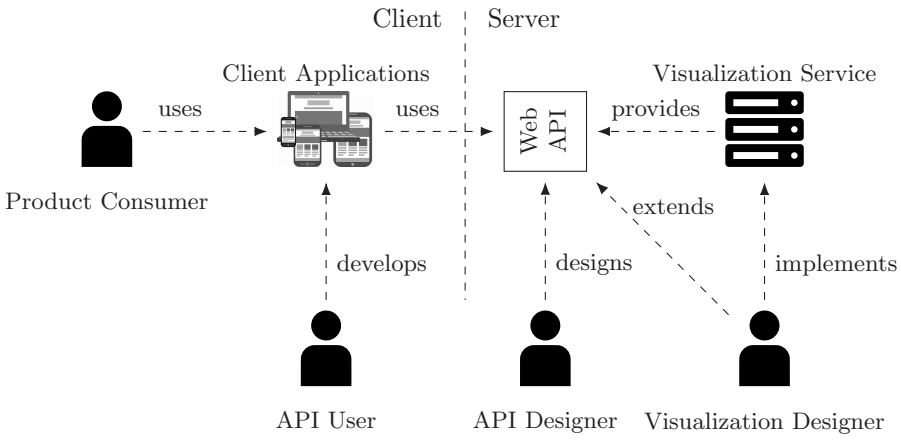


Fig. 4. There are four groups of stakeholders that interact with a hierarchy visualization service directly or indirectly. Mainly, the service is shaped by the API designer and the visualization designer. The API user develops client applications that use a hierarchy visualization service to manage tree-structured data or display depictions of the data. These client applications are used by the product consumer.

5.1 Stakeholder

Stylos and Myers describe three stakeholders that interact with an API directly or indirectly: the API designer, the API user and the product consumer [64]. They imply that an API is created to provide an interface for an existing implementation. To this end, no stakeholder for implementation of a system using an existing API Design as an interface is considered. We explicitly want to design a process with such a visualization designer stakeholder and, thus, added this role to the concept of API stakeholders (cf. Figure 4).

5.2 API Design Requirements

In contrast to provisioning of actual features, an efficient API should be further constrained in *how* the features are provided. From an API design viewpoint – and disregarding the domain of hierarchy visualizations –, a hierarchy visualization service API should support the following general requirements for efficient usage:

Adaptability and Extensibility It must be possible to adapt the API to the capabilities of the visualization service. This adaptation is characterized by extending concepts of the abstract hierarchy visualization service – instead of adding new ones – so all adapted hierarchy visualization service APIs keep a consistent structure.

Documentation of General Specification A hierarchy visualization service API must be specified unambiguously. A client application programmer who uses a specific hierarchy visualization service via Web API must know how to use and adapt it.

Documentation of Adapted Specification Further, an API user should be able to discover all routes and understand input parameters and their meaning without looking at external documentation. A hierarchy visualization service must specify a way to include comprehensive documentation as part of the web service.

More specific to RESTful APIs, the REST standard architecture defines constraints on the API design [18]: *Client-Server Architecture*, *Layered System*, *Stateless*, *Cacheability*, *Code on Demand*, and *Uniform Interface*. In order to obtain a uniform interface, further architectural constraints are defined: *Identification of Resources*, *Manipulation through Representations*, *Self-descriptive Messages*, and *Hypermedia as the Engine of Application State*. Thus, an additional requirement is the reusability by means of re-usage of information and previously submitted or computed results on a visualization service. This facilitates the creation of visualizations with the same options for different data as the options do not have to be recreated. Further, reusability of information allows to use already processed information, instead of processing the same information twice. To ensure an efficient process, the amount of requests and the request response time should be as minimal as possible. The quality of a RESTful API can be measured by assessing its maturity with respect to the *Richardson Maturity Model* [70]. Our target is quality level 3 of the RMM, i.e., (1) using a different URL for every different resource, (2) using at least two HTTP methods semantically correct, and (3) using hypermedia in the response representation of resources.

5.3 Routes & Mapping

The design of a service composition with data processing service and visualization service as well as the distinction between an operational data model and a capabilities model results in four parts of our proposed API. There is (1) the routes for the operational data of the data processing service (main route is `/datasets/`, cf. Table 1), (2) the routes for the capabilities of the data processing service (main route is `/datasettypes/`, cf. Table 2), (3) the routes for the operational data of the visualization service (main route is `/visualizations/`, cf. Table 3), and (4) the routes for the capabilities of the visualization service (main route is `/visualizationtechniques/`, cf. Table 4). Thereby, each route allows for query of single elements and whole collections of the associated resource.

Next to the domain-specific routes for the data processing service and the visualization service, we define two additional routes to satisfy the requirements of the documentation on the API specification and the *Hypermedia as the Engine of Application State*:

`/` The root route that lists all main resources, i.e., Datasets, Dataset Types, Visualizations, and Visualization Techniques.

`/api` The route that provides the API specification.

Table 1. The mapping from the operational data resources of the data processing service to RESTful API routes.

Operational Data	Route	Element	Collection
Datasets	`/datasets/[id]`	✓	✓
Topology	`/datasets/<id>/topology/[id]`	✓	✓
Attribute	`/datasets/<id>/attributes/[id]`	✓	✓
Attribute Data	`/datasets/<id>/attributes/<id>/data/[id]`	✓	✓
Dataset Configurations	`/dataset-configurations/[id]`	✓	✓
Attribute Configurations	`/attribute-configurations/[id]`	✓	✓

Table 2. The mapping from the capabilities resources of the data processing service to RESTful API routes.

Capability	Route	Element	Collection
Dataset Types	`/datasettypes/[id]`	✓	✓
Dataset Processing Techniques	`/datasettypes/<id>/datasetprocessingtechniques/[id]`	✓	✓
Topology Formats	`/datasettypes/<id>/topologyformats/[id]`	✓	✓
Data Processing Techniques	`/datasettypes/<id>/dataprocessingtechniques/[id]`	✓	✓
Data Formats	`/datasettypes/<id>/dataformats/[id]`	✓	✓

Table 3. The mapping from the operational data resources of the visualization service to RESTful API routes.

Operational Data	Route	Element	Collection
Visualizations	`/visualizations/[id]`	✓	✓
Visualization Configurations	`/visualization-configurations/[id]`	✓	✓
Layouts	`/visualizations/<id>/layouts/[id]`	✓	✓
Layout Configurations	`/visualizations/<id>/layout-configurations/[id]`	✓	✓
Models	`/visualizations/<id>/models/[id]`	✓	✓
Model Configurations	`/visualizations/<id>/model-configurations/[id]`	✓	✓
Images	`/visualizations/<id>/images/[id]`	✓	✓
Image Configurations	`/visualizations/<id>/image-configurations/[id]`	✓	✓

Table 4. The mapping from the capabilities resources of the visualization service to RESTful API routes.

Capability	Route	Element	Collection
Visualization Techniques	`/visualizationtechniques/[id]`	✓	✓
Layout Techniques	`/visualizationtechniques/<id>/layouttechniques/[id]`	✓	✓
Layout Formats	`/visualizationtechniques/<id>/layoutformats/[id]`	✓	✓
Visual Variable Mappings	`/visualizationtechniques/<id>/visualvariables/[id]`	✓	✓
Model Formats	`/visualizationtechniques/<id>/modelformats/[id]`	✓	✓
Rendering Techniques	`/visualizationtechniques/<id>/renderingtechniques/[id]`	✓	✓
Image Formats	`/visualizationtechniques/<id>/imageformats/[id]`	✓	✓

5.4 Adaptation Process

The API is designed to support various hierarchy visualization techniques and implementations. Therefore, the API designer should be able to adapt the API to the capabilities of the visualization service – at best, without structural changes to the API. This is accomplished as the API designer does not need to create new base resource types, irregardless their relation to the creation of an image, a model, or other results from the service. Instead, the base resource types have to be extended, i.e., subtypes have to be created. Typically, this adaption is done continuously within the development of the visualization service.

6 Evaluation

Our proposed service landscape and the API are evaluated by use of example applications and assessment. Thereby, we want to introduce a treemap visualization service, where we want to consider treemaps as a common example of hierarchy visualization techniques. Next, we want to take up on the former use cases and assess the use of the proposed API. Last, we describe the differences between the reiterated, proposed API and the former HiViSer API.

6.1 Treemaps as a Service

To demonstrate the service landscape, we describe a treemap visualization service. The service provides data management for tree-structured data as well as configuration and provisioning of treemap-based data visualization. Thereby, the service supports the proposed hierarchy visualization API while being restricted to treemap visualization techniques. The available processing and visualization techniques and the implementation specifics of the service are as follows:

Data Management and Preprocessing. Regarding source data, an end user can create own datasets via Datasets and upload data as CSV, i.e., we do not utilize a separate data mining service. The file should contain one row per leaf node and one column per attribute. The first column is expected to contain the node identifier with slashes as delimiters to encode the parent path of nodes (e.g., a file path). The parts of the node identifier are used as a nominal attribute for node labeling (for both inner and leaf nodes). The other columns of the CSV are used as further numeric attributes that are registered within the service and exposed via the API. The topology of the tree is extracted from the encoded path of the CSV file. Derived data is provided by means of attribute value normalization, transformations (e.g., quantization), and computation of a degree of interest [37]. The supported output formats are an edge list for the topology (pairs of parent identifier and child identifier for each edge using a breadth-first order) and a value buffer for attribute data (float-encoded values using breadth-first order).

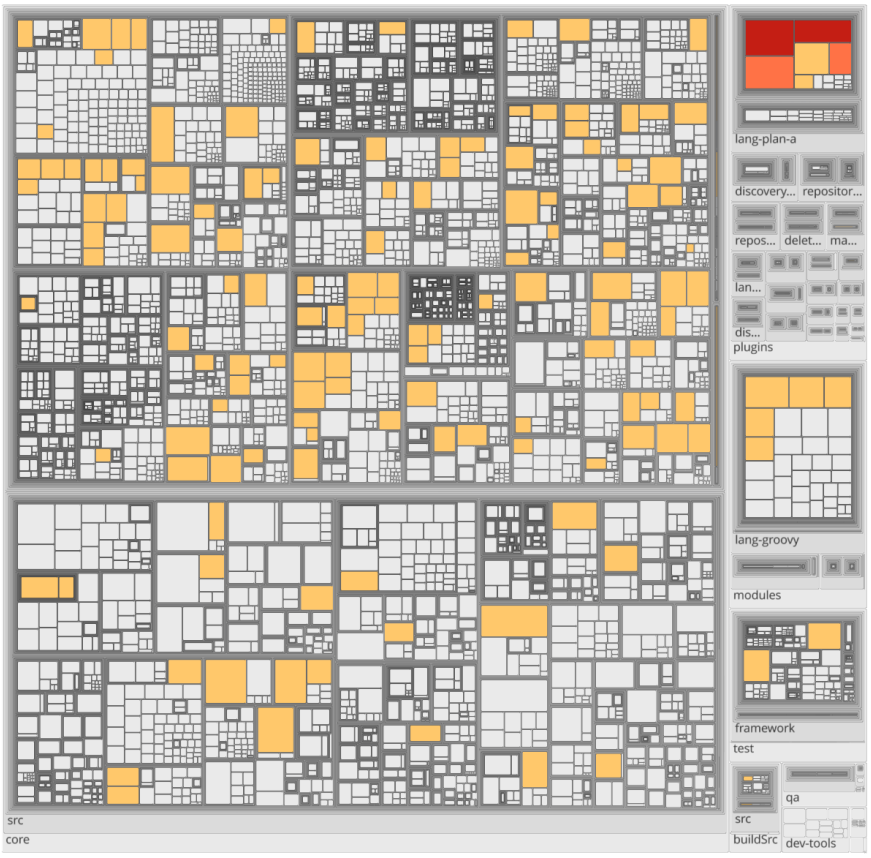


Fig. 5. A visualization result using a software dataset snapshot from ElasticSearch from 2014. The dataset is depicted using a 2D treemap, color mapping, and labeling.

Layouting and Mapping. The service provides rectangular treemap layouting algorithms – namely Slice'n'Dice [30], Squarified [11], Strip [4], Hilbert, Moore [65], Approximation [43], and EvoCells [53] – and additional layout margins for hierarchical structure depiction. The supported visual variables are color with ColorBrewer color schemes [26], height, sketchiness [34], and in-situ geometries [39] – allowing the creation of 2.5D treemaps (cf. Figure 6). Labeling is supported through use of OpenLL [35]. The supported output formats for the treemap geometry are XML3D, X3DOM, glTF [38], VRML, and – as GPU-native format – a buffer encoding with attributed vertex cloud encoding [50].

Rendering. The treemap images are rendered using progressive rendering [33], allowing for both basic 2D depictions and sophisticated graphical effects for virtual 3D environments. Next to the rendered image, the g-buffers for per-pixel buffers using false-color-encoded ids and normal vectors are provided, too.

Fig. 6. A visualization result of a software system dataset. The layout was created using EvoCells layouting. The resulting visualization is a 2.5D treemap with color and height mapping, in-situ diff visualization, and labeling.

Supported Client Approaches. With this set of features and interfaces, the service allows for the following types of clients: (1) a thin client, that displays server-rendered visualization images and maps user interaction to service requests and updates to the displayed image, (2) a mixed approach where the client displays a 3D model of a visualization in the web browser by use of a polyfill layer, and (3) a rich client, that uses only preprocessed data and a topology to perform layouting, geometry creation, and rendering on the client. The latter approach provides the most flexibility with least requests to the service but imposes more load to the client.

6.2 Use Case Evaluation

A use case assessment on the proposed API results in the following observations:

Visual Analysis of Source Code and Software. The API can represent software datasets with multiple revisions – multiple Datasets – and multiple attributes. Thereby, the model is flexible enough to allow different types of nodes (e.g., developers, source code modules, and commits). Further, the API allows to use different data preprocessing, layout algorithms and visual variables to allow for the targeted software visualization techniques. With the flexibility to provide implementation-defined 3D model formats, even Minecraft worlds are within the scope of the API.

Federal Budget Management, Reporting, and Controlling. As with software data, budget data is representable using the API as well. The creation of the hierarchy through use of categorical grouping is supported through dataset processing techniques. The support of polygonal layouts, e.g., for GosperMaps, is assured

```
{ "visualization": {
    "type": "/visualizationtechniques/2.5dtreemap",
    "options": {
      "dataset": "/datasets/Tierbestand2014",
      "layout": {
        "type": "striptreemap",
        "options": {
          "weights": { "type": "normalization",
            "options": { "source": "count", "min": 0, "max": "source" } },
          "parentPadding": 0.01, "siblingMargin": 0.03 }
      },
      "visualVariables": [
        { "type": "leaf-colors",
          "options": { "attribute": "count", "gradient": "colorbrewer-3-OrRd" } },
        { "type": "leaf-heights", "options": { "attribute": "count" } }
      ],
      "labeling": true
    }
  },
  "options": {
    "width": 1920, "height": 1080,
    "eye": [ 0.0, 1.2, 0.8 ],
    "center": [ 0.0, 0.0, 0.0 ],
    "up": [ 0.0, 1.0, 0.0 ],
    "quality": 100, "format": "png" } }
```

Listing 1.1. Example request JSON for a treemap image at the route `/visualizations/Tierpark2014/images`. This request was issued after the dataset `Tierpark2014` was created and configured.

though an abstraction layer that places this responsibility on the implementation of the service. The evolution of layouts is supported through re-usability of precomputed layouts and specific operations that implement the evolution.

Mobility Data Visual Analysis. From an API perspective, this use case is similar to the federal budget management. However, the use of the HiVE visualization grammar is possible to be used with our API structure, e.g., by providing an implementation of a dataset processing technique that parses the HiVE grammar.

Hierarchy Visualization as Navigational Tool. The use for a navigational tool is more broad as it adds the navigation task to the display of images. With the provisioning of both images and id-buffers, a user click into the image can be converted into the clicked node id. This node id can be used to select the original node and its metadata to extract the navigation target and, thus, allow to navigate the user further.

6.3 Comparison to HiViSer API

A comparison of the proposed API to the HiViSer API can be described as a mapping of pre-existing routes and the addition of routes to self-describe transmission formats and available processing and visualization techniques. For us, the concept of DataSources from HiViSer is conceptually located in the data mining

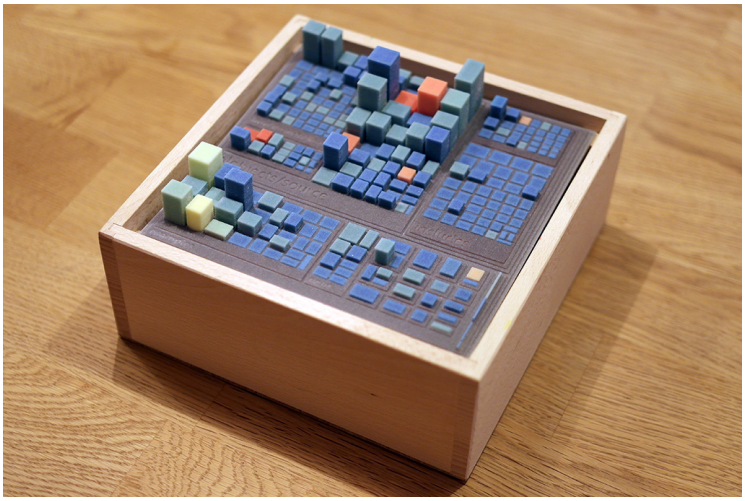


Fig. 7. A 3D printed treemap from a model that was created using a treemap service, embedded into a wooden frame.

service, and, thus, not focus of the API proposed in this article. The original concept of Datasets and Nodes is adapted and mapped to routes in a similar fashion. However, with this proposed API, the attributes are modeled as part of a dataset and Buffers are represented as Attribute Data. BufferViewOptions and BufferTransformations are remodeled to Data Processing Techniques and references to Attribute Data. The concepts Layout, Images, and Models are now part of Visualizations. The VisualVariable was moved to the self-description interface of Visualization Techniques. Labeling is currently not part of the proposed API – although we believe it can be specified by adding Labels to the data processing service and Labeling to the visualization service (similar to the HiViSer API).

7 Conclusions

We decomposed the recently presented hierarchy visualization service HiViSer analogously to the stages of the visualization pipeline model, with focus on data processing and visualization. Thereby, we described a resource-based Web API for tree-structured data management and visualization, provided instructions for the specification of its base resources, and specified routes targeting OpenAPI. We think this not only will facilitate the integration into a larger service landscape for data mining, postprocessing, video generation but also ease the implementation of future visualization APIs. Overall, we presented a proposal for Visualization-as-a-Service for tree-structured data that strives to make the development of visualization clients faster, more flexible, and less error-prone. At best, we hope this to be a valuable step towards standardization of the API and associated communication artifacts.

For future work, several directions are designated for extending the API. For example, we plan to formalize the labeling part of the API and add management for and visualization of additional relations [28]. We also imagine configurations of images, where multiple depictions are embedded within one. The latter would allow for small multiples visualizations [52]. At large, we want to evaluate if the API design is applicable to a broader class of data and corresponding information visualization techniques, to support the "[i]ntegration and analysis of heterogeneous data" and therefore tackle "one of the greatest challenges for versatile applications" [42] in information visualization.

Acknowledgments

This work was partially funded by the German Federal Ministry of Education and Research (BMBF, KMUi) within the project "BIMAP" (www.bimap-project.de) and the German Federal Ministry for Economic Affairs and Energy (BMWi, ZIM) within the projects "ScaSoMaps" and "TASAM".

References

1. Auber, D., Huet, C., Lambert, A., Renoust, B., Sallaberry, A., Saulnier, A.: Gospermap: Using a gosper curve for laying out hierarchical data. IEEE Transactions on Visualization and Computer Graphics **19**(11), 1820–1832 (2013). https://doi.org/10.1109/TVCG.2013.91
2. Balogh, G., Szabolics, A., Beszédes, Á.: Codemetropolis: Eclipse over the city of source code. In: Proc. IEEE International Working Conference on Source Code Analysis and Manipulation. pp. 271–276. SCAM '15 (2015). https://doi.org/10.1109/SCAM.2015.7335425
3. Baudel, T., Broeksema, B.: Capturing the design space of sequential space-filling layouts. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2593–2602 (2012). https://doi.org/10.1109/TVCG.2012.205
4. Bederson, B.B., Shneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. ACM Transactions on Graphics **21**(4), 833–854 (2002). https://doi.org/10.1145/571647.571649
5. Behr, J., Eschler, P., Jung, Y., Zöllner, M.: X3DOM - A DOM-based HTML5/ X3D Integration Model. In: Proc. ACM International Conference on 3D Web Technology. pp. 127–135. Web3D '09 (2009). https://doi.org/10.1145/1559764.1559784
6. Bethge, J., Hahn, S., Döllner, J.: Improving Layout Quality by Mixing Treemap-Layouts Based on Data-Change Characteristics. In: Proc. EG International Conference on Vision, Modeling & Visualization. VMV '17 (2017). https://doi.org/10.2312/vmv.20171261
7. Bohnet, J., Döllner, J.: Monitoring code quality and development activity by software maps. In: Proc. ACM Workshop on Managing Technical Debt. pp. 9–16. MTD '11 (2011). https://doi.org/10.1145/1985362.1985365
8. Bostock, M., Heer, J.: Protovis: A graphical toolkit for visualization. IEEE Transactions on Visualization and Computer Graphics **15**(6), 1121–1128 (2009). https://doi.org/10.1109/TVCG.2009.174

9. Bostock, M., Ogievetsky, V., Heer, J.: D3 data-driven documents. IEEE Transactions on Visualization and Computer Graphics **17**(12), 2301–2309 (2011). https://doi.org/10.1109/TVCG.2011.185
10. Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q.Z., Dong, H., Yu, Q., Neiat, A.G., Mistry, S., Benatallah, B., Medjahed, B., Ouzzani, M., Casati, F., Liu, X., Wang, H., Georgakopoulos, D., Chen, L., Nepal, S., Malik, Z., Erradi, A., Wang, Y., Blake, B., Dustdar, S., Leymann, F., Papazoglou, M.: A service computing manifesto: The next 10 years. Communications ACM **60**(4), 64–72 (2017). https://doi.org/10.1145/2983528
11. Bruls, M., Huizing, K., van Wijk, J.: Squarified Treemaps. In: Proc. Springer Data Visualization. pp. 33–42 (2000). https://doi.org/10.1007/978-3-7091-6783-0_4
12. Carpendale, M.S.T.: Considering visual variables as a basis for information visualization. Tech. rep., University of Calgary, Canada (2003). https://doi.org/10.11575/PRISM/30495, nr. 2001-693-14
13. Caudwell, A.H.: Gource: Visualizing software version control history. In: Proc. ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion. pp. 73–74. OOPSLA '10 (2010). https://doi.org/10.1145/1869542.1869554
14. Chazard, E., Puech, P., Gregoire, M., Beuscart, R.: Using Treemaps to represent medical data. IOS Studies in Health Technology and Informatics **124**, 522–527 (2006), http://ebooks.iospress.nl/volumearticle/9738
15. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language (2007), http://www.w3.org/TR/2007/REC-wsdl20-20070626
16. Discher, S., Richter, R., Trapp, M., Döllner, J.: Service-oriented processing and analysis of massive point clouds in geoinformation management. In: Service-Oriented Mapping, pp. 43–61. Springer (2019). https://doi.org/10.1007/978-3-319-72434-8_2
17. Elmqvist, N., Fekete, J.D.: Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. IEEE Transactions on Visualization and Computer Graphics **16**(3), 439–454 (2010). https://doi.org/10.1109/TVCG.2009.84
18. Fielding, R.T.: REST: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
19. García, S., Luengo, J., Herrera, F.: Data preprocessing in data mining. Springer (2015). https://doi.org/10.1007/978-3-319-10247-4
20. Graham, M., Kennedy, J.: A survey of multiple tree visualisation. SAGE Information Visualization **9**(4), 235–252 (2010). https://doi.org/10.1057/ivs.2009.29
21. Guerra-Góomez, J.A., Boulanger, C., Kairam, S., Shamma, D.A.: Identifying best practices for visualizing photo statistics and galleries using treemaps. In: Proc. ACM International Working Conference on Advanced Visual Interfaces. pp. 60–63. AVI '16 (2016). https://doi.org/10.1145/2909132.2909280
22. Hadley, M., Gudgin, M., Karmarkar, A., Mendelsohn, N., Nielsen, H.F., Lafon, Y., Moreau, J.J.: SOAP version 1.2 part 1: Messaging framework (second edition). Tech. rep., W3C (2007), http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
23. Hadley, M.J.: Web application description language (WADL). Tech. rep., Sun Microsystems Inc. (2006)
24. Hagedorn, B., Coors, V., Thum, S.: OGC 3D portrayal service standard. Tech. rep., Open Geospatial Consortium (2017), http://docs.opengeospatial.org/is/15-001r4/15-001r4.html

25. Hahn, S., Döllner, J.: Hybrid-Treemap layouting. In: Proc. EG EuroVis 2017 – Short Papers. pp. 79–83. EuroVis '17 (2017). https://doi.org/10.2312/eurovisshort.20171137
26. Harrower, M., Brewer, C.A.: Colorbrewer.org: An online tool for selecting colour schemes for maps. Taylor & Francis The Cartographic Journal **40**(1), 27–37 (2003). https://doi.org/10.1179/000870403235002042
27. Heer, J., Card, S.K., Landay, J.A.: Prefuse: A toolkit for interactive information visualization. In: Proc. ACM SIGCHI Conference on Human Factors in Computing Systems. pp. 421–430. CHI '05 (2005). https://doi.org/10.1145/1054972.1055031
28. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Transactions on Visualization and Computer Graphics **12**(5), 741–748 (2006). https://doi.org/10.1109/TVCG.2006.147
29. Jern, M., Rogstadius, J., Åström, T.: Treemaps and choropleth maps applied to regional hierarchical statistical data. In: Proc. IEEE International Conference Information Visualisation. pp. 403–410. iV '09 (2009). https://doi.org/10.1109/IV.2009.97
30. Johnson, B., Shneiderman, B.: Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In: Proc. IEEE Visualization. pp. 284–291. Visualization '91 (1991). https://doi.org/10.1109/VISUAL.1991.175815
31. Kehrer, J., Hauser, H.: Visualization and visual analysis of multifaceted scientific data: A survey. IEEE Transactions on Visualization and Computer Graphics **19**(3), 495–513 (2013). https://doi.org/10.1109/TVCG.2012.110
32. Koller, D., Turitzin, M., Levoy, M., Tarini, M., Croccia, G., Cignoni, P., Scopigno, R.: Protected interactive 3d graphics via remote rendering. ACM Transactions on Graphics **23**(3), 695–703 (2004). https://doi.org/10.1145/1015706.1015782
33. Limberger, D., Döllner, J.: Real-time rendering of high-quality effects using multi-frame sampling. In: Proc. ACM SIGGRAPH Posters. p. 79. SIGGRAPH '16 (2016). https://doi.org/10.1145/2945078.2945157
34. Limberger, D., Fiedler, C., Hahn, S., Trapp, M., Döllner, J.: Evaluation of sketchiness as a visual variable for 2.5d treemaps. In: Proc. IEEE International Conference Information Visualisation. pp. 183–189. iV '16 (2016). https://doi.org/10.1109/IV.2016.61
35. Limberger, D., Gropler, A., Buschmann, S., Döllner, J., Wasty, B.: OpenLL: an API for dynamic 2d and 3d labeling. In: Proc. IEEE International Conference on Information Visualization. pp. 175–181. iV '18 (2018). https://doi.org/10.1109/iV.2018.00039
36. Limberger, D., Pursche, M., Klimke, J., Döllner, J.: Progressive high-quality rendering for interactive information cartography using WebGL. In: ACM Proc. International Conference on 3D Web Technology. pp. 8:1–8:4. Web3D '17 (2017). https://doi.org/10.1145/3055624.3075951
37. Limberger, D., Scheibel, W., Hahn, S., Döllner, J.: Reducing visual complexity in software maps using importance-based aggregation of nodes. In: Proc. SciTePress International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP. pp. 176–185. VISIGRAPP/IVAPP '17 (2017). https://doi.org/10.5220/0006267501760185
38. Limberger, D., Scheibel, W., Lemme, S., Döllner, J.: Dynamic 2.5d treemaps using declarative 3d on the web. In: Proc. ACM International Conference on 3D Web Technology. pp. 33–36. Web3D '16 (2016). https://doi.org/10.1145/2945292.2945313
39. Limberger, D., Trapp, M., Döllner, J.: In-situ comparison for 2.5d treemaps. In: Proc. SciTePress International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP. pp. 314–321. VISIGRAPP/IVAPP '19 (2019). https://doi.org/10.5220/0007576203140321

40. Limberger, D., Wasty, B., Trümper, J., Döllner, J.: Interactive software maps for web-based source code analysis. In: Proc. ACM International Conference on 3D Web Technology. pp. 91–98. Web3D '13 (2013). https://doi.org/10.1145/2466533.2466550
41. Liu, S., Cao, N., Lv, H.: Interactive visual analysis of the NSF funding information. In: Proc. IEEE Pacific Visualization Symposium. pp. 183–190. PacificVis '08 (2008). https://doi.org/10.1109/PACIFICVIS.2008.4475475
42. Liu, S., Cui, W., Wu, Y., Liu, M.: A survey on information visualization: Recent advances and challenges. Springer The Visual Computer **30**(12), 1373–1393 (2014). https://doi.org/10.1007/s00371-013-0892-3
43. Nagamochi, H., Abe, Y.: An approximation algorithm for dissecting a rectangle into rectangles with specified areas. Elsevier Discrete Applied Mathematics **155**(4), 523–537 (2007). https://doi.org/j.dam.2006.08.005
44. Nguyen, Q.V., Huang, M.L.: Enccon: An approach to constructing interactive visualization of large hierarchical data. Palgrave Information Visualization **4**(1), 1–21 (2005). https://doi.org/10.1057/palgrave.ivs.9500087
45. Park, D., Drucker, S.M., Fernandez, R., Niklas, E.: ATOM: A grammar for unit visualizations. IEEE Transactions on Visualization and Computer Graphics **24**(12), 3032–3043 (2018). https://doi.org/10.1109/TVCG.2017.2785807
46. Richter, M., Söchting, M., Semmo, A., Döllner, J., Trapp, M.: Service-based Processing and Provisioning of Image-Abstraction Techniques. In: Proc. International Conference on Computer Graphics, Visualization and Computer Vision. pp. 79–106. WCSG '18 (2018). https://doi.org/10.24132/CSRN.2018.2802.13
47. Roberts, R.C., Laramee, R.S.: Visualising business data: A survey. MDPI Information **9**(11), 285;1–54 (2018). https://doi.org/10.3390/info9110285
48. dos Santos, S., Brodlie, K.: Gaining understanding of multivariate and multidimensional data through visualization. Elsevier Computers & Graphics **28**(3), 311–325 (2004). https://doi.org/10.1016/j.cag.2004.03.013
49. Satyanarayan, A., Moritz, D., Wongsuphasawat, K., Heer, J.: Vega-Lite: A grammar of interactive graphics. IEEE Transactions on Visualization and Computer Graphics **23**(1), 341–350 (2017). https://doi.org/10.1109/TVCG.2016.2599030
50. Scheibel, W., Buschmann, S., Trapp, M., Döllner, J.: Attributed vertex clouds. In: GPU Zen: Advanced Rendering Techniques, chap. Geometry Manipulation, pp. 3–21. Bowker Identifier Services (2017)
51. Scheibel, W., Hartmann, J., Döllner, J.: Design and implementation of web-based hierarchy visualization services. In: Proc. SciTePress International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP. pp. 141–152. VISIGRAPP/IVAPP '19 (2019). https://doi.org/10.5220/0007693201410152
52. Scheibel, W., Trapp, M., Döllner, J.: Interactive revision exploration using small multiples of software maps. In: Proc. SciTePress International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 2: IVAPP. pp. 131–138. VISIGRAPP/IVAPP '16 (2016). https://doi.org/10.5220/0005694401310138
53. Scheibel, W., Weyand, C., Döllner, J.: EvoCells – a treemap layout algorithm for evolving tree data. In: Proc. SciTePress International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP. pp. 273–280. VISIGRAPP/IVAPP '18 (2018). https://doi.org/10.5220/0006617102730280
54. Schulz, H.J.: Treevis.net: A tree visualization reference. IEEE Computer Graphics and Applications **31**(6), 11–15 (2011). https://doi.org/10.1109/MCG.2011.103

55. Schulz, H.J., Akbar, Z., Maurer, F.: A generative layout approach for rooted tree drawings. In: Proc. IEEE Pacific Visualization Symposium. pp. 225–232. PacificVis '13 (2013). https://doi.org/10.1109/PacificVis.2013.6596149
56. Schulz, H.J., Hadlak, S., Schumann, H.: Point-based visualization for large hierarchies. IEEE Transactions on Visualization and Computer Graphics **17**(5), 598–611 (2011). https://doi.org/10.1109/TVCG.2010.89
57. Schulz, H.J., Hadlak, S., Schumann, H.: The design space of implicit hierarchy visualization: A survey. IEEE Transactions on Visualization and Computer Graphics **17**(4), 393–411 (2011). https://doi.org/10.1109/TVCG.2010.79
58. Schulz, H.J., Schumann, H.: Visualizing graphs – a generalized view. In: Proc. IEEE International Conference on Information Visualization. pp. 166–173. IV '06 (2006). https://doi.org/10.1109/IV.2006.130
59. Slingsby, A., Dykes, J., Wood, J.: Using treemaps for variable selection in spatio-temporal visualisation. Palgrave Information Visualization **7**(3), 210–224 (2008). https://doi.org/10.1057/PALGRAVE.IVS.9500185
60. Slingsby, A., Dykes, J., Wood, J.: Configuring hierarchical layouts to address research questions. IEEE Transactions on Visualization and Computer Graphics **15**(6), 977–984 (2009). https://doi.org/10.1109/TVCG.2009.128
61. Soares, A.G., dos Santos, D.H., Barbosa, C.L., Goncalves, A.S., dos Santos, C.G., Meiguins, B.S., Miranda, E.T.: Visualizing multidimensional data in treemaps with adaptive glyphs. In: Proc. IEEE International Conference Information Visualisation. pp. 58–63. iV '18 (2018). https://doi.org/10.1109/iV.2018.00021
62. Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S., Slusallek, P.: Xml3d: Interactive 3d graphics for the web. In: Proc. ACM International Conference on 3D Web Technology. pp. 175–184. Web3D '10 (2010). https://doi.org/10.1145/1836049.1836076
63. Steinbrückner, F., Lewerentz, C.: Understanding software evolution with software cities. SAGE Information Visualization **12**(2), 200–216 (2013). https://doi.org/10.1177/1473871612438785
64. Stylos, J., Myers, B.: Mapping the space of api design decisions. In: Proc. IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 50–60. VL/HCC '07 (2007). https://doi.org/10.1109/VLHCC.2007.44
65. Tak, S., Cockburn, A.: Enhanced spatial stability with hilbert and moore treemaps. IEEE Transactions on Visualization and Computer Graphics **19**(1), 141–148 (2013). https://doi.org/10.1109/TVCG.2012.108
66. Tan, W., Fan, Y., Ghoneim, A., Hossain, M.A., Dustdar, S.: From the service-oriented architecture to the web api economy. IEEE Internet Computing **20**(4), 64–68 (2016). https://doi.org/10.1109/MIC.2016.74
67. Tu, Y., Shen, H.: Visualizing changes of hierarchical data using treemaps. IEEE Transactions on Visualization and Computer Graphics **13**(6), 1286–1293 (2008). https://doi.org/10.1109/TVCG.2007.70529
68. Veras, R., Collins, C.: Optimizing hierarchical visualizations with the minimum description length principle. IEEE Transactions on Visualization and Computer Graphics **23**(1), 631–640 (2017). https://doi.org/10.1109/TVCG.2016.2598591
69. Vernier, E.F., Telea, A.C., Comba, J.: Quantitative comparison of dynamic treemaps for software evolution visualization. In: Proc. IEEE Working Conference on Software Visualization. pp. 96–106. VISSOFT '18 (2018). https://doi.org/10.1109/VISSOFT.2018.00018
70. Webber, J., Parastatidis, S., Robinson, I.: REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media, 1st edn. (2010)

71. Wettel, R., Lanza, M.: Visual exploration of large-scale system evolution. In: Proc. IEEE Working Conference on Reverse Engineering. pp. 219–228. WCRE '08 (2008). https://doi.org/10.1109/WCRE.2008.55
72. Wood, J., Brodlie, K., Seo, J., Duke, D., Walton, J.: A web services architecture for visualization. In: Proc. IEEE Internation Conference on eScience. pp. 1–7. eScience '08 (2008). https://doi.org/10.1109/eScience.2008.51
73. Wood, J., Isenberg, P., Isenberg, T., Dykes, J., Boukhelifa, N., Slingsby, A.: Sketchy rendering for information visualization. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2749–2758 (2012). https://doi.org/10.1109/TVCG.2012.262
74. Würfel, H., Trapp, M., Limberger, D., Döllner, J.: Natural phenomena as metaphors for visualization of trend data in interactive software maps. In: Proc. EG Computer Graphics and Visual Computing. CGVC '15 (2015). https://doi.org/10.2312/cgvc.20151246