Towards Interactive Geovisualization Authoring Toolkit for Industry Use Cases

Jiří Hynek[1](✉) and Vít Rusňák[2]

[1] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
hynek@fit.vut.cz

[2] Institute of Computer Science, Masaryk University, Brno, Czech Republic
rusnak@ics.muni.cz

Abstract. Interactive visualizations of geospatial data are commonplace in various applications and tools. The visual complexity of these visualizations ranges from simple point markers placed on the cartographic maps through visualizing connections, heatmaps, or choropleths to their combination. Designing proper visualizations of geospatial data is often tricky, and the existing approaches either provide only limited support based on pre-defined templates or require extensive programming skills. In our previous work, we introduced the Geovisto toolkit – a novel approach that blends between template editing and programmatic approaches providing tools for authoring reusable multilayered map widgets even for non-programmers. In this paper, we extend our previous work focusing on Geovisto's application in the industry. Based on the critical assessment of two existing usage scenarios, we summarize the necessary design changes and their impact on the toolkit's architecture and implementation. We further present a case study where Geovisto was used in the production-ready application for IoT sensor monitoring developed by Logimic, a Czech-US startup company. We conclude by discussing the advantages and limitations of our approach and outlining the future work.

Keywords: Geospatial data · Geovisualizations · Visual Authoring tools

1 Introduction

Interactive geovisualizations are used in various use cases, ranging from simple choropleths in newspaper articles to specialized analytical applications for disaster management [9] or ornitology [19]. The underlying *geospatial data* can combine location information, descriptive attributes (categorical, numerical, or even textual), and optionally also temporal dimensions (e.g., timestamp, duration). The interactive geovisualizations can display data in multiple layers and enable users to explore them at different detail levels through zooming and panning. The right choice of visual geospatial data representation is not always straightforward. When done wrong, it might lead to obscuring data perspectives that are essential

© Springer Nature Switzerland AG 2023

for the users [3]. Moreover, creating efficient interactive geovisualizations usually requires programming skills.

In the last decade, we can observe growing efforts toward developing visualization authoring systems enabling the creation of such interactive visualizations for non-programmers [7,13]. Shortcomings of these tools include the focus on 2D charts, only a few types of available geovisualizations, and limited capabilities in terms of input data or support for multi-layered geovisualization interaction.

Our work aims to provide the geovisualization authoring toolkit, which has an extensible API for programmers and allows authoring various use cases through a user-friendly interface for users without programming skills. In this paper, we: a) summarize the feedback acquired from the two usage scenarios in which the early Geovisto prototype introduced in [10]; b) introduce the architectural and implementation changes of the redesigned version; c) present a novel use case of Geovisto in the production-ready application for IoT sensor monitoring.

The paper is structured as follows. Section 2 overviews the related work and summarizes the shortcomings of current geovisualization authoring approaches. Section 3 summarizes the Geovisto prototype and the two use cases described in detail in our previous work [10]. Section 4 presents the design requirements revisions and proposed changes to Geovisto's architecture and implementation. Sections 5 and 6 show the novel architecture of the toolkit and the implementation respectively. The case study of Geovisto's application in the context of IoT sensor monitoring is in Sect. 7. Finally, Sect. 8 discusses the advantages and disadvantages of our approach, and Sect. 9 summarizes the paper and outlines future work.

2 Related Work

In this section, we first overview several widely-used 2D geovisualization types. Next, we present the *visualization authoring methods* with a particular focus on authoring tools and their limitations.

2.1 Geovisualization Types

Geovisualizations often take advantage of combining multiple layers where each layer presents only a subset of data. The typical base layer is a cartographic map that provides a spatial context for visualized data. The geographic information systems (e.g., QGIS[1] or ArcGIS[2] and the web mapping applications available to the general public (e.g., Google Maps, Bing Maps, Open Street Maps) provide the cartographic layers through public APIs, which can be used in the 3rd party applications.

Further, we list the most common geospatial data visualization types. *Point distribution maps* represent the simplest geovisualizations used for visualizing datasets of elements containing only the location information (1 a)). If the

[1] https://qgis.org/en/site/.
[2] https://www.arcgis.com/index.html.

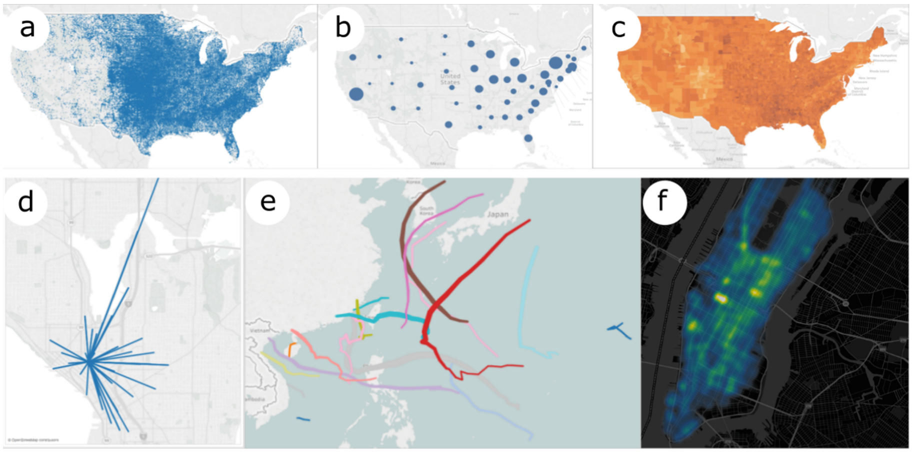


Fig. 1. Common geovisualization types: a) point distribution maps; b) proportional symbol map; c) choropleth; d) spider map; e) flow map; f) heatmap. Source: Tableau Software LLC.

elements contain the location and one descriptive numerical value, they could be plotted as *(proportional) symbol maps* (1 b)). The symbol can be a circle or a glyph whose physical size determines the value. Symbol maps are also useful when the element has two or three descriptive attributes since we can distinguish their size, shape, and color. More advanced modifications enable us to show even more data, e.g., when the glyph symbol is replaced by small 2D charts (e.g., pie/donut/bar charts). When the dataset contains information, not for single locations but the whole regions, *choropleth maps* (or filled maps) are the best option. The color fill of the region represents value (1 c)). Another type of spatial visualization is *heatmaps* (or density maps) that are common, e.g., from weather maps showing the measured temperature or precipitation (1 f)). A particular category is geovisualizations showing routes and paths. Well-known from navigations, the basic *path maps* show direction between two points. However, in the case of flight monitoring websites or computer network visualizations, we can naturally extend the point-to-point to multipoint connections, also known as *spider maps* (1 d)). Finally, by adding the temporal dimension, we can visualize *flow maps*, enabling traffic visualization on the edges (1 e)).

2.2 Geovisualization Authoring Approaches

The interactive visualizations can be authored in three general ways: by programming, template editing, or authoring tools and applications.

Programmatic approaches are the most demanding in terms of users' skills and learning curve but offer the most versatility for fine-tuning of visual appearance and interaction capabilities. The frameworks for developing interactive visualizations are usually designed for web applications. D3 [2] is one of the most popular imperative frameworks nowadays, and many other libraries use it. It allows mapping the input data to a Document Object Model (DOM) and transforms it via data-driven operations. ProtoVis [1] toolkit, also leveraging the imperative paradigm, is based on the idea of decomposing visualizations into hierarchies of visual primitives whose visual properties are data functions. The declarative paradigm frameworks represent Vega [17] and Vega-lite [16]. They both provide a set of building blocks for interactive visualization designs. They differ in the level of abstraction and primary use cases. Vega-lite is a high-level grammar built on top of Vega and was designed for prototyping standard chart types rapidly. Backward compatibility allows programmers to implement more advanced use cases in Vega. There are also several widely-used geovisualization libraries for the web development such as Mapbox GLJS[3], OpenLayers[4] or Leaflet[5]. The latter one we utilize in Geovisto.

Template editing is the exact opposite of programmatic approaches. It is a well-established way to create simple charts in spreadsheet applications like Microsoft Excel or Apache OpenOffice. The main characteristics are limited functionality in terms of interaction and the ability to visualize tabular-based data in a pre-defined set of charts (e.g., pie charts, bar charts, or choropleths). Users can modify only a basic set of parameters such as color, font, chart shape, or legend position. Template editing is also available in dashboard platforms like Grafana [6]), data analysis tools such as Tableau [18], or analytical frameworks such as ElasticSearch in the form of extension library [4]. They allow the users to connect their dataset through API. On the other hand, their main disadvantage is that they centralize their platform's visualization with limited support for their export.

The *authoring tools* build on the advantages of the former two. We can imagine them as advanced graphics software focusing on designing interactive charts. They allow users to create visualizations from basic building blocks that can be widely customized in visual appearance and interaction capabilities through GUI. The output visualizations can be exported as web components and published still without programming skills. The visualization community introduced several projects in the last couple of years. Lyra [15], Data Illustrator [12], or Charticulator [14] represents such tools or systems. However, their primary focus is on authoring 2D charts, and geospatial data visualization is often limited to data presentation in a single layer. Another downside is that the tools require specific visual design knowledge, limiting some users. There are also examples of domain-specific visualization authoring applications. For example, NewsViews [5] targets data journalists to help them create interactive geovisualizations for online news articles. GeoDa Web [11] platform leverages the cloud storage and computing capabilities and enables data analysts to visualize and publish maps and plots to social media in a user-friendly way. Unlike the general visualization authoring systems, the domain-specific ones are simpler and reduce the need for specific visual design knowledge.

[3] https://www.mapbox.com/mapbox-gljs.
[4] https://openlayers.org.
[5] https://leafletjs.com.

2.3 Limitations of Current Authoring Tools

We aim to generalize the geovisualization authoring tool while focusing on ease of use for professional and novice users. In general, we identified three limitations of the current tools that we address in our work.

Tabular data as the primary input format. Most of the tools expect the data in a tabular format (e.g., CSV), where columns are attributes (or domains) of elements in rows. However, many of the recent data sets are in hierarchical object formats such as JSON or NoSQL databases. For these, additional data transformation or preprocessing is necessary before their use in visualizations. Our goal is to allow users to upload arbitrary geospatial data in an object-oriented format and select the visualization attributes.

Limited number of configuration options. Since the existing tools focus mainly on general 2D chart visualizations, the list of available geovisualization types and their configuration options are narrow. The most frequent are choropleths, heatmaps, or spider maps. As a result, the user can often display only a few data attributes. Our goal is to enable a combination of visualization types in multiple layers and let users decide which suits their needs.

Limited interaction capabilities. Finally, current tools provide only limited interaction capabilities with visualized geospatial data such as their filtering or region-based selection. Our goal is to let users configure the output geovisualization in line with the expected usage and allow them to set multi-layer interaction capabilities and cross-layer data linkage.

We propose Geovisto – the geovisualization authoring toolkit, which enables configuring geospatial data visualizations for use in web-based dashboard applications or as a part of visual analytics workflows. In the reminder, we present its design and prototype implementation. Two usage scenarios demonstrate its applicability.

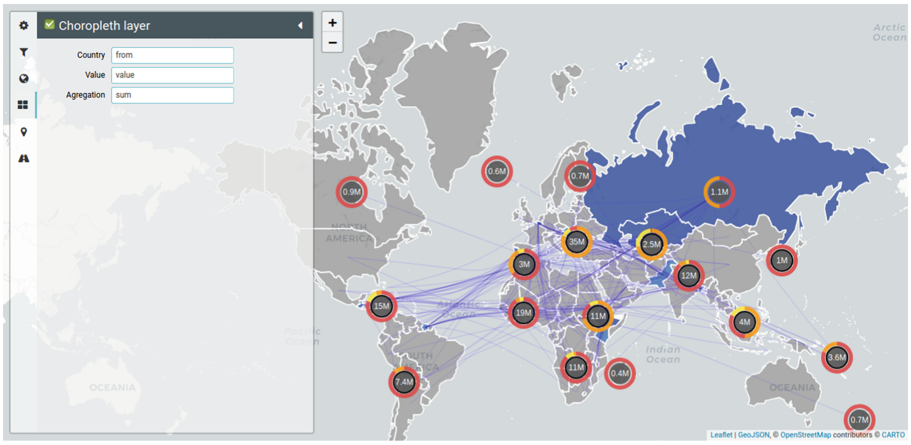
3 Geovisto Prototype

In this section, we outline the Geovisto prototype and two usage scenarios described in detail in our earlier paper [10]. The usage scenarios that were used to demonstrate Geovisto's features also served for further revision of the requirements.

Geovisto prototype is a standalone ReactJS[6] component, using Leaflet and D3.js JavaScript libraries. Thus, it can be shared and included as a widget in third-party web applications. The implementation is based on the four design requirements: a) it has a component-based *architecture*; b) the input data are transformed to a flat *data model*; c) its *user interface* enable user-defined data mapping to multiple configurable layers; d) users can export and import *map configurations*.

Geovisto renders the base map with one or more predefined layers when loaded. Users can specify their dataset and import or export map configurations. Available layers are choropleth, marker, and connection layer. The layers

[6] https://reactjs.org/.


Choropleth layer

Country: from

Value: value

Agregation: sum

Fig. 2. An example of the Geovisto prototype map widget. It contains the sidebar (on the left) used for the configuration of layers, the definition of filter rules, and the map's general setting. The map contains the choropleth, marker, and connection layers. The example shows the configuration of the choropleth layer. It links the 'from' data domain with the 'country' visual dimension, the 'value' data domain with the 'value' visual dimension, and uses the 'sum' function to aggregate the values.

can be managed from the user interface (e.g., show/hide the layer, define the data visualized in each layer, apply basic filtering on the visualized data). The Geovisto prototype also provides functions for projecting geographic features onto the map and interaction capabilities with the base map. It handles events invoked by the map layers (user interaction) and ships them to other layers, which can process them (since the map layers are independent). Figure 2 shows an example Geovisto prototype map widget.

We further demonstrated the Geovisto prototype on two distinct usage scenarios: Covid-19 pandemic open data, and DDoS attack analysis.

Covid-19 Pandemic Open Data: We used the map to visualize the worldwide spread of the COVID-19 disease to demonstrate the widget's general applicability. We used the data from the rapidapi.com[7] service and converted them to the JSON format to import them into the map widget. Since the map allows the users to change the data domains, the users can compare the countries from different aspects (sum of confirmed cases, numbers of recoveries, and deaths). They can also combine these views using the choropleth and marker layers. Figure 3 shows an example of the use case.

[7] https://rapidapi.com/.

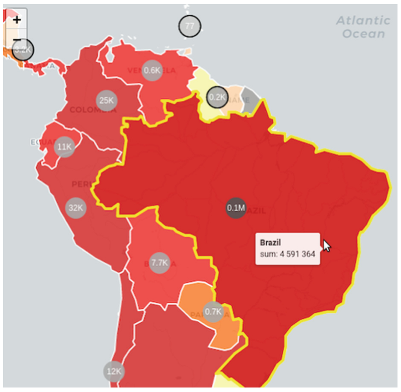


Fig. 3. Covid-19 pandemic data. The choropleth layer compares the number of confirmed cases with the disease. The marker layer shows the number of deaths caused by the disease.

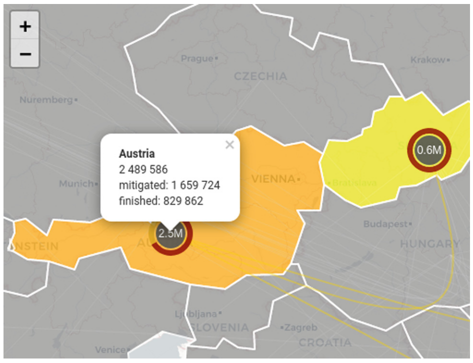


Fig. 4. DDoS attack analysis usage scenario. The example shows the share of mitigated and finished attacks for selected countries.

DDoS Attack Analysis: The scenario was performed in cooperation with Flowmon Networks a.s.[8], a company providing complex tools for automated monitoring, analysis, and network traffic protection. Figure 4 shows a DDoS monitoring component. It provides an overview of all DDoS attacks and the source and destination countries. The connections between the countries visualize the relations of the traffic flows. A country's details show specific aspects of the attacks, such as the state of the attacks (active, mitigated, finished) and their numbers (in pop-up windows). The multilayer map meets the requirements. It contains either the information about the source of DDoS attacks or their destinations. Then, the connection layer can display the relations between the countries. The users can filter the data to display only a specific subset. Finally, they can select a particular country in the choropleth and highlight all the related data presented in the same or other map layers.

While the former usage scenario focused on demonstrating Geovisto's features and general applicability, the latter demonstrated its utilization in the real-world example from the cybersecurity domain. We further analyzed and evaluated the usage scenarios to revise the design requirements and identify the shortcomings limiting Geovisto's applications in the industry.

4 Design Requirements Revision

The main area of Geovisto's deployment is industrial applications. In order to meet the requirements of the industry, we reformulated the initial design requirements based on the Geovisto prototype assessment in the presented scenarios. The revised requirements focus on five main aspects:

[8] https://www.flowmon.com/en.

- ***Usability***: Geovisto should utilize the concept of authoring systems and provide map layers representing ready-to-use thematic maps to decrease the effort to prototype visual projections of geographical data quickly;
- ***Modularity***: Geovisto should not be served as a monolith but in the form of independent modules which would provide particular functionality such as map layers and map controls; these modules could be used only when needed in order to decrease the size of the result;
- ***Configurability***: Geovisto's basemap and its layers and controls might be customized with generic datasets, geographical data, appearance, and behavior; users might capture the current map state and reload this state later;
- ***Extensibility***: Geovisto should provide a core with programming API to customize the map programmatically, allow further extensions of the existing modules of the library, or implement new own modules concerning the current requirements;
- ***Accessibility***: the library (and modules of the library) should be available through a known package manager software to support versioning and improve integration into its own system infrastructure, build and deliver the solution as a part of its system.

The following subsections list these aspects reflecting the initial prototype and identify required modifications to the original prototype design that have driven the re-implementation of the Geovisto toolkit.

4.1 Usability

One of the crucial problems during the design and implementation of user interface and visualization is to overcome the communication barrier between customers providing requirements and programmers delivering the final product for them. Usually, salespeople, UX designers, or data analysts try to break this barrier. However, their lower knowledge of the system architecture and underlying data models might skew the description of the requirements to the programmers. These specifications are usually vague, and they do not consider actual data representation (e.g., data types and relations between data entities) and effort (e.g., complexity and price of database queries) to map the data into the UI components. The main goal of the Geovisto toolkit is to blend the programmatic and template editing approaches known from contemporary mapping libraries to improve user experience during the prototyping phase.

The idea of Geovisto is to provide a UI layer composed of ready-to-use map layers and controls, which would allow the UI designer to project the actual data and prototype the map views corresponding to the end-user requirements. Then, a snapshot of the map state could be exported and shared in a serialized format. Such a configuration might help the programmer who implements a new map widget into the production version of the application or service used by the end-users, as illustrated in Fig. 5.

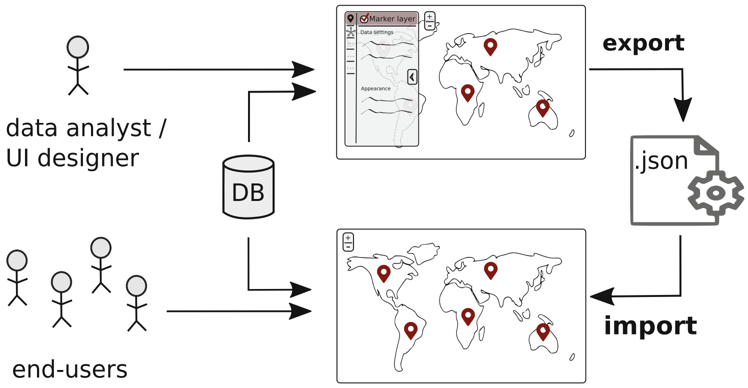


Fig. 5. Geovisto's authoring and configuration sharing workflow. First, the UI designer creates a data projection into the map layers and exports the configuration into a JSON file. The configuration can be shared with end-users or programmers developing the web front-end.

The authoring tools could have been used while clarifying the user requirements and possible use cases of the Flowmon UX team. Since Geovisto can work with custom datasets, the UX team members used the prototype independently, generating multiple map configurations providing perspectives of their custom data without any coding knowledge. This approach improved mutual communication between them and programmers and rapidly increased the ability to generate new geovisualization use cases.

While the Geovisto prototype provided decent authoring tools to prepare the map drafts, export, and import the map state, the programmatic definition of the state as properties of map and map layers was rather cumbersome. Since the Geovisto prototype was implemented in JavaScript, the property types were unclear, leading to occasional crashes and debugging requests. One of the most needed requirements was to re-implement the project into TypeScript[9], which only emphasizes the importance of statically typed languages in the industry and even the front-end and data visualization.

4.2 Modularity

When implementing the prototype, our initial goal was to design a clear code structure and decompose the library into the so-called *Geovisto tools* representing particular map layers and controls and the map core handling communication between the tools. Although we fulfilled the modular approach, the main drawback stood for how the library was delivered – in the form of one JavaScript repository. Thus, programmers using Geovisto needed to load the whole library,

[9] https://www.typescriptlang.org.

although they wanted to use only a subset of Geovisto's tools. To fully accomplish the proposed modularity, it was necessary to break apart the repository into standalone packages to be included as project dependencies when needed. The crucial task to handle this was to solve the problem of possible dependencies between the tools (e.g., one tool needed to know the type of event object sent by another tool).

4.3 Configurability

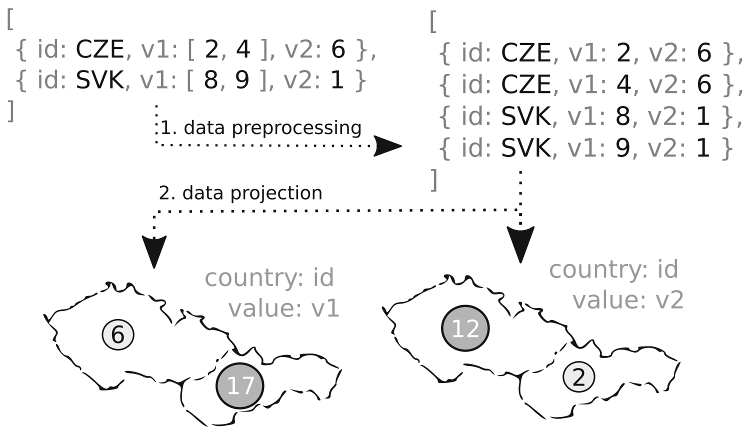
The Geovisto prototype worked with the following types of inputs:

- **Geographical Data:** the specification of polygons and their centroids represented in GeoJSON format. The prototype used the specification of world countries published by J. Sundstrom[10] but it should be replaceable with generic specifications. The only requirement was that every GeoJSON feature had to contain a polygon identifier (e.g., country code), which is needed to connect the geographical data with the dataset.
- **Datasets:** the values stored in a serialized format (JSON). The data needs to contain at least one data domain representing an identifier of the geographical feature (e.g., country code).

Since there are various models representing the data, it was essential to create a mechanism for processing such models – choosing the data domains and binding them to the map layers' visual dimensions. Hence, every map layer provides a list of visual dimensions, which can be associated with data domains. The users can select the data domains manually using the layer settings provided by the map sidebar. In contrast to existing authoring tools, this approach focuses only on geospatial data. Users can work with multiple data domains representing geographic location formats (such as the ISO 3166 country codes) and use them in various use cases.

Since the dataset can represent non-tabular data structures (e.g., JSON or XML formats), recursive data preprocessing was needed to construct a valid data model representing data domains. Only then, the list of the data domains was served to the UI. Figure 6 shows an example demonstrating the principle. The Geovisto prototype provides a basic flattening algorithm that expands all nested arrays into a combination of flat data visual projections and aggregations.

[10] https://github.com/johan/world.geo.json.



```
[
 { id: CZE, v1: [ 2, 4 ], v2: 6 },
 { id: SVK, v1: [ 8, 9 ], v2: 1 }
]
```

```
[
 { id: CZE, v1: 2, v2: 6 },
 { id: CZE, v1: 4, v2: 6 },
 { id: SVK, v1: 8, v2: 1 },
 { id: SVK, v1: 9, v2: 1 }
]
```

Fig. 6. An example of data composed of two records stored in the pseudo-JSON format. Since the records contain nested lists, they need to be preprocessed first. They are expanded into four records represented by all combinations of the values. This representation is characterized by data domains that can be mapped into visual dimensions. The figure shows two different projections and aggregation of data.

Another requirement from the Flowmon evaluation was to implement different flattening strategies. For instance, some of the nested lists might represent specific qualitative characteristics of network traffic which should not be preprocessed as described in the example of Fig. 6. For this purpose, the Geovisto prototype lacked a solid mechanism to deliver custom data preprocessing algorithms.

Similarly, overriding the default Geovisto behavior and settings was cumbersome and required decent knowledge of the library architecture. For instance, Flowmon needed to integrate the map instances into its own environment, which is characterized by specific appearance (e.g., types of controls familiar to the company's customers). Hence, the requirement to completely redesign the map layers and controls has a high priority for future deployment of the map instances to corporate environments.

4.4 Extensibility

Another requirement from the Flowmon evaluation was to implement different flattening strategies. For instance, some of the nested lists might represent specific qualitative characteristics of network traffic which should not be preprocessed as described in the example of Fig. 6. For this purpose, the Geovisto prototype lacked a solid mechanism to deliver custom data preprocessing mechanisms.

Similarly, overriding the default Geovisto behavior and settings was cumbersome and required decent knowledge of the library architecture. For instance, Flowmon needed to integrate the map instances into its environment, characterized by specific appearance (e.g., types of controls familiar to its customers). Hence, the requirement to completely redesign the map layers and controls has a high priority for future deployment of the map instances to corporate environments.

4.5 Accessibility

Since our goal is to deliver Geovisto in compact modules, it is essential to maintain modules versioning. These modules should be available in package management systems, such as `npm` (Node.js Package Manager), and the modules users (i.e., programmers) should decide on their own when to switch to a higher version. It is another essential requirement in the industry when delivering new product releases. In order to deliver a stable product to its customers, the dependent libraries must be reliable.

Flowmon develops its user interfaces in the React.js library. It helps organize the user interface into logical parts (React components) and manage its rendering lifecycle and UI events. The Geovisto prototype was wrapped in the React component, but we decided to leave this abstraction in future versions and provide the React extension as a standalone package. We kept Geovisto as a Leaflet-based TypeScript library that can be integrated into any web application by the companies which might use different UI frameworks (such as Angular or Vue.js).

5 Architecture

We updated the Geovisto prototype's architecture concerning the listed requirements, which decomposes the library into the map core and map layers. We designed a revised architecture reflecting the new aspects of Geovisto. The idea of new Geovisto architecture is similar to the old one. However, it generalizes the Geovisto modules as so-called *Geovisto Tools*, representing map layers, map controls (e.g., sidebar), or utilities (e.g., filters and themes). Map layers are a particular type of tool (Fig. 7).

The reason behind the decomposition was to provide standalone `npm` packages that can be included in a project when needed. Every *Geovisto Tool* is a TypeScript project with a package.json file containing the `npm` metadata[11]. It contains two peer dependencies – *Geovisto Core* and Leaflet – which force the programmer to include *Geovisto Core* and Leaflet library in own project. Hence, the built packages of *Geovisto Tools* do not include the code of these libraries in order to minimize the size of the packages and prevent conflicts in dependencies.

5.1 Core

The management of the *Geovisto Tools'* life cycle, inputs, and map state is provided by the *Geovisto Core*. It is an `npm` package[12] which represents an abstraction of the Leaflet library, and it needs to be (together with Leaflet) included in every project which utilizes *Geovisto Tools*.

[11] Metadata required by the Node.js Package Manager when resolving the tree of package dependencies, running, building, and publishing the package.
[12] https://www.npmjs.com/package/geovisto.

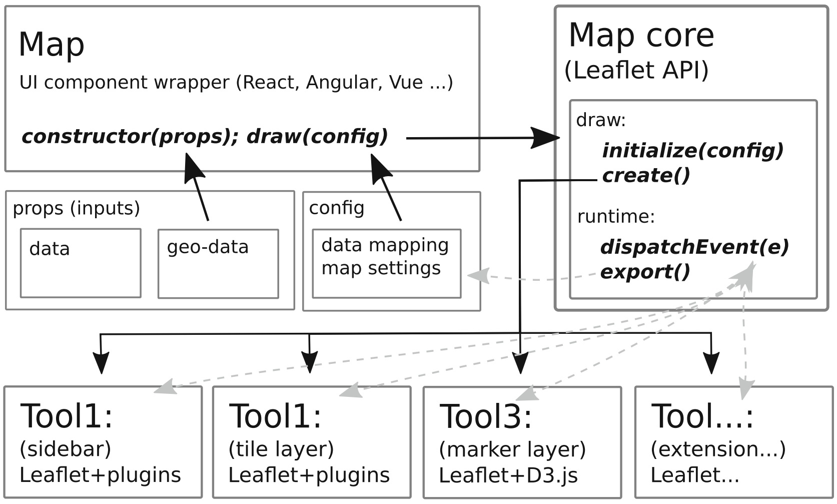


Fig. 7. Geovisto architecture overview. The map component takes props and config and renders a Leaflet-based map composed of map tools – usually SVG elements generated via the Leaflet API or D3.js library. The map tools are independent of each other and communicate via events. They represent map layers, map controls, and utilities.

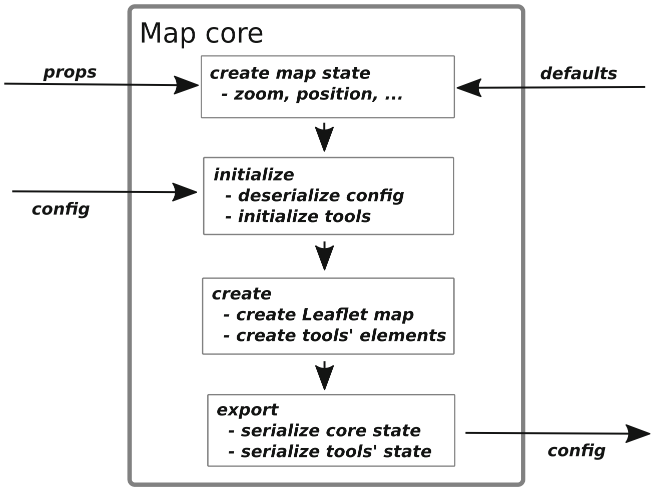


Fig. 8. Map core life-cycle. First, the map state is initialized by default values of the Geovisto toolkit and props given by the programmer. Then, the user can override the state using config (such as data mapping or appearance). The map and tools are rendered based on the combination of values given by Leaflet, programmer, and user. Finally, the user can use the map, change the state and export the config.

Figure 8 describes the workflow of the *Geovisto Core*. First, it processes the inputs and initializes the state of the map and instances of the required *Geovisto*

Tools. Then, based on the initial state, it creates and renders the Leaflet map and instances of *Geovisto Tools*. The phase of state initialization is crucial, and it determines the data projection and appearance of the map and tools. It is based on the following inputs:

- **Default Values (*defaults*)** – the state defined by Geovisto: the implicit values and functions representing the map's default behavior and appearance. They make the initial state of the map and the tools.
- **Properties (*props*)** – the state defined by the programmer: the default state can be overridden programmatically by using props. The programmer might influence either the map's appearance (static values) or override the default behavior (functions). Some of the props are optional (e.g., initial map zoom); however, there are several important mandatory props that are essential to render the map:
  - *Geo-data manager* providing geographical data (available GeoJSON definitions), which can be used across the tools.
  - *Data manager* providing dataset and strategy for data preprocessing into a suitable list data records which can be used across the tools.
  - *Tool manager* providing tool instances that should extend the *Geovisto Core*. Every tool might accept its own props and config to override its default state.
- **Configuration (*config*)** – the state defined by the user: the ability of the user to override default values and properties defined by the programmer. Config is defined by serialized format data (e.g., JSON) and manager, which processes this format and transforms it to the *Geovisto Core* model and tools. It can also be exported during the map runtime to capture the snapshot of the current map state. In contrast to the props, it represents only the static characteristics of the map, not the procedural characteristics of the map. Config usually defines map appearance and data mapping (projections of data domains to the dimensions of map layers).

All of the inputs are wrapped in so-called *input managers*, which process the inputs and transform them into Geovisto models. This approach gives the programmer the opportunity to work on their own input format and provide their own strategies to process these formats. It makes the library more generic and applicable in different environments.

5.2 API

In contrast to the Geovisto prototype, the new Geovisto version (Core and selected tools) was reimplemented using the TypeScript language. Using static types allowed us to describe the exact model of all map objects. The code structure was split into two parts representing:

- **Types and Interfaces:** declaration of all Geovisto objects, their properties, and methods (e.g., map, tool, layer, state, defaults, props, config, events, geo-data, data domains, layer dimensions, aggregation functions, filters, etc.)

- **Classes:** internal default definitions of map objects which provide implicit behavior, ready to use

Both the declarations and definitions of Geovisto objects are exported using ES6 module exports, so they can be used to design new Geovisto objects or extend the existing ones. In order to allow comfortable overriding of implicit map objects, the Factory design pattern was applied. Besides that, the programmer can approach low-level Leaflet objects and utilize all capabilities of this library.

The architecture of *Geovisto Core* is used in *Geovisto Tools* implementing the interfaces and types and extend the classes of *Geovisto Core API*. Since the tools might also be extended or used by each other, they export their own declarations and definitions of tool objects, similarly to the *Geovisto Core*. Then, the tools might communicate with each other using events and the Observer design pattern and the types of event objects are be known by the observers (e.g., the change of preferred style in the themes tool).

The tools are represented as standalone `npm` packages. In order to avoid direct dependencies between the `npm` packages, only the types and interfaces can be imported, and so-called devDependecies[13] are used.

6 Implementation

The *Geovisto Core* is distributed along with eight already published *Geovisto Tools* in the `npm` repository[14] under the MIT license. The source codes are available on Github[15]. Several other tools are in the development. Further, we present the published ones.

6.1 Layers

Layer tools represent thematic map layers. Besides *Tile layer*, each of the following layers allows defining a GeoJSON describing geographical objects of the layer (e.g., polygons, or points based on the type of thematic map). In contrast to the prototype, Geovisto accepts multiple definitions of GeoJSON directly in the props. Then, a data mapping needs to be set to connect the data domains with the layer's dimensions (e.g., geographical dimension, value dimension, or aggregation function) as illustrated in Fig. 2. The following map layer tools are already available in the `npm` repository.

- **Tile Layer Tool** represents the base map layer which utilizes Leaflet Tile layer API to show underlying maps of existing tile providers[16]. This might be required when the data needs to be connected with real geographical places.
- **Choropleth Layer Tool** allows to use GeoJSON specifications of polygons representing geographic regions and link them with the data. Unlike basic choropleth widgets, our implementation can process custom definitions of

[13] https://nodejs.dev/learn/npm-dependencies-and-devdependencies.
[14] https://www.npmjs.com/search?q=geovisto.
[15] https://github.com/geovisto.
[16] https://github.com/leaflet-extras/leaflet-providers.

geographic areas. Primarily, we work with the specification of world countries. However, different GeoJSON files can be used, as described in Sect. 5. The advantage of this approach is the higher scalability of the layer. We can use the layer in different situations and detail (e.g., countries, districts, custom areas). We can also adjust it according to the foreign policy of specific countries (e.g., visualization of disputed territories).

- **Marker Layer Tool** works with GeoJSON specification of points and visualizes the data related to exact geographic locations via *markers*. Similar to the choropleth polygons, every marker has a unique identifier and geographic position (e.g., country code and country centroid). Since marker visualization could be problematic when many are close to each other (clutter of markers), we use Leaflet.markercluster plugin[17] to overcome this issue by clustering the close markers into groups and aggregating the values.
- **Connection Layer Tool** visualizes relations between geospatial locations in the form of edges. The layer enables the user to select two required dimensions: *from* and *to*, representing nodes of the rendered edges (by default, we work with the country centroids identified by country codes). Optionally, the user can set the *value*, which affects the strength of the lines.

A common problem of connection maps is their complexity and poor edge placement. Holten and Van Wijk [8] proposed a force-directed edge bundling rendering technique that significantly reduces the clutter of edges. S. Engle demonstrated its application[18] on a flight map in the US. The example implements the technique using the d3-force[19] module of the D3.js library, which "implements a velocity Verlet numerical integrator for simulating physical forces on particles." In Geovisto, we implemented an SVG overlay layer using the Leaflet API and rendered the SVG elements representing edges using the D3.js library and the d3-force module according to Engle's approach. It was necessary to implement correct projections of the SVG elements into the Leaflet map concerning the map's zoom and current position. The result provides a comprehensive view of edges that can be zoomed in/out.

6.2 Controls and Utilities

The second type of tool adds additional controls and functionality to the map layers. Currently, Geovisto provides the support for adding custom UI controls in the sidebar, filtering and selection of data, and changing style themes:

- **Sidebar Tool** provides a collapsible sidebar control and the API allowing other tools to add custom sidebar tabs or sidebar tab fragments. The map layer tools utilize this to provide controls for their customization and changing data mapping.
- **Filters Tool** provides either UI controls to filter visualized data records and the API to define custom advanced filter operations. The users specify filter rules as conditional expressions evaluating selected data domains' values.

[17] https://github.com/Leaflet/Leaflet.markercluster.

[18] https://bl.ocks.org/sjengle/2e58e83685f6d854aa40c7bc546aeb24.

[19] https://github.com/d3/d3-force.

– **Selection Tool** provides a mechanism for connecting map layers with selected map layer geographical objects. The communication between the layers is implemented via the observer design pattern. Every event passed to the layers contains information about the source element selected by the user. It consists of the identifier of the geographic element (e.g., country code) and the layer. The identifiers of geographic elements can be used in more than one layer (e.g., choropleth country, country marker, connection node). Then, the filtering is based on the search of these identifiers through the map layer elements. The search algorithm avoids the cyclic event invocation. The elements found on the map are highlighted (Fig. 9).
– **Themes Tool** provides a set of predefined styles (e.g., colors), which are delivered to other tools via events and API, which allows defining own custom style themes.

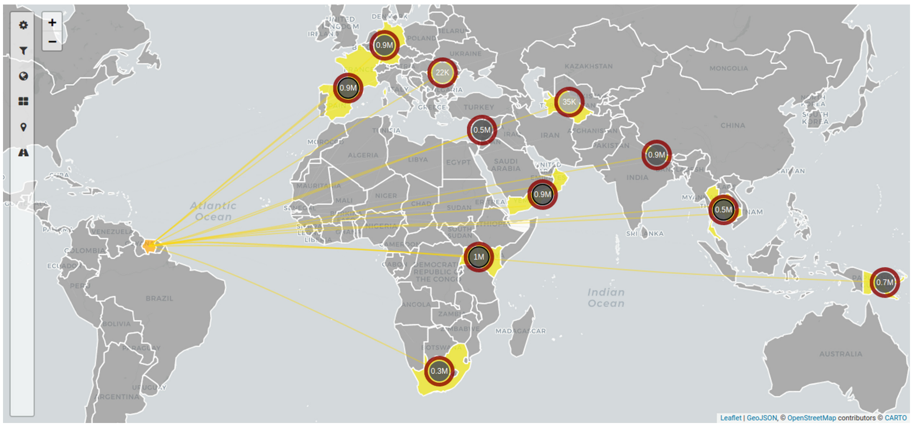


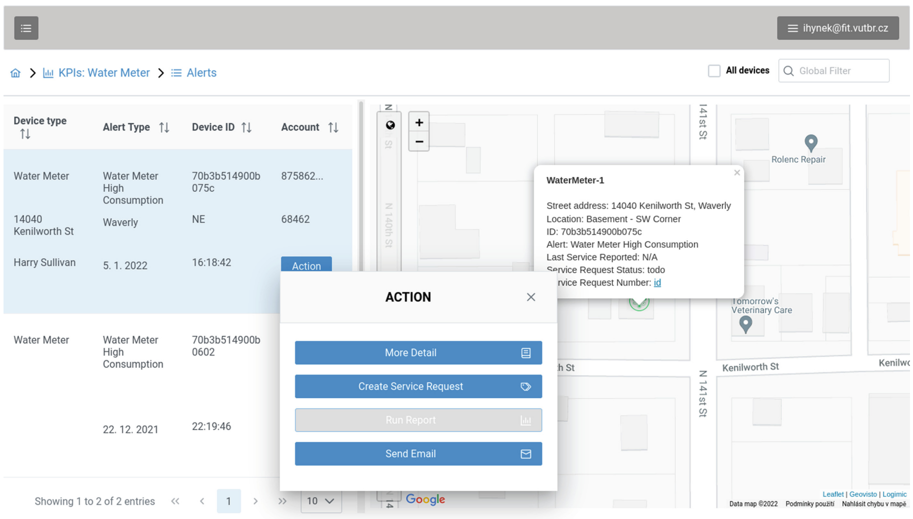
Fig. 9. Geographic element selection. The selection of Suriname in the choropleth layer invokes an event that is passed to other map layers. The connection layer handles the event, finds, and highlights all the edges which connect Suriname with other countries. This selection invokes another event which contains all the countries connected with Suriname. It affects the choropleth and marker layer, which highlights appropriate countries. Further invocations of events are stopped.

7 Case Study: Logimic

We demonstrated Geovisto's applicability in a case study that was done in cooperation with Logimic – the Czech-US startup company that brings innovative IoT solutions to industry.[20]

[20] Logimics' products include smart city dashboards for monitoring billions of sensors, street lighting control systems, indoor monitoring of temperature and humidity with small battery-operated wireless sensors, wireless control of industrial heaters, and many others (https://www.logimic.com/).

One of Logimic's products is a user-friendly web application implemented in TypeScript and Angular framework. It processes and aggregates large-scale cloud data gathered from devices and provides monitoring and analytic tools to end-users (e.g., administrators of smart devices, service workers, or inhabitants of smart cities). Its main strength is the simplicity of data presentation delivered in the form of several dashboards, systematically organized in different levels of detail. For instance, basic users can monitor devices using high-level views with simple indicators presenting key performance indicators (KPIs). On the other hand, analysts could utilize drill-down actions to watch KPI alerts, detect device problems and analyze the reasons behind these problems to create service requests for service workers. Due to devices' geographic locations, KPI alerts and related device problems are problematic. Since many devices might be distributed in the city, such information is crucial to navigating the service worker to the device. It should be comprehensively displayed when an alert is focused. Geovisto was used for this purpose. Figure 10 shows an example of a device KPI alert selection and map focus.



Device type	Alert Type	Device ID	Account
Water Meter	Water Meter High Consumption	70b3b514900b075c	875862...
14040 Kenilworth St	Waverly	NE	68462
Harry Sullivan	5. 1. 2022	16:18:42	Action
Water Meter	Water Meter High Consumption	70b3b514900b0602	
	22. 12. 2021	22:19:46	

Showing 1 to 2 of 2 entries

WaterMeter-1

Street address: 14040 Kenilworth St, Waverly
Location: Basement - SW Corner
ID: 70b3b514900b075c
Alert: Water Meter High Consumption
Last Service Reported: N/A
Service Request Status: todo
Service Request Number: id

Fig. 10. Device alerts view. The user can click on an alert in the list (on the left) and the corresponding device is zoomed in on the map with description popup. Then, the user can create a service request.

In order to include the Geovisto map into the Logimic application, it was necessary to create an Angular component that serves as a wrapper for the Logimic map. The principle was similar to the React component created for Flowmon (Sect. 3). Since the *Geovisto Core* library is delivered as an npm package and it was reimplemented to TypeScript, it was easy to set this library as the npm dependency and import the library in the Angular component written in

TypeScript. Geovisto creation and API functions calls were integrated into the Angular lifecycle callbacks, similarly as it can be done by using React lifecycle hooks. We expect the library to be used either in pure JavaScript/Typescript projects or in various advanced UI frameworks (such as Vue.js or Svelte).

For the purposes of Logimic, we tested the following existing tools, currently provided by Geovisto:

– **Tile layer tool** displays a real-world map which is vital to locate the devices. Also, the satellite view is beneficial when the service worker is in the field and needs to quickly locate the devices (e.g., actual positions of lamps, entrance to buildings) as shown in Fig. 11. The tool itself does not provide a map tiles server. It only provides the ability to connect the tool with a chosen mapping provider service[21] using the Leaflet API. Logimic purchases API keys concerning the policy of the chosen map providers.

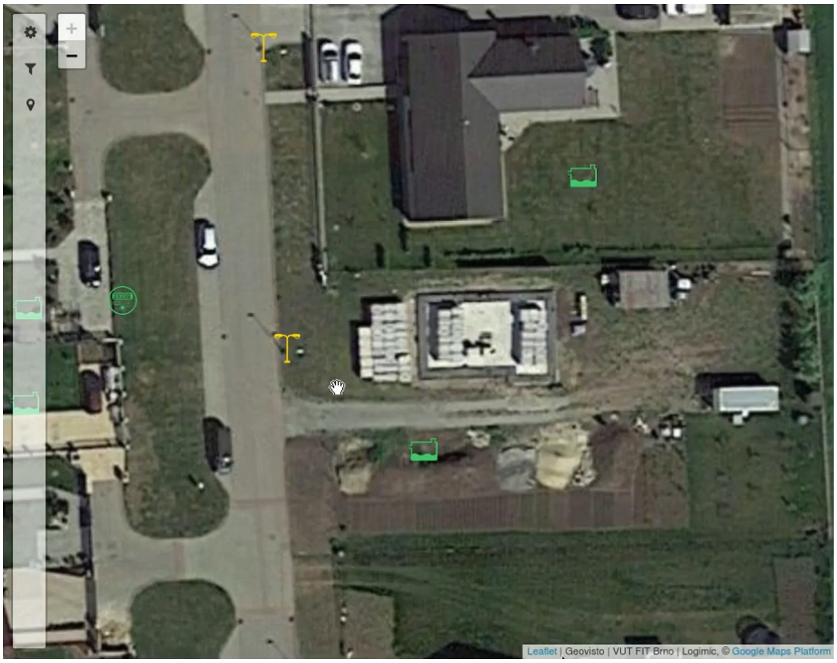
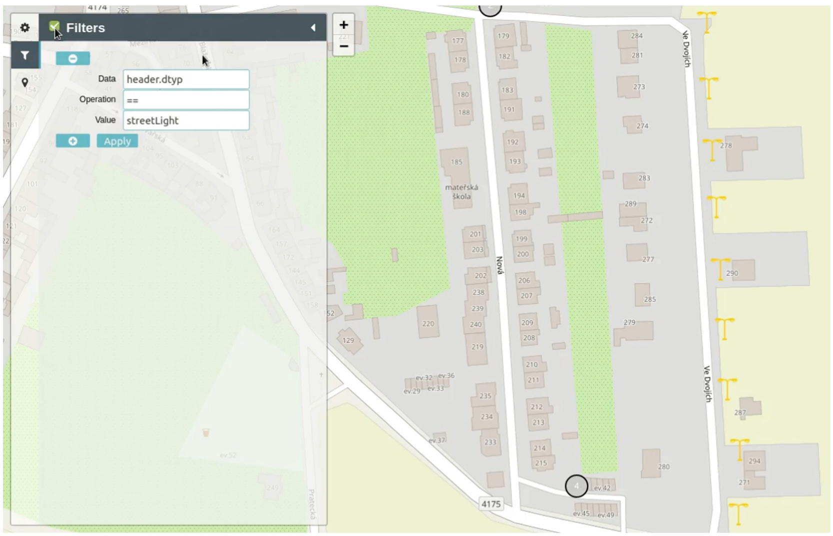

+
−
Leaflet | Geovisto | VUT FIT Brno | Logimic, © Google Maps Platform

Fig. 11. Satellite perspective using Google maps. It can provide the service worker with a detailed view of the focused neighborhood and help associate the device marker with the actual location.

– **Filters tool** allows to select only the devices that might be important either for the desktop user to analyze a map region or the service worker for navigation across the series of same devices (e.g., the distribution of lamps in the street – Fig. 12).

[21] https://github.com/leaflet-extras/leaflet-providers.



Data: header.dtyp
Operation: ==
Value: streetLight

Fig. 12. Device filtering. In this example, service workers can only see one device type (lamps) to unclutter the view and remove unneeded markers.

- **Themes tool** enables the widget appearance customization, so it fits the surrounding environment.
- **Sidebar tool** allows to add new controls for customization of mentioned tools. Currently, the controls are displayed as a part of the map widget, but with the planned redesign of the application, it might be required to move these controls outside the map widget and include them in the global menu. This was also one of Flowmon's requirements. Since the *Geovisto Core* and *tools* provide API for their customization, it is possible to control them externally.

Regarding the visualization of device markers, we integrated the Geovisto toolkit as part of the Logimic environment to demonstrate the possibility of extending Geovisto externally. The tool is similar to the Marker layer provided by Geovisto, but it adds some new Logimic-specific functionality:

- The tool is connected to the list of device alerts and listens to the selection changes in order to focus the device on the map when an alert is clicked.
- It extends marker popups for additional device metadata and allows to invoke service requests dialogs directly.

This tool can be published as an npm package; however, it represents an internal extension applicable in the Logimic application.

8 Discussion

As confirmed by the evaluation with Logimic, Geovisto's attributes (usability, modularity, configurability, extensibility, and accessibility) improved vastly. The multilayered map appeared to be an excellent way to display the geographic locations of IoT devices distributed in the cities. Satellite view, filter, and focus tools help locate the devices that alert KPI problems and navigate service workers to these devices. We expect that Geovisto might be used with different use cases of different industrial systems. Further, we present Geovisto's advantages and disadvantages.

8.1 Advantages

One of Geovisto's main advantages is that it provides UI tools to prototype map instances without implementing much code. Then, the state of a map instance can be serialized and exported. Even though the prototyping possibilities are limited, this functionality might be beneficial for a programmer trying to find the best map configuration. Also, it might improve the communication between the UX team, which can find an appropriate map configuration, and the programmers, which can use this configuration to implement the production version of the map.

Geovisto can work with generic datasets and project them onto custom geographical objects (concerning the capabilities of chosen map layers). Users can select various data domains, which allows them to explore the information further. Showing more layers at the same time helps the users to see the data in context. The interactive data filtering emphasizes the relations between geographical locations. The selection of map layer objects can be propagated to other layers, focusing the important details better with a combination of highlighting tools.

When the programmers need to create an unusual data projection, they can either choose from the existing *Geovisto Tools* or implement their own using the *Geovisto Core API.* The second version of the library was rewritten in the statically typed TypeScript language, improving code readability and decreasing the number of runtime errors caused by the wrong API application.

The library does have a modular architecture. It provides a thin core layer delivered as an `npm` package providing the core API. Then, additional *Geovisto Tools* extend the Core and provide particular map layers, controls, and utilities. Programmers can import additional `npm` packages in their projects only when required, which decreases the size of the result.

8.2 Limitations

Geovisto is an open-source library that is still under development. The usage scenarios showed only a fragment of geospatial data types that could be visualized. More usage scenarios and case studies are needed to validate the generalization of our approach. We should focus more on processing the large-scale data and the performance and profiling of the library.

Another limitation relates to data preprocessing that has to be done to gain a flat data structure. Our implicit approach causes enlargement and redundancy of the data. Thanks to that, the data can be processed quickly. However, it would be helpful to design an algorithm that can work with the data without preprocessing and more efficiently. The new version of Geovisto offers the possibility to override these algorithms. However, a more extensive set of implicit data processing strategies would improve its usability.

Last but not least, the lack of proper usability evaluation is another drawback of our work. For instance, selecting colors and combining several map layers is limited by people's comprehensibility of the widget. The users might be overwhelmed by the data, mainly when chosen inappropriate color combinations. The z-index of the layers is hardcoded, and users cannot change it. Usually, the users should not need to change the default settings (e.g., the marker and connection layers are above the choropleth), but they should control their ordering in the future with more layers added.

9 Conclusion and Future Work

There are three general approaches to authoring interactive geovisualizations: programmatic, template editing, and authoring tools. The existing authoring tools, however, are mainly focusing on 2D charts and diagrams, and their capabilities for authoring geovisualizations are limited. In our work, we focus on designing and implementing an authoring tool specifically focused on delivering interactive geovisualizations.

Earlier, we implemented a prototype version of the Geovisto toolkit based on JavaScript and Leaflet and demonstrated it on two usage scenarios (DDoS Attack Analysis and Covid 19 Pandemic open data) in [10]. In this paper, we followed up with an analysis of usage scenarios, identified prototype limitations, and revised the design requirements for the new version in terms of *usability*, *modularity*, *configurability*, *extensibility*, and *accessibility*.

Modular architecture allows to include only necessary parts of the library and decrease the size of the product. Configurability allows customization of included parts to integrate the map into the company's environment. Improved extensibility offers the creation of new map tools specific to companies. We re-implemented the Geovisto toolkit in TypeScript – a statically-typed language – that improves further development and decreases runtime errors.

The library has been published in the form of npm several packages[22]. The source codes are available on Github under MIT license[23].

The requirements reflected the intended industrial use and invoked changes in Geovisto's architecture and implementation. We presented the case study where Geovisto was used in the production-ready application for visualizing IoT sensor devices on the map developed by Logimic startup company. The case study confirmed that the Geovisto toolkit fulfills our goal of creating a programmatic mapping library that provides template editing known from mapping authoring systems.

[22] https://www.npmjs.com/search?q=geovisto.
[23] https://github.com/geovisto.

9.1 Future Work

There are also several sub-projects we are currently working on. One of them is a web service enabling non-programmers to prototype map instances using the Geovisto toolkit and include them as widgets on their websites. We are developing an infrastructure that will manage map configurations, datasets, and GeoJSONs. The system will provide the front-end application wrapper, including the UI tools to manage user-defined maps. The back-end will provide a configuration database and API for remotely fetching the configurations. We also plan to support loading data from third-party relational and non-relational database systems.

The second area deals with layer improvements and creating new ones. We have already implemented the bubble map, spike map, and heatmap layers, which play an essential role in the comprehensive data distribution visualization. Another almost finished tool provides animated geospatial data visualization with the time dimensions (i.e., timestamps). It enables the animation of spatio-temporal data domains and seeks them to the defined time frame. As a result, it will allow the user to see the evolution of values in individual geographic regions in time. Also, we plan to improve the visual appearance of the layers and add interactive map legends. Since the users can show multiple layers in the map simultaneously, we will pay closer attention to the color pallets used in the layers (e.g., sufficient color contrast, color-blind safe palette combinations). Lastly, we will add further visual dimensions to the layers and controls for manipulating the layers.

The third area focuses on quality assurance. Since the project is getting larger and composed of several packages that might use the API of others, it is necessary to do proper testing before publishing new versions. We will also pay attention to the documentation to better describe the API, including usage examples.

Acknowledgements. Jiří Hynek was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science – LQ1602". Vít Rusňák was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822) project. We also thank Progress Flowmon and Logimic, which provided usage scenarios and cooperated during the evaluation of the Geovisto toolkit.

References

1. Bostock, M., Heer, J.: Protovis: a graphical toolkit for visualization. IEEE Trans. Vis. Comput. Graph. **15**(6), 1121–1128 (2009)
2. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE Trans. Vis. Comput. Graph. **17**(12), 2301–2309 (2011). https://doi.org/10.1109/TVCG.2011.185

3. Degbelo, A., Kauppinen, T.: Increasing transparency through web maps. In: Companion Proceedings of the The Web Conference 2018, WWW 2018, pp. 899–904.International World Wide Web Conferences Steering Committee, Geneva (2018). https://doi.org/10.1145/3184558.3191515
4. Elasticsearch, B.: Maps for Geospatial Analysis (2020). https://www.elastic.co/maps, Accessed 10 Feb 2020
5. Gao, T., Hullman, J.R., Adar, E., Hecht, B., Diakopoulos, N.: NewsViews: an automated pipeline for creating custom geovisualizations for news. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2014, pp. 3005–3014. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2556288.2557228
6. Grafana Labs: Grafana: The Open Observability Platform (2020). https://grafana.com/, Accessed 10 June 2020
7. Grammel, L., Bennett, C., Tory, M., Storey, M.A.: A survey of visualization construction user interfaces. In: Hlawitschka, M., Weinkauf, T. (eds.) EuroVis - Short Papers. The Eurographics Association (2013). https://doi.org/10.2312/PE.EuroVisShort.EuroVisShort2013.019-023
8. Holten, D., Van Wijk, J.J.: Force-directed edge bundling for graph visualization. Comput. Graph. Forum **28**(3), 983–990 (2009). https://doi.org/10.1111/j.1467-8659.2009.01450.x
9. Huang, Q., Cervone, G., Jing, D., Chang, C.: DisasterMapper: a CyberGIS framework for disaster management using social media data. In: Proceedings of the 4th International ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data, BigSpatial 2015, pp. 1–6. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2835185.2835189
10. Hynek, J., Kachlík, J., Rusňák, V.: Geovisto: a toolkit for generic geospatial data visualization. In: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. SCITEPRESS - Science and Technology Publications (2021). https://doi.org/10.5220/0010260401010111
11. Li, X., Anselin, L., Koschinsky, J.: GeoDa web: enhancing web-based mapping with spatial analytics. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2015. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2820783.2820792
12. Liu, Z., Thompson, J., et al.: Data illustrator: augmenting vector design tools with lazy data binding for expressive visualization authoring. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, pp. 1–13. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3173574.3173697
13. Mei, H., Ma, Y., Wei, Y., Chen, W.: The design space of construction tools for information visualization: a survey. J. Visual Lang. Comput. **44**, 120–132 (2018). https://doi.org/10.1016/j.jvlc.2017.10.001
14. Ren, D., Lee, B., Brehmer, M.: Charticulator: interactive construction of bespoke chart layouts. IEEE Trans. Vis. Comput. Graph. **25**(1), 789–799 (2019)
15. Satyanarayan, A., Heer, J.: Lyra: an interactive visualization design environment. In: Proceedings of the 16th Eurographics Conference on Visualization, EuroVis 2014, pp. 351–360. Eurographics Association, Goslar, DEU (2014)
16. Satyanarayan, A., Moritz, D., Wongsuphasawat, K., Heer, J.: Vega-Lite: a grammar of interactive graphics. IEEE Trans. Vis. Comput. Graph. **23**(1), 341–350 (2017)

17. Satyanarayan, A., Russell, R., Hoffswell, J., Heer, J.: Reactive vega: a streaming dataflow architecture for declarative interactive visualization. IEEE Trans. Vis. Comput. Graph. **22**(1), 659–668 (2015)
18. Tableau Software, LLC.: Mapping Concepts in Tableau (2020). https://help.tableau.com/current/pro/desktop/en-us/maps_build.htm, Accessed 10 Feb 2020
19. Xavier, G., Dodge, S.: An exploratory visualization tool for mapping the relationships between animal movement and the environment. In: Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Interacting with Maps, MapInteract 2014, pp. 36–42. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2677068.2677071