https://doi.org/10.2352/EI.2023.35.1.VDA-402

VVAFER — Versatile Visual Analytics Framework for Exploration and Research

Moritz Zeumer; German Aerospace Center (DLR); Braunschweig, Germany
Jonas Gilg; German Aerospace Center (DLR); Braunschweig, Germany
Pawandeep Kaur Betz; German Aerospace Center (DLR); Braunschweig, Germany
Andreas Gerndt; German Aerospace Center (DLR); Braunschweig, Germany; University of Bremen; Bremen, Germany

Abstract

The development of interactive visualization applications that are applicable to many real-world problems is a challenging affair. For every new project, developers need to follow the same repetitive steps of fetching the raw data, transforming the data into processable form, defining visual structures and then displaying them appropriately. To accelerate this, we propose the Versatile Visual Analytics Framework for Exploration and Research (VVAFER). VVAFER is planned to be an extensible visual analytics framework, upon which different applications can be developed with minimum overload at the development side. Through modular architecture, unified data formats, reusable templates and software components, developers will be able to quickly deploy and create their visualization applications by configuring existing templates with their own specific functionalities. In this paper, we describe our motivation for this future framework and its architectural design.

Introduction

In developing a visualization application, a developer must follow several complex steps, which are an essential part of a visualization development pipeline. Following the information visualization pipeline from [1], the steps involved are: fetching raw data from the source, transforming the data into a data table, defining the visual structure of the data, and then displaying them appropriately. If these steps are not followed appropriately, people might interpret the data unintendedly or not understand the underlying information [2].

Visualization frameworks offer an off-the-shelf data import/storage solution. They often include a variety of widely used visualization layouts and algorithms that not only create the front-end visualizations but also provide an end-to-end instance for the complete visualization application. For an end-to-end solution in visualization development, frameworks are composed of different models: abstract data, visualizers, interaction operators, views, the user, etc. [3]. Due to the involvement of many discrete steps, the deployment, development and maintenance of frameworks is a labour-intensive job. Moreover, the coding skills required in such frameworks hinder users from effective construction [4]. D3.js [5], for example, is one of the prevalent visualization libraries that create very expressive visualizations which are difficult to realize in many current visualization production tools [4]. Additionally, the programming paradigm followed in these frameworks is imperative, where the software components are tightly coupled. Therefore, a lot of pre-processing or code modification is required to create a new data structure or visualization technique in the available framework. Previous studies [6] suggest that most development time is spent manually tweaking data or creating ad-hoc code. It is important to note that in most places, especially research institutions, developers primarily are also scientists. Thus, instead of allocating their valuable time to producing scientific artifacts, they have to spend it on non-scientific development tasks of repetitive nature. This raises the need for a solution that seamlessly performs all the concrete tasks needed in initially creating and deploying the base application.

To answer many such problems and accelerate our process of creating visualization applications in the visual analytics group of DLR, we propose VVAFER. In our group, we receive scientific datasets prominently for spatio-temporal analysis. Most of the times, the main questions that need to be answered are similar; however, sometimes they deviate based on the datasets and type of study. Earlier, we were developing a new framework for producing the visual analytics tools for every new dataset. We figured out that as most visual tasks are moderately similar, i.e. looking at spatial distribution, finding correlations, and subsequent tasks. The amount of work in creating relatively similar visualizations and tools can be easily reduced if we have a versatile framework in place. This leads to the conception of VVAFER, a Versatile Visual Analytics Framework for Exploration and Research.

VVAFER is planned to be the basis for future development in the visual analytics (VA) field at DLR. This framework will provide essential, pre-designed modules for the easy deployment of VA applications. From an easy project setup to the data acquisition from different sources, data wrangling, and the creation of visualization and its controls — it will provide high-level reusable software modules to perform all these activities seamlessly and efficiently. So, it will provide a solution for developers to deploy, configure and run their VA applications without much labor unrelated to the visualization task itself.

Motivation

While developing applications to visualize scientific or statistical data, lots of repetitive steps need to be undertaken. At the same time for every new visualization project, many tasks are similar, if not the same. They all include the same code for initializing a graphical context, for file input/output, data pre-processing, user interaction, etc. [7]. To illustrate this, we present three applications in the area of visualization and have highlighted their similarities.

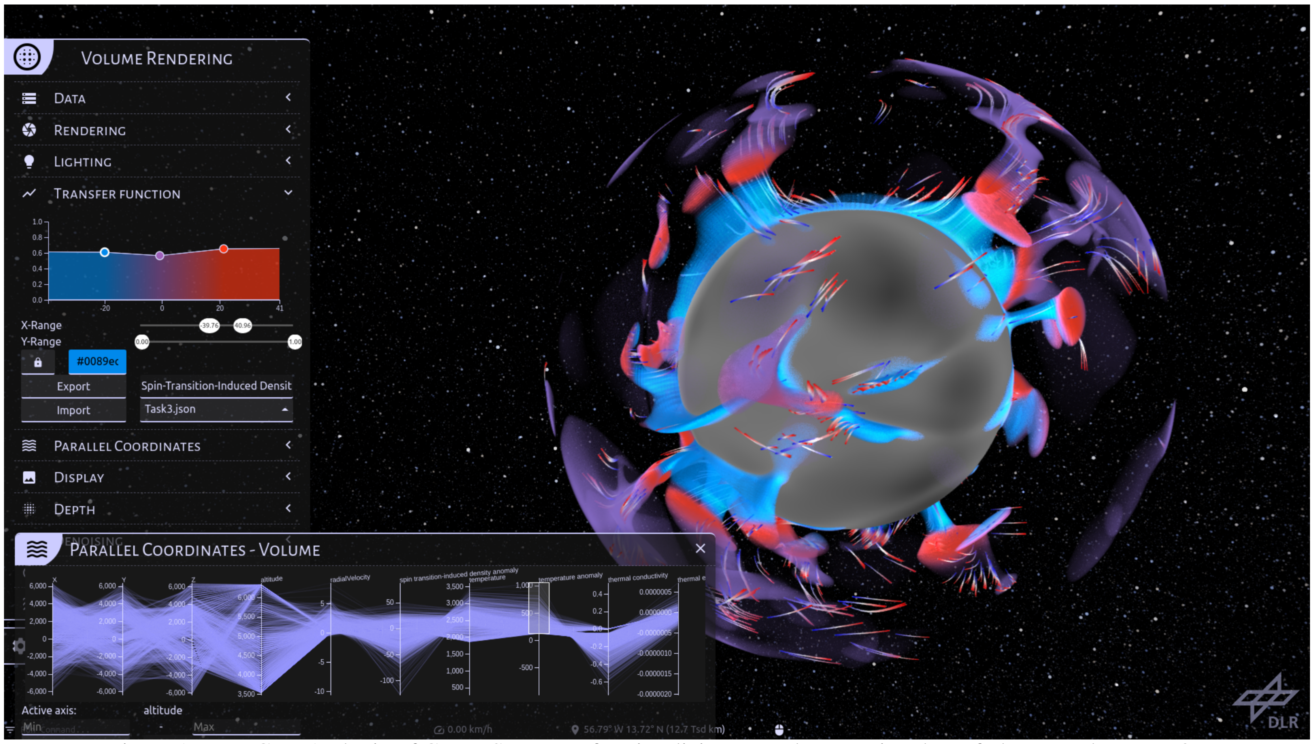


Figure 1: RayPC — A plugin of CosmoScout VR for visualizing mantel convection data of planets and moons [8].

CosmoScout VR

CosmoScout VR [9] is a visualization tool for scientific data within the solar system. It has many use cases for visualizing data.

One such use case is visualizing the mantle convection of planets like Earth and Mars. A CosmoScout VR plugin called RayPC [8] was developed to tackle the task. Initially, the raw data was provided by planetary scientists, and the data is unsuitable to work with directly and has to be pre-processed. In this specific case, an octree-based level-of-detail data structure was generated. Additionally, derived variables were computed, which are helpful for the analysis later.

For data analysis an interactive visualization was developed. A screenshot can be seen in Figure 1. In the center is the 3D visualization of the Earth mantle. On the left are configuration options for the visualization. Currently, the transfer function editor is selected. At the bottom is a parallel coordinate plot that displays how the scalar values are distributed over the cells of the dataset.

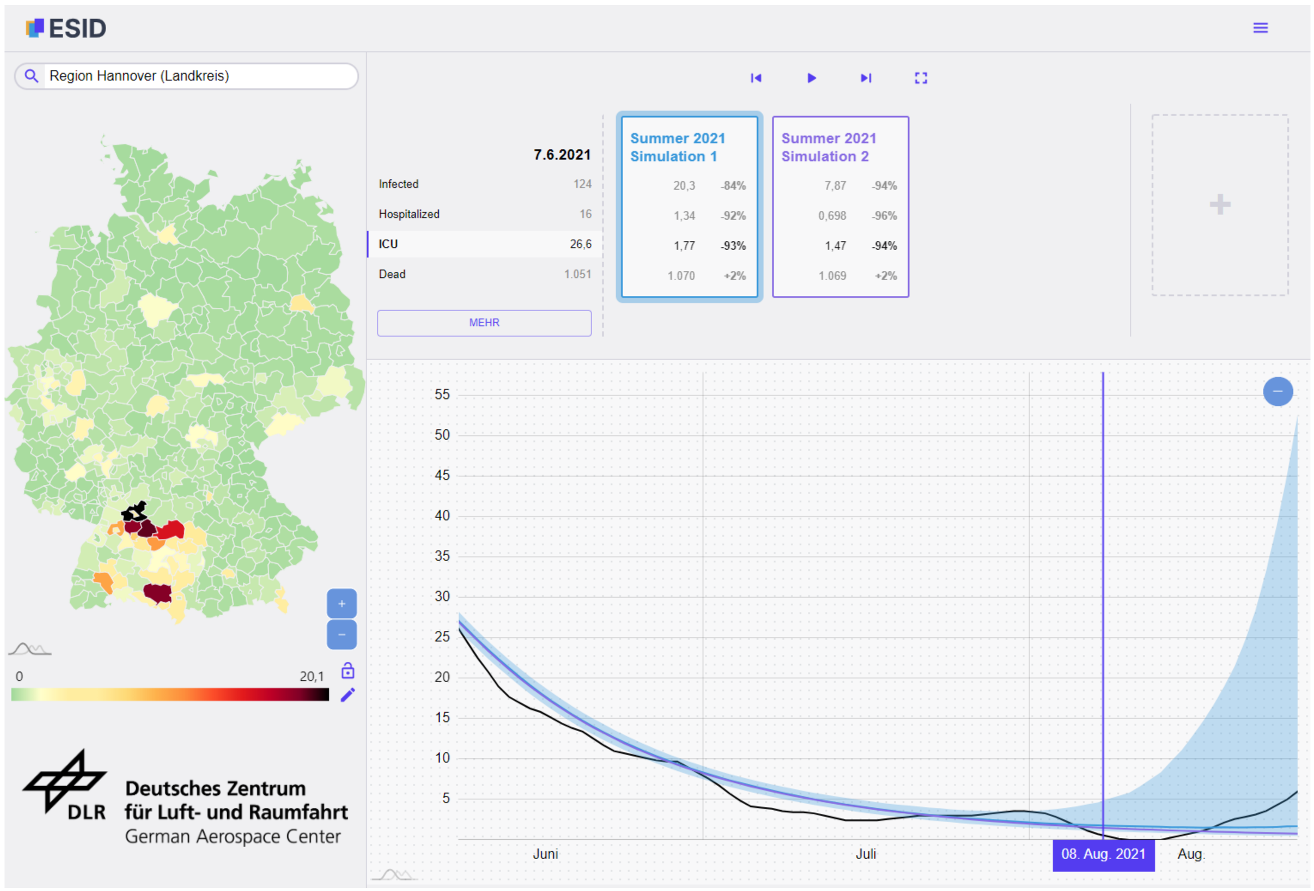
There are multiple ways to interact with this visualization. First, it is possible to move the camera freely in 3D space and look at the mantle data from different viewpoints. Second, the transfer function is editable; thus, the coloring and transparency of the 3D visualization can be modified in real-time. Lastly, the parallel coordinate plot can be used to filter the cells by scalars. This interactivity helps understand the flows and structures of the mantle convection easily.

While the 3D visualization is written in C++, the user interface is a standard website with HTML, JavaScript and CSS. The transfer function editor and the parallel coordinate plot come from different libraries and need different formats for working with the data.

ESID

ESID is the visualization and analytics interface developed as frontend for metapopulation models by Koslow et al. [10] and Kühn et al. [11], [12] to analyze simulations of epidemics and pandemics. ESID reads a predefined data format and, therefore, could also visualize output from other models as long as they comply with the format definition. Primarily, it is intended to be used by local health authorities and political decision-makers to create non-pharmaceutical interventions with minimal restrictions and the largest effect. Additionally, it should also be open to the public, so that decision processes can be retraced and better understood which should improve the acceptance of the taken measures. The interface is being developed according to preliminary user interface (UI) and user experience (UX) studies by Stoll and Grappendorf [13].

Figure 2 shows an early version of ESID. On the left side is a map of the official German districts with a customizable heat legend. In the top section are two cards representing two different simulation scenarios: blue and purple. Left of the scenario cards is a list of compartments, like *infected*, *hospitalized*, and *dead*. The bottom portion contains a line chart comparing the scenarios over a set timeline. This tool has multiple ways to interact, filter and select data. The following table shows the relationship among different components and how they coordinate with each other and visualize data.



7.6.2021		Summer 2021 Simulation 1		Summer 2021 Simulation 2	
Infected	124	20,3	-84%	7,87	-94%
Hospitalized	16	1,34	-92%	0,698	-96%
ICU	26,6	1,77	-93%	1,47	-94%
Dead	1.051	1.070	+2%	1.069	+2%

Figure 2: An early version of ESID, an application for analyzing pandemic simulation data.

	Scenario	Compartment	Date	District
Card	select	all	1	1
List	none	select	1	1
Chart	all	1	select	1
Map	1	1	1	select

Table 1: Displayed information and interaction between visualization components of ESID.

The table can be read the following way: The Map shows the data of one scenario for one compartment on one date over all districts. Since it shows all districts, it also serves as a selection tool for districts. The Chart shows the data of all scenarios of one compartment over all days of one district. Since it shows all dates, it also serves as a selection tool for dates.

From Table 1 and Figure 2, one can see that all components are coordinated and pass some filtered data to each other which then requires extensive communication with a unified data store and formatting of the data. Unfortunately, the current application version displays each component using different libraries, which use different data structures. Each component has to transform the incoming data to the required format and pass it to the visualization library.

PANDEMOS-UI

PANDEMOS-UI, similar to ESID, is a visualization and analysis tool for pandemic simulations. It differs from ESID in the type of data it needs to visualize. While ESID shows district-level data with daily time steps, PANDEMOS focuses on the simulation and visualization of individual agents. The data is about moving agents with a spatial resolution of a few meters and a temporal resolution of minutes.

In the case of Germany, 80 million agents would be moving through the simulation, and visualizing this is not trivial. First, the data has to be loaded via a REST-API serving JSON data. The data needs to be filtered using different visual components so the user can get meaningful information from it. To see the spatio-temporal movement of incoming trajectories for one month, for example, first the time must be filtered and then the specific geographical locations. Finally, using different statistical charts, the data about different agents needs to be shown. To fulfil all these tasks, we need different libraries for getting the data, filtering the data and displaying the data. Different libraries come with their conventions, which a developer needs to adhere.

Common Problems

There are multiple common problems we encounter in all our visual analytics applications. The first problem is loading the data from different sources (APIs, databases, etc.) and in different formats (JSON, geoJSON, CSV, etc.). Then, an even bigger problem is to support the coordinated environments, which are a must in real-world visual analytics applications. In a coordinated environment, different visual components talk to each other when a user interacts with them. This leads to lots of data passing between the components. Each visual component needs its specific data structure. For example, maps need geoJSON data, whereas most statistical charts can be created with data formatted in arrays or CSV. Different libraries enable different tasks, leading to a lot of boilerplate code that deals with data conversion. This is inefficient for the developer and for the application's performance.

Related Work

There is a plethora of impressive visualization frameworks and toolkits available today, and the in-depth review of each is beyond the scope of this paper. Interested readers can look into a community-curated list available at the InfoVis-Wiki[1]. In the following section of our paper, we will examine some notable works in visualization software development.

The Visualization Toolkit (VTK) [14] is one of the earliest known open-source software system for 3D computer graphics, image processing, and visualization. It is written in C++ and provides a collection of libraries and tools for developers to create interactive 3D visualizations and simulations. VTK supports a wide range of data types and file formats, and it can be used for various applications, including scientific visualization, medical imaging, and virtual reality. Though it provides a modular architecture, it has a steep learning curve and a complex architecture, making it difficult for new users to get started. Moreover, as it is primarily designed for 3D visualizations, its support for 2D and information visualizations is limited. VTK does not have a built-in user interface component, which means developers must use a separate library or create their own graphical user interface. However, some of these drawbacks can be mitigated using ParaView [15], which is built on top of VTK and provides a collection of tools and libraries for scripting via Python, web visualization through ParaViewWeb, and in-situ analysis with Catalyst. Like VTK, to use Paraview, developers need to set up their project environment and build the Paraview libraries manually. This usually involves downloading the source code, configuring the build environment, and then compiling the libraries. This process can be time-consuming and may require some knowledge of the underlying operating system and build tools.

Prefuse [16] and the Infovis Toolkit [17] are a few of the earliest known open-source Java-based toolkit for creating interactive data visualizations. They provide a set of libraries and tools for building visualizations that can be integrated into web-based and desktop applications. They provide wide range of visualization techniques, which can be quickly set up and used. However, as these tools are not actively maintained, thus they cant run on modern browser which do not support flash or java applets anymore. They provide limited support to modern multidimensional visualization techniques like parallel coordinates and heatmaps and does not provide advanced interactions like linked views.

Commercial visualization software like Tableau, Microsoft PowerBI, and Charts in Microsoft Excel are primarily built for business applications and end-users. They follow a "one-big-tool" strategy [18], providing a rich interface with numerous features that are designed to be user-friendly and easy to explore. However, this can make the interface overly cluttered and confusing for some users. Additionally, since these software's source is not open, users and developers cannot add and remove features and configure the interface to their specific requirements. This can make it difficult for users to customize the software to meet their unique needs. Another drawback is that many commercial visualization software hosts users' data on their servers, which can lead to data privacy and security issues, especially for scientific and research projects. This can limit the use of these tools in sensitive or confidential environments where data privacy and security are crucial. Additionally, these tools are generally more expensive than open-source visualization tools, which can be a concern for some users.

Open-source visualization libraries have earned much community attention since the last decade, mainly due to the popularity of scripting languages like JavaScript, Python and R. These libraries are primarily intended for developers and require knowledge of both the library and the base scripting language. For example, D3.js [5] for javascript, matplotlib [19] for Python and ggplot2 [20] for R. While relatively easy to learn, they may require significant training for non-technical users. These libraries are not capable of creating complete visualization applications and are often used as visualization components in modern web-based frameworks like Angular[2] and React[3]. For example, the RawGraphics [21] visualization toolkit is based on AngularJS for the frontend and D3.js [5] for the visualization models.

Recent years have seen a significant growth in visualization applications based on modern web frameworks that are specifically designed for a particular domain. However, these frameworks can have limitations in their flexibility and reusability, as they often require significant base code changes when applied to other domains and have limited support for data management capabilities. For example, Geovisto [22] is a geospatial analysis framework that combines the advantages of authoring systems and programming libraries to let the user configure maps with some interaction and control. Their base component is a map and all the other interactive components (zoom, filters), data models (JSON, XML, GeoJSON) and statistical charts (nested pie charts) are based on this base component. Therefore, their modules cannot be reused for non-spatial scenarios. These Frameworks primarily support front-end configuration and provide little assistance with the complete visualization pipeline. Such domain specific frameworks can be tightly connected and configured to the demands of their specific datasets, but it also makes them less versatile and less adaptable to different types of data. This can make them less suitable for developers looking to customize or configure their visualization pipeline. For example, ICE[23] is a highly responsive user interface for exploring spatial patterns in large-scale environmental datasets. They support routines from their own modeled datasets and thus provide limited data management capabilities.

Although our motivation mainly stems from our projects in spatio-temporal domains, the goal of VVAFER is to create a framework that can be easily applied to different domains with minimal changes to its base functionality. In the next section the core aspects and architecture of this proposed framework is presented in more detail.

[1]https://infovis-wiki.net/wiki/Toolkit_Links

[2]https://angularjs.org/

[3]https://reactjs.org/

VVAFER

VVAFER is planned to be an extensible visual analytics framework upon which different visualization applications can be developed with minimum overhead on the development side. Through different modular components, a developer is free to deploy any visualization project and configure and show different visualization types. The core architecture of VVAFER is shown in Figure 3 and its features are described below:

- **Standard Visual Analytics (VA) Framework**: VVAFER will provide a standard visual analytics framework which provides various base components pertinent to different steps in the visualization pipeline. This way, a full-fledged visualization application can be developed without using other libraries or programming packages. This will save a developer's time for more productive tasks, rather than searching, including and connecting these libraries in their project.
- **Reusability**: VVAFER will provide template-based module development. Each module consists of its boilerplate code and routines that can be further configured to match the individual requirements of the project.
- **Standard Visualization Toolkit**: We will primarily be handling the visualizations that assist in spatio-temporal analysis (different forms of maps), basic statistical analysis (scatter plot, line charts, bar charts, pie charts etc.) and multidimensional data analysis (parallel coordinate charts, scatter plot matrices). All these chart modules can be easily configured and reused in different applications. The framework itself will be flexible enough, so that inclusion of new visualization types can be possible without much overhead. At the same time, the exclusion of one or another visualization types will not affect the functionality of other visualization types.
- **Standard Data Handling Modules**: The pre-processing steps for data handling in creating visualization systems are quite similar. For example: Loading the data from different formats, creating a standard data schema, ensuring data quality, creating data views etc. VVAFER must support a standard module to handle this for different types of data we receive. We would also be creating routines that can be expandable to the data requests not handled by us.
- **Extensibility**: The VVAFER framework must be easily extensible without much change to the base architecture and with minimal programming so that anyone who understands basic coding can deploy it to create their application.
- **Data Controls**: We will support basic data and interactive controls to explore the data. For example, controls that support: lenses, dragging, zooming, common data transformation (calculating min, max, distances) etc.
- **Project Setup**: A framework that assists in easy project setup (e.g. as in create-react-app[4] with minimal dependencies, so users with limited coding knowledge can adapt to their environments. At the backend, the framework will provide boilerplate functions for testing and debugging.
- **Rapid Prototyping**: A framework that allows developers to quickly and easily build and test different ideas and concepts without having to start from scratch each time. This can save a significant amount of time and effort, and also allows for more experimentation and iteration in the development process.
- **Accessibility and Open Source**: The first requirement of accessibility can be accomplished by building on web technologies, which allows access to the application from any device with a browser, without the need to install any additional software. Web technologies also have the advantage of a large ecosystem with easy access for developers. This framework will be open source and available with an appropriate license.

[4]https://create-react-app.dev/

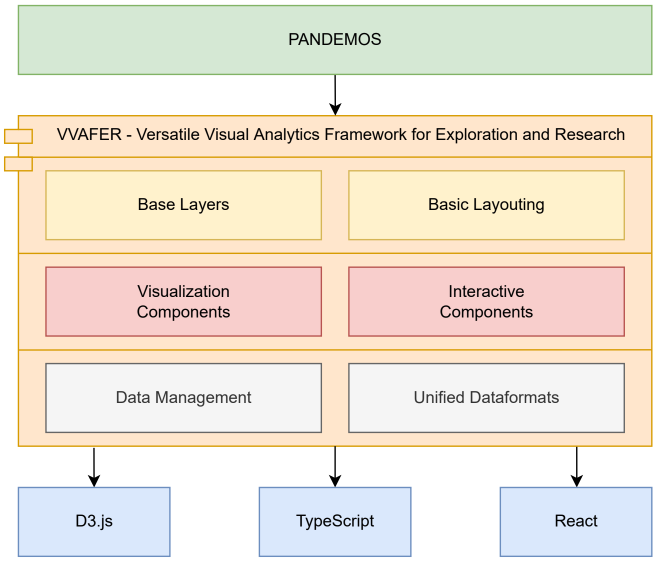


Figure 3: An overview of the proposed architecture for VVAFER.

Architecture

VVAFER's architecture consists of multiple layers with modularity and separation of concerns in mind. The foundational layer of VVAFER provides the necessary tools for data management and standardization, including data import functionality and a unified data format. This makes it easy to import and format data, and ensures efficient and standardized access and interoperability by higher-level components. The use of a unified data format throughout the architecture also helps to reduce complexity and ensure consistency.

The component library is seperated into visualization and interaction components, which provides greater flexibility. For example, CosmoScout VR's case, where only interactive components are part of the web interface and the visualization is handled in C++.

VVAFER's design is influenced by the React library, which is used for building interactive and visual components in web applications. Unlike React, VVAFER places a greater emphasis on separating the visualization and interaction components. This approach provides a powerful tool for creating custom visual analytics and visualization applications, while also ensuring ease of maintenance and extension.

In summary, the VVAFER architecture provides a flexible and efficient solution for creating visual analytics and visualization applications, with a focus on modularity, separation of concerns, and standardization.

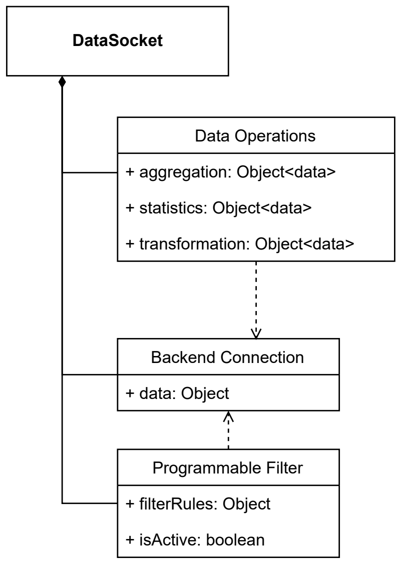


Figure 4: Diagram of the data socket component structure.

Data Management

The connection between the data layer and the higher components is a data socket component which handles the data management behind the scenes, like connections to data backends (REST APIs, databases or other sources). The data socket also provides information on the dataset like statistical values, aggregations and other operations. Additionally, programmable filters can be added, providing alternative versions of the raw dataset based on filter rules and similar transformations. In summary, the data socket component handles the fundamental data management and provides datasets in a unified format for the visualization and interactive components.

Unified Dataformats

One of the largest problems in developing visualization applications is that each library uses its own data format. The data loading library uses a different format than the graphing library for example. This leads to a lot of code dealing with conversion between data formats. VVAFER will use a single data format for each data type across the whole application. This will reduce the amount of code, improve the communication between components and improve the performance. Since there are a lot of different data types for different use cases, an incremental approach will be used. There is already research that tackles this task of creating a standard data format for specific domains. For movement data, which will be the first priority of VVAFER, there is a good starting point by Andrienko et al. [24]. Data formats for other domains will be researched and implemented when required, since it is impossible to deal with all formats of all domains from the start.

Interactive Components

The interactive component is one of the two key component types of VVAFER. A key concept of visualizations, especially visual analytics, is the ability to interact and manipulate the data and visualization views to explore the data. All interactions are bundled into an interactive component container where core interactive elements like buttons, sliders, drop-down menus, and corresponding labels are compiled and arranged inside the container. An interactive container can also have a component listening for input on peripherals like key-press events on keyboards or other peripherals. Each element contains its styling and other internal code, and only exposes the properties to the parent container relevant for the high-level interaction. This can be a function handle, which is called when a button is pressed or a slider changed, as well as properties like a list of selectable options and the current value for drop-down menus and search bars. The parent container handles the interaction logic, modifies the data sockets and exposes specific interactions to change properties of visualization components to the parent layout container, which facilitates the connection between the components. An example of an interactive component container is map controls for the visualization of geo-referential data. The container has two buttons that are bundled in a function realizing zooming in and out of the map, as well as a nested interactive container that uses a set of checkboxes and exposes a list of visible layers on the map. These functions and properties are exposed to the parent Layouting component, which connects the information to the visualization component used to display and interact with the map correctly. The interaction components are strongly separated from the visualization components to facilitate easier nested and coordinated visualization controlled by a single set of interactive controls manipulating multiple visualization components or the shared data sockets.

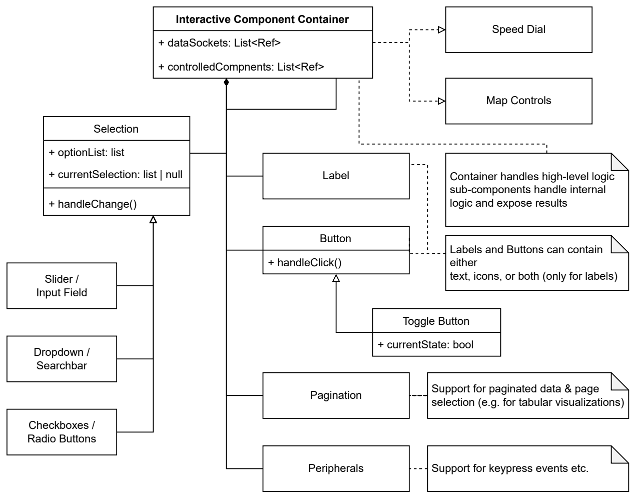
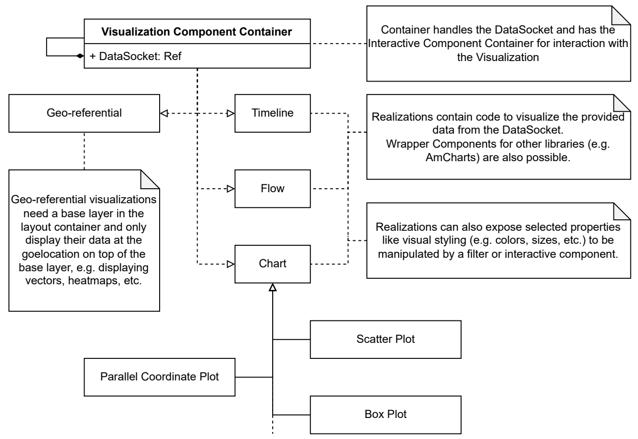

Interactive Component Container
+ dataSockets: List<Ref>
+ controlledCompnents: List<Ref>
Speed Dial
Map Controls
Selection
+ optionList: list
+ currentSelection: list | null
+ handleChange()
Label
Container handles high-level logic sub-components handle internal logic and expose results
Button
+ handleClick()
Labels and Buttons can contain either text, icons, or both (only for labels)
Slider / Input Field
Dropdown / Searchbar
Checkboxes / Radio Buttons
Toggle Button
+ currentState: bool
Pagination
Support for paginated data & page selection (e.g. for tabular visualizations)
Peripherals
Support for keypress events etc.

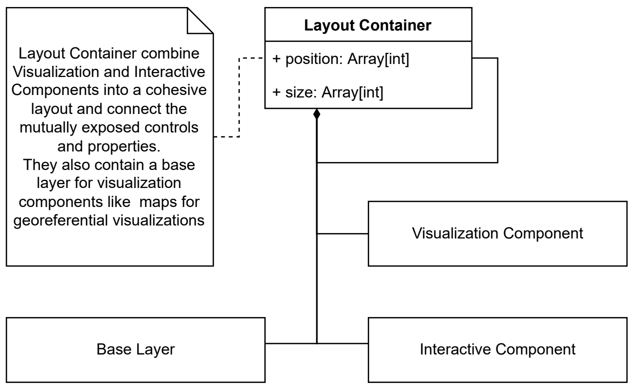
Figure 5: Diagram of the interactive component structure.

Visualization Components

The visualization component is the other fundamental component. It uses the data provided by the data socket and handles the visualization of it. The visualization component also exposes necessary settings to be modified or passed to interactive components. Since the visualization type is strongly dependent on the data, the component resides in an abstract wrapper that only contains a reference to a data socket. The realization can then be flexible and even wrap other libraries that visualize the data from the data socket, like amCharts[5] or Leaflet[6]. This way new visualizations can be created as components and added to a library of visualizations which can be reused in any VVAFER project.

[5]https://www.amcharts.com/
[6]https://leafletjs.com/


Visualization Component Container
+ DataSocket: Ref
Container handles the DataSocket and has the Interactive Component Container for interaction with the Visualization
Geo-referential
Timeline
Realizations contain code to visualize the provided data from the DataSocket.
Wrapper Components for other libraries (e.g. AmCharts) are also possible.
Flow
Geo-referential visualizations need a base layer in the layout container and only display their data at the goelocation on top of the base layer, e.g. displaying vectors, heatmaps, etc.
Realizations can also expose selected properties like visual styling (e.g. colors, sizes, etc.) to be manipulated by a filter or interactive component.
Chart
Scatter Plot
Parallel Coordinate Plot
Box Plot
Figure 6: Diagram of the visualization component structure.


Layout Container combine Visualization and Interactive Components into a cohesive layout and connect the mutually exposed controls and properties.
They also contain a base layer for visualization components like maps for georeferential visualizations
Layout Container
+ position: Array[int]
+ size: Array[int]
Visualization Component
Base Layer
Interactive Component
Figure 7: Diagram of the layout container structure.

Additionally, visualization components can contain nested visualization components, for example, to display visualizations like polar diagrams or pie charts instead of glyphs or markers in other diagrams. For example, a geo-referential visualization uses the data from the data socket to fill a canvas with information on top of the base layer of its parent Layout container, like heatmaps or movement vectors displayed on top of a topographical map. The visibility state of the layer is exposed, so an interactive component can manipulate the layer visibility.

Layouting and Base Layers

Combining the two key components into a cohesive structure is a standard container system. These layout containers can be arranged like HTML *div* elements with size and position in the overall layout. They can contain nested layout containers, visualization or interactive component containers. Additionally, the functions and properties exposed by visualization and interactive component containers are connected in the layout component, similar to a breadboard connecting the general sub-components. The layout containers can also be influenced by interactive components in parent layout containers to change their properties if needed, like their size, position, visibility, etc. Additionally, a layout container may contain a base layer necessary for components like geo-referential or other special visualizations that require a specific environment, like a topological map, to function properly. For example, this base layer can contain a map, which is the background for a visualization component to draw a heatmap onto.

Workflow

A flow chart for creating a visualization or visual analytics dashboard with VVAFER is shown in Figure 8. Initially, the basic VVAFER template is generated, creating the boilerplate code, which initializes the basic functionality and data types using `npx create-vvafer-app <app_name>`, similar to the creation of React apps[7]. Then, the raw data is imported and provided to the application through a data socket. Next, a visualization component container is created with the desired visualization inside. If the visualization requires any form of interactive controls, an interactive component is created with the control elements to manipulate the visualization. If the visualization needs programmable or static filters, they can be added to the data socket and controlled by the interactive component. These steps are repeated if more visualization components are needed. Different datasets can be added via the same or a different data socket, depending on the data source. Finally, the created components are styled, arranged, and connected within the layout containers, and base layers are configured where needed. The different components can also be created using npx commands to generate a template for the components. The template contains a basic react function component where necessary properties for a VVAFER component are already in place, like a hook that provides the reference to the corresponding data socket. The desired visualization or interaction can then be implemented at locations in the code indicated by comments.

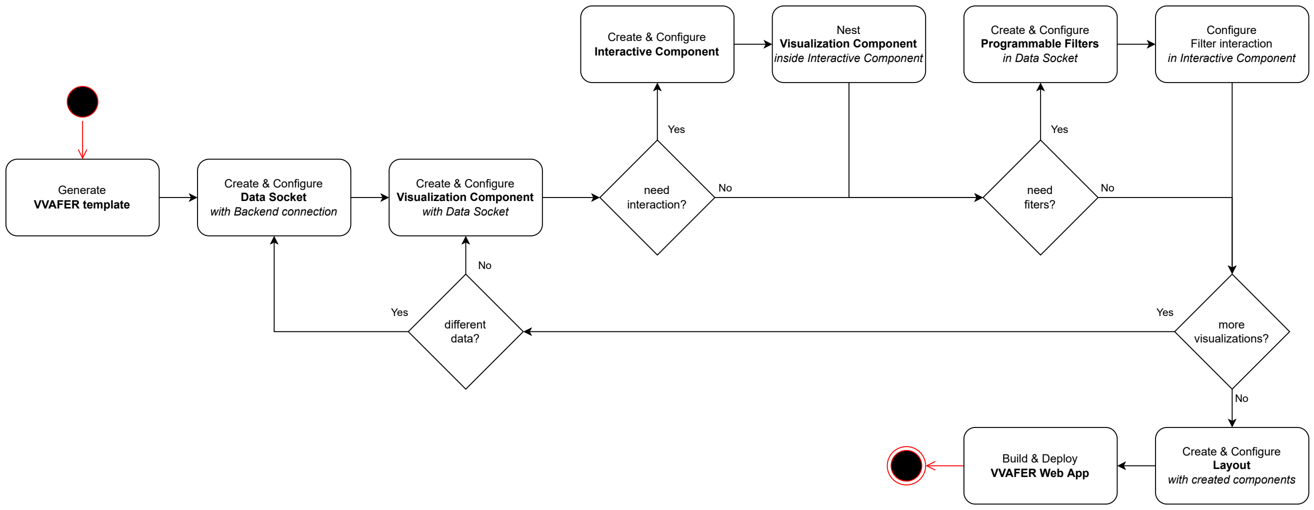
Example: ESID

In order to visualize the process more clearly, we present an example of how the workflow is used to create a visualization project like ESID. As mentioned above, first the general project structure is generated. This includes the foundational file structure as well as necessary boilerplate code. Next, the database and REST API are connected to the application through a data socket component that bundles the application's requests to the database. Depending on available resources and data size, the data socket can also cache responses to increase performance. With the data flow into the app created, the necessary visual and interactive component can be created according to the screenshot in Figure 2:

- **Scenario Cards** (top): a visualization component to display the scenario and case data values and an interactive component bundling functions to select scenarios and compartments, and the buttons to move through the data along the temporal axis.
- **District Overview** (left side): a visualization component that handles drawing the choropleth map, as well as an interactive component containing functions and modules to handle map controls, selecting districts, and the transfer function editor to modify the coloring of the map.
- **Timeline Chart** (bottom): a visualization component drawing the available data along the temporal axis, and finally an interactive component with functions to zoom into the line chart and select dates.

With the basic component created, nested layout containers are created to tie the application together. The layout containers determine the makeup and visual structure of the application as

[7]https://reactjs.org/



Generate VVAFER template
Create & Configure Data Socket with Backend connection
Create & Configure Visualization Component with Data Socket
need interaction?
Yes
No
Create & Configure Interactive Component
Nest Visualization Component inside Interactive Component
need fiters?
Yes
No
Create & Configure Programmable Filters in Data Socket
Configure Filter interaction in Interactive Component
more visualizations?
Yes
No
different data?
Yes
No
Create & Configure Layout with created components
Build & Deploy VVAFER Web App

Figure 8: An example flowchart of creating a visualization or visual analytics dashboard in VVAFER.

seen in the screenshot (Figure 2). Inside the layout container the properties of the visualization components, like highlighting for selected scenarios, compartments, dates, and districts, as well as other functionalities like the transfer function editor modifying the palette used for the choropleth map.

Once everything is connected, the application can be built and deployed. New, additional features in visualization or interaction can be incorporated into existing modules or preferably implemented in new components without the need to touch exiting code unless absolutely necessary.

Conclusion

From our various visual analytics projects at DLR, we have felt the need for a framework that avoids repetitive programming of similar modules. We realized the benefits to have a versatile framework that supports all common processing steps and is extensible to include and configure the new functionalities. This resulted in a conception of VVAFER (Versatile Visual Analytics Framework for Exploration and Research) which we have presented in this paper. We are looking forward to start development based on the reasoning behind this framework, the shown architecture, and its modular designed components. Through its modular component architecture, VVAFER can support the construction of any domain-independent visualization and analytics application with minimum development overhead.

VVAFER is a proposed framework for visual analytics that aims to avoid repetitive programming and support common processing steps. It is designed to be versatile and extensible, with a modular component architecture that allows for the construction of domain-independent visualization and analytics applications with minimal development overhead. The authors of the paper have presented the reasoning and architecture behind VVAFER and plan to begin development based on these ideas.

Funding

The authors gratefully acknowledge the funding by the German Federal Ministry for Digital and Transport within the mFUND Innovation program under the grant agreement FKZ19F2211A.

Bibliography

[1] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, Jan. 1999, ISBN: 9781558605336.

[2] O. Kulyk, R. Kosara, J. Urquiza, and I. Wassink, "Human-centered aspects," in *Human-Centered Visualization Environments: GI-Dagstuhl Research Seminar, Dagstuhl Castle, Germany, March 5-8, 2006, Revised Lectures*, A. Kerren, A. Ebert, and J. Meyer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 13–75, ISBN: 978-3-540-71949-6. DOI: `10.1007/978-3-540-71949-6_2`.

[3] M. Sanver and L. Yang, "A linking mechanism to integrate components of a visualization framework," in *2009 13th International Conference Information Visualisation*, 2009, pp. 92–97. DOI: `10.1109/IV.2009.39`.

[4] "Viscomposer: A visual programmable composition environment for information visualization," *Visual Informatics*, vol. 2, no. 1, pp. 71–81, 2018, Proceedings of PacificVAST 2018, ISSN: 2468-502X. DOI: `10.1016/j.visinf.2018.04.008`.

[5] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011. DOI: `10.1109/TVCG.2011.185`.

[6] A. Ruiz. "The 80/20 data science dilemma." (2017), [Online]. Available: `https://www.infoworld.com/article/3228245/the-80-20-data-science-dilemma.html` (visited on 07/22/2022).

[7] P. Gralka, M. Becher, M. Braun, *et al.*, "Megamol–a comprehensive prototyping framework for visualizations," *The European Physical Journal Special Topics*, vol. 227, no. 14, pp. 1817–1829, 2019. DOI: `10.1140/epjst/e2019-800167-5`.

[8] J. Fritsch, M. Flatken, S. Schneegans, A. Gerndt, A.-C. Plesa, and C. Hüttig, *Raypc: Interactive ray tracing meets parallel coordinates*, 2022. DOI: `10.48550/ARXIV.2207.12011`.

[9] S. Schneegans, M. Flatken, and A. Gerndt, *CosmoScout VR*. DOI: 10 . 5281 / zenodo . 3381953. [Online]. Available: https : / / github . com / cosmoscout / cosmoscout-vr.

[10] W. Koslow, M. J. Kühn, S. Binder, *et al.*, "Appropriate relaxation of non-pharmaceutical interventions minimizes the risk of a resurgence in sars-cov-2 infections in spite of the delta variant," *PLOS Computational Biology*, vol. 18, no. 5, pp. 1–26, May 2022. DOI: 10 . 1371 / journal . pcbi.1010054.

[11] M. J. Kühn, D. Abele, T. Mitra, *et al.*, "Assessment of effective mitigation and prediction of the spread of sars-cov-2 in germany using demographic information and spatial resolution," *Mathematical Biosciences*, vol. 339, p. 108 648, 2021, ISSN: 0025-5564. DOI: https://doi.org/10.1016/j.mbs.2021.108648. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0025556421000845.

[12] D. Kühn Martin J.and Abele, S. Binder, K. Rack, *et al.*, "Regional opening strategies with commuter testing and containment of new sars-cov-2 variants in germany," *BMC Infectious Diseases*, vol. 22, no. 1, p. 333, 2022, ISSN: 1471-2334. DOI: 10.1186/s12879-022-07302-9.

[13] J. Stoll and V. Grappendorf, "Esid - epidemiologisches simulationstool für den infektionsschutz in deutschland," Betreuung der Arbeit im DLR: Martin Joachim Kühn, Bachelorarbeit, Hochschule für Gestaltung Schwäbisch-Gmünd, 2021. [Online]. Available: https://elib.dlr.de/143505/.

[14] W. J. Schroeder, L. S. Avila, and W. Hoffman, "Visualizing with vtk: A tutorial," *IEEE Computer graphics and applications*, vol. 20, no. 5, pp. 20–27, 2000. DOI: 10.1109/38.865875.

[15] J. Ahrens, B. Geveci, C. Law, C Hansen, and C Johnson, "36-paraview: An end-user tool for large-data visualization," *The visualization handbook*, vol. 717, pp. 50 038–1, 2005.

[16] J. Heer, S. K. Card, and J. A. Landay, "Prefuse: A toolkit for interactive information visualization," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005, pp. 421–430. DOI: 10 . 1145 / 1054972 . 1055031.

[17] J.-D. Fekete, "The infovis toolkit," in *IEEE Symposium on Information Visualization*, IEEE, 2004, pp. 167–174. DOI: 10.1109/INFVIS.2004.64.

[18] H. Childs, E. Brugger, B. Whitlock, *et al.*, "Visit: An end-user tool for visualizing and analyzing very large data," 2012.

[19] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[20] P. M. Valero-Mora, "Ggplot2: Elegant graphics for data analysis," *Journal of Statistical Software*, vol. 35, pp. 1–3, 2010.

[21] M. Mauri, T. Elli, G. Caviglia, G. Uboldi, and M. Azzi, "Rawgraphs: A visualisation platform to create open outputs," in *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*, ser. CHItaly '17, Cagliari, Italy: ACM, 2017, 28:1–28:5, ISBN: 978-1-4503-5237-6. DOI: 10.1145/3125571.3125585.

[22] J. Hynek, J. Kachlík, and V. Rusnák, "Geovisto: A toolkit for generic geospatial data visualization," in *VISIGRAPP (3: IVAPP)*, 2021, pp. 101–111. DOI: 10.5220/0010260401010111.

[23] J. D. Walker, B. H. Letcher, K. D. Rodgers, C. C. Muhlfeld, and V. S. D'Angelo, "An interactive data visualization framework for exploring geospatial environmental datasets and model predictions," *Water*, vol. 12, no. 10, p. 2928, 2020. DOI: 10.3390/w12102928.

[24] G. Andrienko, N. Andrienko, P. Bak, D. Keim, S. Kisilevich, and S. Wrobel, "A conceptual framework and taxonomy of techniques for analyzing movement," *J. Vis. Lang. Comput.*, vol. 22, pp. 213–232, Jun. 2011. DOI: 10.1016/j.jvlc.2011.02.003.

Author Biography

Moritz Zeumer received his M.Sc. degree in applied Computer Science in 2021 from the University of Applied Sciences and Arts Hanover. Since then, he is employed as a research scientist at the German Aerospace Center in the Institute for Software Technology. His research domain is visualization and visual analytics with a focus on human-computer-interaction.

Jonas Gilg received his M.Sc. degree in Applied Computer Sciences from the University of Applied Sciences and Arts, Hanover in 2019. Since then, he is employed as a research scientist at the German Aerospace Center in the Institute for Software Technology. His research domain is visual analytics with a focus on georeferenced data.

Dr. Pawandeep Kaur Betz received her doctorate in Data Visualization in 2021 from the Friedrich Schiller University of Jena, Germany. Since 2022, she is employed as a research scientist at the German Aerospace Center in the Institute for Software Technology. Her research domain is visual analytics, user-centric visualization designs and evaluation studies, automated insights, and visual interfaces for data management.

Prof. Dr. Andreas Gerndt received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for several companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a post-doctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008 to head a department at the German Aerospace Center (DLR). Since 2019, he is also Professor in High-Performance Visualization at the University of Bremen, Germany.