IVIS: Highly customizable framework for visualization and processing of IoT data

Lubomír Bulej, Tomáš Bureš, Petr Hnětynka, Václav Čamra, Petr Siegl, Michal Töpfer
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
Email: {bulej, bures, hnetynka}@d3s.mff.cuni.cz

***Abstract*—This tool paper presents the IVIS platform for processing and visualizing IoT and CPS data. The platform provides a web-based interface that allows both definition of complex visualizations and data processing jobs as well as exploring the data. Compared to the existing open-source and commercial offerings, IVIS follows a different model and focuses on flexibility. Instead of providing a complex administrative UI for creating visualizations by dragging and dropping components onto a dashboard, IVIS provides a set of JavaScript-based visualization components that are glued together using simple JavaScript code. Similarly, the data processing jobs can be defined using code in scripting languages, such as Python, which allows exploiting the wealth of existing libraries for numerical processing. This not only makes the definition of visualizations and data processing jobs much more expressive, but it also turns out to be significantly easier to use when building complex parametric visualizations—especially when they need to deal with many sensors. This proved to be crucial in deploying IVIS in a number of international research projects, because it enabled us to rapidly setup complex visualizations and data-processing tasks, catering to project- and partner-specific requirements.**

***Index Terms*—Visualization; data processing; customization; IoT.**

I. Introduction

Data processing and visualization have become an increasingly important part of Internet-of-Things (IoT) and Cyber-Physical Systems (CPS) because the insights gained from the data can enable a better understanding of the process being observed and allow taking the right actions. Furthermore, as these systems gradually become self-adaptive, self-optimizing, and self-learning, there is a huge amount of data on the internal working of such systems which is critical for assessing whether they work correctly, and for finding ways for further improvements. However, even in this case, the first step in making sense of a huge amount of data is to visualize it.

Data visualization and processing are not new in the IoT and CPS domains—there are several existing and mature frameworks backed by industry, and we provide an overview of the most important ones in Section II. However, we argue that there is still room for a framework that puts flexibility and a developer-oriented approach at the forefront to help with the high degree of customization required in IoT and CPS use cases. This allows developing and sharing common core functionality across different projects and yet allow customizing the framework for each respective project (including various project-specific visualization components, data connectors to import data from project partners, integration to other project partners' tools, etc.).

We also argue that project-level collaborations present a use-case that is somewhat alien to existing data-visualization frameworks, which focus on making it easy for an end-user to create visualizations by dragging/dropping common components onto a dashboard. In collaborative projects, partners typically do not set up visualizations on their own and instead cooperate with a partner responsible for the visualization framework. In such a context, there is little need for sophisticated and, from the development perspective, very costly graphical UI. Instead, it is more important that the partner responsible for visualization can rapidly set up rather complex and very customized dashboards (typically combining and aggregating data from several datasets). It is also necessary to allow these dashboards to be easily cloned and modified because the functionality required by different partners will be similar in its core, but very different on the surface.

In this paper, we describe an open-source data visualization and processing framework (called IVIS), which we have been developing and employing as the core for visualizations in various projects. In particular, we share our experience with employing IVIS in three different projects—focusing on (i) smart air quality, (ii) smart farming, and (iii) edge-cloud video processing.

II. Related work

Data visualization is not new, as is evident from the number of available production-level offerings. Grafana [1] is among the most popular ones. It is an open-source visualization and monitoring framework that can be connected with a multitude of data sources and provide attractive visualizations. A similar project, Kibana [2], is a visualization dashboard for ElasticSearch [3], a distributed, RESTful search and analytics engine suitable for large amounts of data. Chronograf [4] is yet another solution for visualization and monitoring, designed to visualize data from InfluxDB [5], a database designed for time-series data. The database is typically populated using Telegraf [6], a plugin-based system agent, which can collect data from a large number of different sources, including other databases.

There are also many smaller projects (such as FreeBoard, ThingsBoard, etc.) which provide similar functionality (custom data processing and visualization) in their scope, however, we do not aim to provide an exhaustive list and detailed comparison in this short paper. We aim to point out that in general, these projects allow users to quickly design monitoring dashboards using different kinds of standardized visualization widgets and charts (histograms, line graphs, pie charts, etc.). The focus of these projects is to provide users with a user-friendly GUI that enables interactive placement and configuration of visualization widgets on the dashboard.

While this is very useful for layman users and common visualizations, such as system monitoring dashboards, our experience from several projects was that when aiming at highly customized expert-grade visualizations, the interactive design soon becomes a limiting factor. Suddenly, it becomes very difficult to create charts that dynamically compute limits, conditionally display data based on summary statistics, or show min-max bands around a signal to give information about its fluctuations when displaying data aggregated over long periods. It turns out that there is a need for visualization frameworks that may be less forthcoming to layman users, but provide much more flexibility and enable rapid development of highly customized visualizations. The IVIS framework presented in this paper is one such framework.

III. Main features

A. IVIS architecture

An overview of the IVIS architecture is shown in Fig. 1. The system consists of a backend running on a server in the cloud (the top part of the figure) and a frontend running in a client's web browser (the bottom part of the figure).

The backend is responsible for managing the data and provides an API for the frontend as well as an interface for tasks (plugins) that execute (primarily analytics) jobs over the data. The data server receives master data from various sources (sensors) and stores them in a relational database. This data is then indexed by an ElasticSearch engine to enable fast searches and on-the-fly aggregations required by the front end. The frontend is primarily responsible for providing a view of the data through customized dashboards. Users can access the visualizations either directly, via the integrated web portal, or through a 3rd-party user interface, which embeds the visualizations from IVIS exported as an HTML iframe.

Technically, IVIS is developed in JavaScript (ES6) [7]. On the server, it runs within Node.js [8], and the frontend running in a web browser is built using the React.js framework [9]. The visualization components rely on the D3 [10] framework for creating charts and visualizations in SVG. The analytic plugins can be developed in any programming language (currently, Python is directly supported).

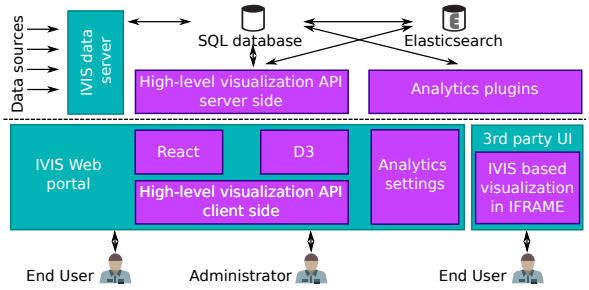

Fig. 1. IVIS architecture

B. Rapid development of visualizations

One of the key features of IVIS is that it allows rapid development of visualizations using simple code snippets. To this end, IVIS provides the concepts of visualization templates, panels, and workspaces. The template is the most important element of the visualization framework because it defines how to display data with a particular structure and does the actual rendering. In contrast, workspaces and panels are just containers.

Templates can be parameterized so that they can be reused with different data sources and in a different context. For example, a template can define a page with a date/time selector on top, a legend below, and a fixed set of line-charts displaying sensor data, e.g., temperature, humidity, and CO2 level. However, the particular data set to be displayed is not fixed and can be provided through template parameters—in our example, the template only captures the assumed structure of the data that will be passed to it, i.e., that the data to be visualized will contain separate sensor data with temperature, humidity, and CO2 level.

Template parameters can be arbitrary (not just a selection of sensor data) and can be arbitrarily nested, which means that it is possible to group related parameters and provide templates with tree-like object structures. For instance, in some of our visualizations, a template is parameterized by a two-dimensional selection of data sources along with a selection of color, ranges, etc. This makes the template a very flexible and powerful concept.

The values of template parameters (e.g., sensor data to be visualized) need to be set when an instance of a template is embedded in a panel, which stores the template's parameter settings. A panel is then accessible directly via an URL, or via a menu structure—here the panels are organized in workspaces, which simplifies navigation and allows grouping related panels.

Technically, a template consists of four parts: (1) *Template code (JSX).* Each template is a JavaScript module that exports a React.Component[1] responsible for rendering. The template code mainly deals with the composition and configuration of other React components in response to template parameters, producing a root component representing the visualization. (2) *Template style sheet (SCSS).* To customize the look-and-feel, the visual style

[1]https://reactjs.org/docs/react-component.html

of a template can be defined using Sassy CSS [11], a style sheet language that is compiled into CSS. The style sheet is loaded along with the template and allows defining not only the template-specific styles but also completely customizing the default styles provided by IVIS. (3) *Template parameter specification* comes in the form of a JSON object capturing the type and cardinality of parameter values, as well as the structure of the parameter object passed to a template. When instantiating a template, IVIS interprets the parameter specification and provides the user with a simple editor for each template parameter. (4) *Template assets* are images and other files required for the visualization.

To simplify the template code to the greatest extent possible (without sacrificing expressiveness), IVIS provides a predefined set of reusable React components which provide support for common types of charts (line/bar/pie/XY, histogram, scatter plot, heatmap, animated SVG) and basic interaction elements (date/time selector, chart legend). All these components have been specifically developed to seamlessly interact with the IVIS server, and their behavior can be customized via properties. This allows, for instance, to partially or fully override the rendering of the legend or the tooltip in a line chart, add fixed or dynamically computed reference lines to a chart, or dynamically change text and colors in an SVG-based image.

The use of JavaScript for template definition (instead of a visual UI) allows to easily create visualizations that work with many inputs or modify data on the fly (e.g., by completing missing values, shifting, rescaling, computing bounds)—all because iteration and conditional execution can be easily expressed in JavaScript. Also, which is not to be underestimated, visualizations can be easily cloned and customized (or stitched together from other visualizations) by simply copy-pasting fragments of the JavaScript code.

All this makes the development of complex visualizations substantially easier for anyone with a rudimentary knowledge of JavaScript, compared to building a complex parametric visualization using a visual editor. Our experience from several projects is that complex visualizations always require someone with at least some level of technical knowledge and that the requirement of basic understanding of JavaScript and the ability to assemble pieces of JavaScript (from examples of other visualizations) is typically not an obstacle.

A special feature of IVIS is that these templates are defined using an integrated web-based editor. Introducing a new template, therefore, does not require any modification to IVIS codebase and does not require the traditional development-test-deployment cycle (which can take hours even in a continuous integration/delivery pipeline, but invariably more). This enables very rapid development and deployment of visualizations.

Technically, to ensure sufficient performance, security, and to minimize the traffic between the server and the client, the templates are compiled on the server, bundled with the style sheet and (any) file assets, and served on-demand as a minified JavaScript to the client, i.e., only when the client wants to display a panel with the particular template. This also allows a user to ensure that the template is syntactically correct when developing it using the web-based editor provided by IVIS.

C. Rapid development of data processing tasks

In addition to receiving and storing master data from sensors, IVIS allows computing synthetic data derived from the master data. This enables a variety of data manipulation operations ranging from simple data conditioning (e.g., completing missing values, shifting, scaling) to complex filtering (e.g., smoothing data with low-pass or band-pass filter, accentuating fluctuations with high-pass filters), data aggregation, or forecasting and anomaly detection.

The synthetic data can be computed either offline or on-the-fly. On-the-fly computations are typically intended for lightweight tasks that do not require significant computing resources, i.e., simple aggregations and simple data conditioning. To this end, IVIS leverages the ElasticSearch backend which can perform on-the-fly computations very efficiently and in a very scalable manner. In particular, IVIS allows defining synthetic data fields via its UI, with the value of the field expressed as a code snippet written in the Painless scripting language [12]. IVIS passes these snippets to ElasticSearch when initiating a query on behalf of a particular visualization and lets ElasticSearch compute the synthetic data fields during the query.

Complex data processing tasks need to be performed offline. They can be defined in Python (with the help of various Python data processing libraries such as Numpy and Scipy) using a web-based UI provided by IVIS, similarly to how visualization templates are defined. This again enables very rapid development and deployment of data processing tasks. Offline data processing tasks are executed incrementally, only processing new data records since the previous computation. The results of the computation are stored in ElasticSearch, making the computed data readily available for use in visualizations.

IV. Main applications

We have successfully applied IVIS in several international projects. Each project was targeting a different domain and required different kinds of visualizations. Here we briefly review the use of IVIS in each of the projects.

ESTABLISH — The ESTABLISH [13] project has been successfully finished and defended at the end of year 2019. Research in the project focused on the use of environmental sensors (air quality, noise, heat, temperature) for improving the quality of life with respect to health. IVIS served as one of the core components through which the results of the project were demonstrated. In particular, IVIS was used to visualize readings from various sensors developed and utilized within the project. One of the most interesting visualizations is shown in Fig. 2, where

data from several sensors are combined with user feedback regarding satisfaction with perceived levels of different quantities (temperature, humidity, noise).

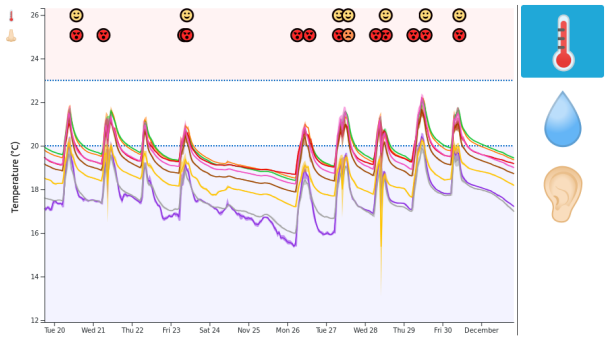

Fig. 2. IVIS in ESTABLISH

AFarCloud — The AFarCloud [14] project focuses on integration of cyber-physical and cloud-based systems in farming to improve efficiency, productivity, animal health, and food quality while reducing farm labor costs. In the project, we are (among other topics) working on models and methods for coordination of autonomous entities such as swarms of drones [15]. To enable rapid experimentation with different coordination strategies, we used IVIS to visualize the output of swarm simulations (shown in Fig. 3), providing immediate feedback on swarm behavior when following a particular strategy. In this particular case, we needed to develop a completely new style of visualization with support for continuous updating.

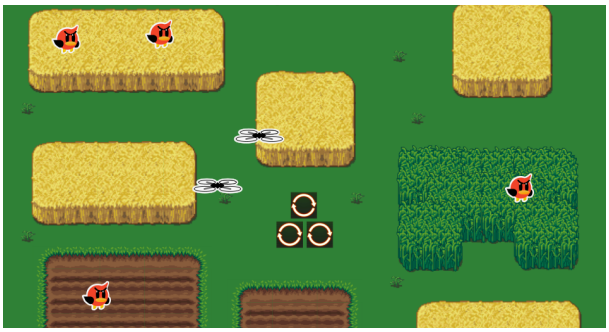

Fig. 3. IVIS in AFarCloud

FitOptiVis — The FitOptiVis [16] project focuses on development of a reference architecture for low latency image processing, along with methods and tools to support design-time optimization and runtime adaptation. In addition to using IVIS for storage and visualization of monitoring data from different systems, we used IVIS to visualize the image processing pipelines (an example shown in Fig. 4). The architecture visualizations are fully interactive, i.e., the layout of the components can be freely rearranged, and can be used to explore the architecture and navigate to visualizations of monitoring data associated with individual components. Both the components and their interconnections are described using a domain-specific language (developed within the scope of the project). To enable rapid turn-around when developing architectural descriptions, IVIS provides a web-based editor with syntax highlighting which allows users to create, visualize, and store architecture descriptions directly on the server.

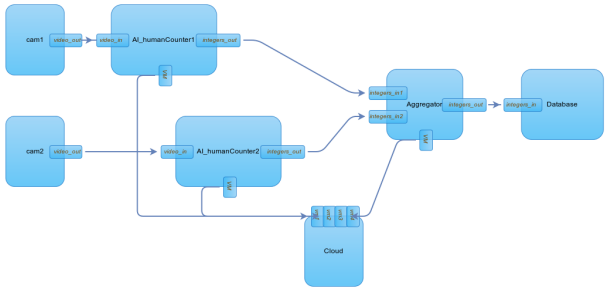

Fig. 4. IVIS in FitOptiVis

V. Conclusion

In this paper, we have presented IVIS, a highly customizable framework for visualization and processing IoT data, which focuses on flexibility to enable the development of customized visualization solutions. We have described the successful usage of IVIS in several research projects with diverse visualization requirements. The core of the IVIS framework is freely available at https://github.com/smartarch/ivis-core under the MIT license.

Acknowledgment

The research leading to these results has received funding from the ECSEL Joint Undertaking (JU) under grants agreement No 783162 and No 783221, and has been partially supported by project no. LTE117003 (ESTABLISH) from the INTER-EUREKA LTE117 programme by the Ministry of Education, Youth and Sports of the Czech Rep.

References

[1] "Grafana," https://grafana.com/.
[2] "Kibana," https://www.elastic.co/kibana.
[3] C. Gormley and Z. Tong, *Elasticsearch: the definitive guide*. O'Reilly, 2015.
[4] "Chronograf," https://github.com/influxdata/chronograf.
[5] "InfluxDB," https://www.influxdata.com/.
[6] "Telegraf," https://github.com/influxdata/telegraf.
[7] "ECMA-262 6th Edition, The ECMAScript 2015 Language Specification," https://www.ecma-international.org/ecma-262/6.0/index.html.
[8] "Node.js," https://nodejs.org.
[9] "React.js," https://reactjs.org/.
[10] "D3," https://d3js.org/.
[11] "SASS," https://sass-lang.com/.
[12] "Painless scripting language," https://www.elastic.co/guide/en/elasticsearch/painless/index.html.
[13] "Environmental Sensing To Act for a Better quality of Life: Smart Health," https://itea3.org/project/establish.html.
[14] "Aggregate FARming in the Cloud," http://www.afarcloud.eu/.
[15] P. Hnetynka, T. Bures, I. Gerostathopoulos, and J. Pacovsky, "Using Component Ensembles for Modeling Autonomic Component Collaboration in Smart Farming," in *Proceedings of SEAMS 2020 (Accepted), Seoul, Korea*, 2020.
[16] "From the cloud to the edge – smart IntegraTion and OPtimisation Technologies for highly efficient Image and VIdeo processing Systems," https://fitoptivis.eu/.