Calibration and Monitoring of IoT Devices by Means of Embedded Scientific Visualization Tools

Konstantin Ryabinin[1](✉), Svetlana Chuprina[1], and Mariia Kolesnik[2]

[1] Perm State University, Bukireva Str. 15, 614990 Perm, Russia
kostya.ryabinin@gmail.com, chuprinas@inbox.ru

[2] Perm Regional Museum/Branch Museum of Permian Antiquities, Monastyrskaya Str. 11, 614000 Perm, Russia
kolesnik.ma@outlook.com

Abstract. In the paper we propose ontology based scientific visualization tools to calibrate and monitor various IoT devices in a uniform way. We suggest using ontologies to describe associated controllers, chips, sensors and related data filters, visual objects and graphical scenes to provide self-service solutions for IoT developers and device makers. High-level interface of these solutions enables composing data flow diagrams defining both the behavior of the IoT devices and rendering features. According to the data flow diagrams and the set of ontologies the firmware for IoT devices is automatically generated incorporating both the data visualization and device behavior code. After the firmware loading, it's possible to connect to these devices using desktop computer or smartphone/tablet, get the visualization client code over HTTP, monitor the data and calibrate the devices taking into account monitoring results. To monitor the distributed IoT networks a new visualization model based on circle graph is presented. We demonstrate the implementation of suggested approach within ontology based scientific visualization system SciVi. It was tested in a real world project of an interactive Permian Antiquities Museum exhibition creating.

Keywords: IoT devices · Scientific visualization tools
Ontology engineering · Data flow diagrams
Firmware source code generation

1 Introduction

According to the elaborate overview made by Internet Society [20], the key issues of Internet of Things (IoT) are security, privacy, interoperability, legislative framework, emerging economy and development. In the same time, the level of development automation tools for IoT is not high enough. As Texas Instruments Inc., one of the biggest manufacturers of IoT-related microelectronics, pointed, "IoT needs to be made easy for inexperienced developers" [27]. Moreover, nowadays the IoT developers community grows exponentially. Survey made by Tractica [28] shows that many enthusiasts become IoT developers. They create Smart Systems for homes, scientific or artistic exhibitions, and just for fun.

Due to the lack of widely supported standards [20], development of IoT devices often requires special software for designing, debugging and steering. Consequently, high-level software tools for building and managing IoT applications in a uniform way are demanded.

Growing IoT networks embrace more and more different fields of life and thereby produce huge quantity of heterogeneous data gathered by connected things. IoT systems have to be intelligent enough to make these data observable for humans, avoiding information overflow and ensuring comfortable cohabitation of humans and smart objects. The overflow problem can be solved by advanced semantic filtering of data performed inside the IoT network.

High complexity and granularity of IoT networks require powerful and ergonomic analytics tools. Especially it relates to the role of Big Data visual analytics in IoT. Modern visual tools for IoT should enable the user not only to monitor device data, but also control the behavior of smart objects and configure/reconfigure their network.

When it comes to IoT devices with different kinds of sensors, calibration and monitoring tasks become complicated. Calibration has many forms and consists in adjusting the parameters. These parameters rule the device to provide an accurate measurement according to known model, ground-truth data from laboratory instruments or already calibrated devices. Normally device manufacturer performs the basic calibration, but in some specific cases user steering is also required. For example, user may need to adjust the sensitivity of light detector to adapt it to concrete ambient lighting conditions. This kind of "user-side" calibration is often empirical and thereby requires the tools to adequately data monitoring, analyzing and predicting the adjustment results.

We propose to tackle the above mentioned IoT challenges using a single software platform based on the scientific visualization methods, corresponding rendering techniques and semantic filtering. Reusing of scientific visualization tools enables graphical representation and visual analysis of data generated by IoT devices and light-weight consumer robotic systems [23]. In this paper we focus on the monitoring and user-side calibration issues.

During the previous research work we developed a concept of a knowledge-driven multiplatform adaptive scientific visualization system (SVS), which can be integrated with third-party data sources by means of high-level tuning without source code modifications. We have implemented this concept in the system called SciVi [22]. The behavior of this system is driven by the ontologies [9] included in its knowledge base that describes the features of data sources, supported data filters, visual objects and graphical scenes. Thereby SciVi can be easily extended to cover new visualization techniques and set up for communication with arbitrary data sources including software solvers and hardware devices.

The aim of the paper is to present the extension of SciVi by self-service tools for automated embedding in IoT devices. This extension enables IoT device calibration and monitoring supported by high-quality interactive visualization. We demonstrate the results by an example of automated creation of interactive IoT-based exhibits in the Museum of Permian Antiquities (Perm, Russia).

2 Key Contributions

In this paper we describe the new capabilities of multiplatform SVS SciVi (version 3.0). The conducted research and corresponding development led to the following key results:

1. Proposed the ontology-driven approach for high-level adaptable monitoring and calibration of the heterogeneous IoT devices by means of scientific visualization and semantic filtering.
2. Proposed the circle graph-based model to graphical depiction of IoT network.
3. Implemented new SciVi high-level tools based on suggested approach to automate the process of SVS embedding into IoT device firmware to enable built-in visualization and steering.
4. Suggested the concept of interactive IoT-based museum exhibition, which uses implemented tools. Demonstrated this concept by 3D-reconstruction of Permian synapsid *Dimetrodon grandis* that exhibits the latest paleontological research results.

3 Related Work

3.1 IoT Development Tools

Since IoT is an exponentially evolving field of computer science and engineering [28], there are literary hundreds of related development tools [17]. They include software and hardware development platforms, middleware, operating systems, integration software, stacks of technologies and protocols. But most of these tools require very deep skills either in programming or in electronics, and the modern trend is to create high-level solutions available for beginners, not only for experts [27]. Such kind of solutions reduces IoT development time and increases productivity, which enables to involve more people into IoT community.

The most popular high-level state-of-the-art software solutions are Processing [18], Blynk [7], XOD [2] and NEXCOM IoT Studio [14]. The first system provides the high-level programming language and the set of libraries to simplify the development of light-weight electronic devices in terms of handling, transforming and monitoring data gathered. Being a powerful programming toolset, it still requires some additional developer's skills and provides no visual programming capabilities. Blynk is a visual programming tool enabling high-level control over the firmware logic of electronic device. It is very ergonomic, but lacks complex visualization capabilities, which are sometimes needed for data monitoring. XOD and NEXCOM IoT Studio use data flow [13] visual programming paradigm enabling user to compose firmware logic as a data transformation diagram. Data flow based programming is very intuitive and its effectiveness has been proven in many different application domains. However as well as Blynk, XOD and NEXCOM IoT Studio focus on the device logic providing just a basic dashboard-based data visualization.

While the mentioned solutions perfectly suite for solving typical IoT tasks, they cannot be easily extended for special cases like, for example, creating IoT-powered interactive museum exhibitions (see Sect. 6).

3.2 Ontology Engineeing Applied to IoT

Knowledge-driven approach based on ontology engineering methods is agile and handy for creating of complex systems [6]. Semantic Web technologies were proven to be suitable for IoT many times. One of the first research works in this field was made by Gyrard et al. [10]. As a result of this work best practices of ontology engineering usage in a context of IoT are defined.

The state-of-the-art ontology-based solutions are described in [3,25,26]. These works leverages ontology-based methods to different aspects of IoT, including network representation, security and interoperability issues. The work [11] describes ontology-based approach to create middleware for IoT. The work [8] describes powerful state-of-the-art ontology-based library code generator OLGA, which is suited for IoT firmware programmers. Finally, W3C Consortium published reusable ontology IoT-Lite of resources, entities and services related to IoT [29].

Despite many successful results, the ontology-driven IoT monitoring and calibration issues are still open. According to our knowledge there are no popular state-of-the-art solutions leveraging ontology engineering for visual analytics of IoT-generated data (see Sect. 5).

3.3 Methods to Represent IoT Network

While IoT networks become very sophisticated, visual analytics tools become highly demanded. These tools enable developers and users to inspect and manipulate the networks' structure as well as to detect problems and invent optimizations. State-of-the-art solutions for monitoring IoT networks unstructured graphs [15] or 3D models of rooms and objects where IoT devices are located [16].

However, it's not always easy enough to analyze unstructured graph or full 3D environment. Sometimes more informative is a concise structured representation. This is why we propose graphs of circular structure [24] to represent IoT device networks (see Sect. 7).

3.4 Scientific Visualization Tools

Previously, we analyzed state of the art of scientific visualization software including capabilities of different popular stand-alone systems like TecPlot, Origin, EasyPlot, IRIS Explorer, Surfer, Grapher, Voxler, Hesperus, ParaView, Avizo, etc. and scientific visualization libraries and frameworks like OpenDX, VTK, VizIt, ScientificVR, etc. [21]. These software solutions provide high-quality visualization, a lot of different tools and useful functions, but have some drawbacks, which may be crucial in some particular cases:

1. Standardization of input. Almost all popular SVS and libraries require input data in particular standard formats and have no high-level tools for integration with arbitrary third-party data sources.
2. Inability of feedback. Usually there are no means to set up control over the data source from the visualization system and the user can just view the data but cannot steer the process of data generation.
3. Complexity of extending. Most of the SVS and libraries can be extended with the new graphical capabilities by source code changing only. Extending and tuning the SVS by plugins requires programming skills and is complicated for the beginners.
4. Lack of portability. Popular visualization systems are implemented either for desktop computers or mobile devices, not for both.

To tackle the above challenges, we propose the ontology-driven SVS SciVi [22] (see Sect. 4). Besides programmers, there are two types of SciVi users: ontology engineers and casuals. The main goal of ontology engineers is to extend the SciVi knowledge base and thereby provide new capabilities for casuals. We demonstrated how we extended SciVi to create IoT-based exhibition of *Dimetrodon grandis*. The casual users in this case are IoT device markers.

3.5 Modern Reconstruction of *Dimetrodon Grandis*

IoT-based solutions have big potential not only for Smart Systems like Smart Home, but also for interactive exhibitions and installations suitable for museums, scientific art shows, etc. To demonstrate this idea we created interactive exhibition of *Dimetrodon grandis* Romer and Price, 1940, early Permian synapsid lived more than 290 million years ago.

According to the current paleontological knowledge, dimetrodon had a sail on its back with a tight network of blood vessels. This sail served to effectively heat the body by the sunrays. To demonstrate this concept in the interactive way we created animated 3D model of dimetrodon using Blender 3D. We took into account the latest reconstruction research results published by Hartman [12], including sail shape, back curvature and high walk pose.

We uploaded this model to SVS SciVi and created IoT device that detects light direction of real light sources and visualizes how dimetrodon rotates its sail to the light rays (see Sect. 6).

4 Key Concepts of SVS SciVi

The detailed description of SVS SciVi including its architecture and key features ensuring efficient and high-quality rendering on both desktop computers and mobile devices can be found in [22]. Its essential part is the knowledge base containing 3 ontologies:

1. $\overline{L}$ that describes input/output (I/O) statements of programming languages. According to this ontology the regex-based parser is automatically generated, which extracts the related structures of I/O data from the solver's source code. This mechanism is used to automate the process of integration of SciVi with third-party solvers.
2. $\overline{F}$ that describes semantic filters used to optional preprocessing the data before visualization. For example, the described filters can sort the data, cluster them, reduce their amount according given criteria, perform mathematical transformations, etc.
3. $\overline{U}$ that describes supported visual objects and graphical scenes. For example, 2D charts and 3D models are available.

Developer or knowledge engineer can add new filters, graphical objects and scenes to SciVi by extending the ontologies $\overline{F}$ and $\overline{U}$ without core's source code modification.

To define the internal SciVi rendering pipeline we use the data flow diagrams [13]. We developed the high-level graphical editor, which enables the easy way to design data flow diagrams and thereby to describe the data retrieving, transformation and rendering. This editor provides a palette of available filters, graphical objects and scenes. The palette is automatically built according to their descriptions in the ontologies $\overline{F}$ and $\overline{U}$. User can add new diagram nodes from this palette and connect their I/O sockets defining the flow of data. Starting from the data source node, user can define the sequence of filters, connect them with visual objects and finally compose the graphical scenes.

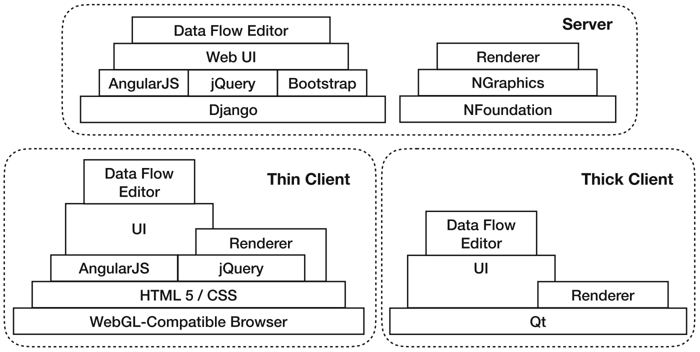

Fig. 1. Stack of software used in SciVi server, thin and thick client

SVS SciVi uses a layered architecture to achieve needed abstraction level from the platform it runs on. The stack of software used in SciVi is shown in the Fig. 1. Thin client is written in JavaScript, thick client's logic is almost entirely implemented in C++ and server is based mainly on C++ and Python; its Web interface is written in JavaScript. Server and thin client share the common code of data flow editor based on AngularJS framework. Thick client has its own data flow editor code to make it more efficient using native drawing API. NFoundation and NGraphics libraries are written in C++ (product of Perm IT-company Nulana Ltd., one of the paper's authors is affiliated with). It provides abstraction layer from the operating system and OpenGL/Direct3D low-level rendering API to enable server-side rendering if needed.

The renderers of client and server are highly configurable thanks to ontology knowledge base, which includes the corresponding links to original and third-party rendering libraries (for example, PixiJS, D3.js and Three.js in the thin client, Qt charts and Assimp in the thick client, etc.).

5 Embedded Scientific Visualization Tools

The aim of this work is to provide self-service solutions for calibration and monitoring of IoT devices by means of embedded scientific visualization tools. These tools make it possible to implement "in situ visualization" [19] for IoT devices. SciVi is capable to connect to arbitrary data source including different electronic devices. The device maker has to set up the SciVi server as intermediate software between the IoT devices and the visualization client (desktop computer, mobile device or browser). This approach is flexible enough, but sometimes it must be simplified for the sake of efficiency. For example, in the museum with interactive stands, the visitor can use his/her mobile device to handle these stands and get corresponding visualization. For this purpose, it's necessary to connect to the interactive stand via browser over WiFi or some similar HTTP-compatible technology and get the data displayed without any additional software. In the other case, when some IoT device is used, it's necessary to calibrate this device and to visualize the data it produces for monitoring purposes. In the last case the direct connection between mobile and IoT devices is preferable.

To achieve this, we propose original tools for automated firmware generation for the IoT device taking into account its specifications. Firmware incorporates both the program code of device logic (called "embedded host"), and the program code of visualization Web-server (called "embedded server"). The thin client is responsible for rendering. The embedded server enables thin client connections, obtains the data from embedded host, transmits them to the client and receives the control commands including the calibrating. The client performs scene rendering and provides the calibration graphical user interface. The "traditional" SciVi assumes that the visualization server is more powerful than the client and specific data processing including some rendering steps can be performed on the server side. But in the case of IoT device firmware the contrary is assumed: the client device should perform all the processing and visualization, because the server platform normally has limitation both in clock frequency and in memory. This firmware is generated by the SciVi server and can include as many visualization routines, as the IoT device memory volume enables.

As mentioned above, previous version of SciVi contained three ontologies in its knowledge base: $\overline{L}$, $\overline{F}$ and $\overline{U}$. To provide the automated generation, we propose to extend the SciVi knowledge base with the ontology $\overline{C}$. This ontology describes different electronic components such as controllers, chips and sensors as well as associated firmware code.

For example, in the Fig. 2 the fragment of ontology $\overline{C}$ describing analog-to-digital converter (ADC) of ESP8266 controller is shown. This popular WiFi-compatible controller is widely used for IoT purposes [4]. The ontology $\overline{C}$ is used in the similar way as ontologies $\overline{F}$ and $\overline{U}$: it supports the data flow diagram editor's functioning and is intended to provide the code fragments necessary for firmware generation. All the four ontologies in SciVi knowledge base have the common part that is a description of supported data types, so they all may be used together at the same time.

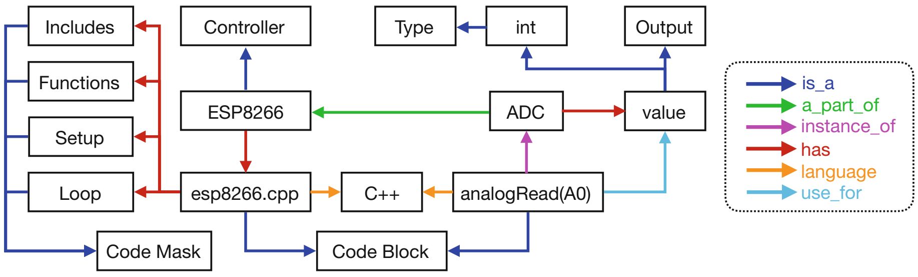

Fig. 2. Fragment of electronic components ontology from the SciVi knowledge base (Color figure online)

The ontologies are represented in standard OWL format. Short code snippets (one-line expressions) are stored directly in the ontology, for example, the call of ADC "analogRead(A0)" in Fig. 2. Larger code blocks are saved in separated files, for example, the firmware template "esp8266.cpp". Code blocks can have masks like "Includes", "Functions", "Setup" and "Loop" of "esp8266.cpp", which are filled automatically by related code during firmware generation. All of these code fragments are stored in the code repository. $\overline{F}$, $\overline{U}$ and $\overline{C}$ ontologies play the role of its semantic index, while the data flow diagram defines the interconnections between the different entities described by these ontologies.

The firmware generator is a part of SciVi server. It is written in Python and executed on the basis of Django, see Fig. 1. Its input is data flow diagram and the type of controller. The generator traverses the data flow diagram and generates the firmware code for chosen controller using the related code fragments according to those parts of ontologies, which are referenced from the data flow diagram nodes. Normally C++-based embedded host and embedded server are created as well as JavaScript-based client. The client is then transmitted to the user via HTTP and communicating with the server via WebSocket. In case of

ESP8266 (see Fig. 2) the `*.ino` file is generated (so-called "sketch") which can be uploaded to the ESP8266 with the help of Arduino IDE.

6 Evaluation

As a feature-reach use-case to evaluate the proposed approach we demonstrate the creation of a light direction sensor based on ESP8266 controller. This sensor needs feedback for calibration. It detects the average direction to an external light source and uses it to direct the light on the virtual scene. The sensor consists of 3 photo resistors VT90N2 placed at the vertices of an equilateral triangle. Minimal resistance of VT90N2 is reached when the light direction is collinear with the normal $\boldsymbol{n}$ of its surface. The surfaces of resistors are oriented to be 45° to the ground. Thereby the normal vectors $\boldsymbol{n}_1$, $\boldsymbol{n}_2$, $\boldsymbol{n}_3$ of the resistors' surfaces build the basis of the vector space. In this space the direction from the sensor to the light source can be expressed as $\boldsymbol{l} = u_1\boldsymbol{n}_1 + u_2\boldsymbol{n}_2 + u_3\boldsymbol{n}_3$, where u_1, u_2, u_3 are the quotients proportional to the voltage on the corresponding photo resistors. Data flow diagram of the light direction sensor is shown in the Fig. 3.

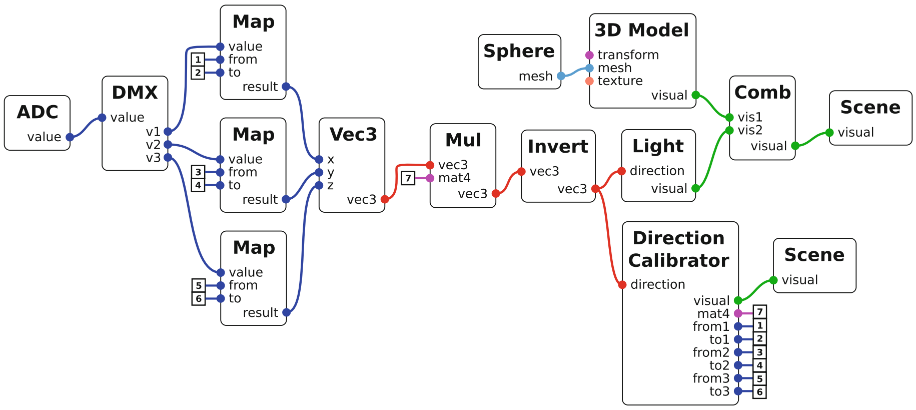

Fig. 3. Data flow diagram for light direction sensor

The ESP8266 controller has only one 10-bit ADC channel depicted on the diagram by "ADC" node with single output. This is why logical demultiplexor (DMX) is used to separate signals from the photo resistors. It corresponds to the analog multiplexor 74HC4051 in the device schematics. The theoretical range of the values obtained from ADC is $[0; 1023]$, however in practice the voltage measured on the resistors does not accurately match that range having some "padding" from the lower and upper boundary. To compensate this, cutting should be performed. The nodes called "Map" cut off their input values according to the "from" and "to" inputs and map them to $[0; 1]$. The cutting values should be calibrated as they depend on the photo resistors and on the ambient lighting conditions.

The mapped values are then assembled into 3-dimensional vector using node "Vec3", which is multiplied by the 4×4 matrix using node "Mul" to match the virtual scene's basis and the actual orientation of the sensor. To perform multiplication, vector is extended by $w = 1$ and then trimmed back to 3 components. The matrix used as input for the "Mul" node contains rotation and should be calibrated.

The direction of the vector assembled is inverted by the node "Invert" to get the light direction. The result is used to direct the virtual light source depicted with the node "Light", which is then added to the graphical scene. 3D model with sphere as a mesh is added to reflect the light.

To enable device calibration there is special graphical object "Calibrator", which takes the light direction as input and produces corresponding visual object as output for related graphical scene. Calibrator renders coordinate axes and explicitly visualizes the vector of light direction as an arrow in the scene space. It also enables user to rotate the coordinate system and therefore change the rotation matrix for the basis, as well as provides graphical interface to change the ranges for 3 mappings of values obtained from ADC. The output of the calibrator is connected to the "Mul" and "Map" nodes, building the *cycle* in the data flow diagram. While traditional data flow diagrams are acyclic, we need to support feedback to enable calibration. To keep the diagram clean and readable (as well as planar in terms of graph theory) we indicate feedback links with the special numbered markers (see Fig. 3) and not by arcs.

The two "Scene" nodes represent two different graphical scenes, which can be accessed by different URLs determined in the nodes' settings. Switching between these two scenes, user can calibrate the device and view the actual lighting simulation. However, this means the calibration values should be stored when calibrator is shown and reused when the 3D model is rendered. To achieve this, we departed from the traditional data flow principal of "freedom from side-effects" [13] and implemented feedback via global variables. While this breaks a traditional data flow concept, it ensures more flexibility and enables easy and readable representation of calibration actions. We still call the result diagram "data flow", because it still represents the path of data through the preprocessing and rendering pipeline. With current enhancements the pipeline has global mutable state, which can be changed by the nodes in runtime enabling the calibration capabilities.

We used the developed device in the interdisciplinary educational project organized by the Museum of Permian Antiquities. The project is designed mainly for middle and senior schoolchildren. Its idea is to create an interactive exhibition of *Dimetrodon grandis* mentioned in Sect. 3.5.

During the educational lesson the attending schoolchildren assemble the light direction sensor based on ESP8266 controller and create the logic of its firmware as a data flow diagram in SciVi. The idea is to tie the angular position of dimetrodon 3D model to the light direction. Next, SciVi generates the firmware code and the attendees calibrate the device they created. As a result, the rendered dimetrodon automatically moves its body to keep the sail nearly perpendicular to the light direction. The aim of this educational project is to present the modern paleontological knowledge to the schoolchildren in a game form as well as to show the alloy of paleontology, biology, physics, robotics and computer science.

We used skeletal animation supported by Three.js for scientifically accurate rendering of the dimetrodon. To ensure flexibility and low latency of data transmission the communication between client and embedded server based on HTTP and WebSocket is used.

7 Graph-Based Visualization of IoT Network

Perm Regional Museum has many branches including historical and paleontological ones, which are territorially distributed. Going further with the idea of IoT-based interactive exhibitions we plan to build the common environment across different branches. The result network may have complex topology establishing connections not only inside single museum branch over local network, but also between different branches over the Internet. In this case the visual analysis of the network would be needed.

We propose new clear and concise visual model named SciVi::CGraph that accelerates a common understanding of the huge amount of data obtained from distributed IoT networks as well as networks structure. SciVi::CGraph is based on the traditional circle graph [1]. Compared to circle graph SciVi::CGraph has multi-level adaptable ring scale around the graph. This scale enables to demonstrate the hierarchical grouping of data, which is adequate for multidimensional data representation. We have used SciVi::CGraph as a visual model in a real-world project to explore the Machine Learning results of social network processing within a remit of the project titled "Socio-Cognitive Modeling of Social Networks Users Verbal and Non-Verbal Behavior Based on Machine Learning and Geoinformation Technologies". The research results of this project have proved that suggested model is very useful for visual analytics goals and enables to evaluate different data processing algorithms. The reported study is supported by Ministry of Education and Science of the Russian Federation, State Assignment No. 34.1505.2017/PCh (Research Project of Perm State University, 2017–2019).

In a context of designing a Smart Museum [5] SciVi::CGraph can be used to visualize the results of IoT networks monitoring including museum visitors activity monitoring. In this case the suggested model shows the data with respect of the distributed IoT network structure. The devices are visualized taking into account their location, so neighbor devices are grouped together in the graph. Figure 4 demonstrates the distributed IoT network structure. The connections are represented by quadratic parabolas. The traffic volume is mapped to the thicknesses of the graph arcs. The kind of devices such as sensor, display, actuator, etc. is mapped to the color of graph nodes. The power consumption of devices is shown by the histogram drawn as nodes' background. The location hierarchy (halls inside buildings) is represented by the ring scale around the graph. Support of this kind of visualization in SVS SciVi required the expansion of the ontology $\overline{U}$.

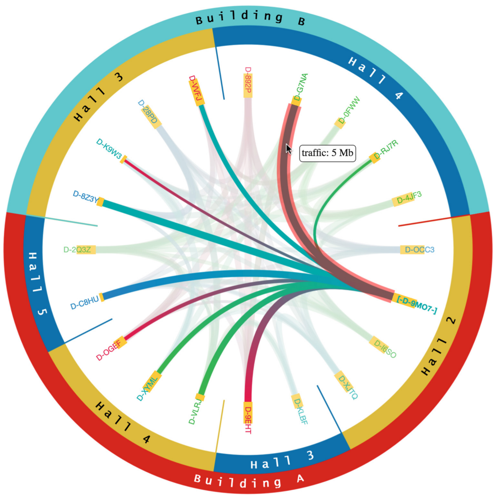

Fig. 4. SciVi::CGraph representation of distributed IoT network structure (Color figure online)

8 Conclusion

In this paper we describe the new version of adaptive SVS SciVi (v3.0) enriched with the high-level features enabling firmware generation for IoT devices. In the context of SVS, IoT devices are treated as solvers generating data to be visualized. The new features as all the previous extensions are added to SciVi in a uniform way by means of ontology-based engineering methods. For this, we extended the SciVi ontological knowledge base with the ontology of electronic components that describes different controllers, sensors and chips and related program resources needed to generate the firmware. This ontology together with ontologies of semantic filters, visual objects and graphical scenes allows the device makers to build data flow diagrams, which describe internal logic of IoT device behavior as well as the calibration and monitoring algorithms.

Currently the calibration is implemented via feedback functionality. The user can tune the device manually according personal needs by sending related control commands while observing the visual representation of device data. But in general the knowledge-driven approach to semantic filtering enables to define automatic calibration algorithms, for example, drift compensation function. To do this within SciVi it is enough to add to ontology knowledge base the corresponding semantic filter that will be automatically applied to raw data obtained from the device.

Upgrade of SciVi knowledge base allows this SVS to play a new role as visual environment for IoT device programming. The paper highlights the testing of the SciVi 3.0 new features by an example of creating the *Dimetrodon grandis* interactive exhibition.

In the future, we plan to build more complex Smart Museum exhibitions using the interconnected IoT devices. To be able to visually inspect distributed IoT networks we propose the new kind of concise visual representation based on circular graph. Also we plan to enrich the IoT ecosystem with augmented reality (AR) capabilities, for example, to generate ontology-based AR-assistant that helps assembling complex IoT devices or networks.

References

1. Ageev, A.: A triangle-free circle graph with chromatic number 5. Discret. Math. **152**, 295–298 (1996). https://doi.org/10.1016/0012-365X(95)00349-2
2. Allan, A.: XOD-a new visual programming language for Arduino. Hackster.io (2017). https://blog.hackster.io/xod-a-new-visual-programming-language-for-arduino-7bc911cdb27. Accessed 31 Mar 2018
3. Ayele, G.: Semantic description of IoT security for smart grid. Master's thesis. University of Agder (2017)
4. Brown, E.: Linux and open source hardware for IoT. Linux.com (2016). https://www.linux.com/news/linux-and-open-source-hardware-iot. Accessed 31 Mar 2018
5. Chianese, A., Piccialli, F.: Designing a smart museum: when cultural heritage joins IoT. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 300–306 (2014). https://doi.org/10.1109/NGMAST.2014.21
6. Chuprina, S., Nasraoui, O.: Using ontology-based adaptable scientific visualization and cognitive graphics tools to transform traditional information systems into intelligent systems. Sci. Vis. **8**(1), 23–44 (2016)
7. Doshi, H.S., Shah, M.S., Shaikh, U.S.A.: Internet of Things (IoT): integration of Blynk for domestic usability. Vishwakarma J. Eng. Res. **1**(4), 149–157 (2017)
8. El Kaed, C., Ponnouradjane, A.: A model driven approach accelerating ontology-based IoT applications development. In: CEUR Workshop (2017). http://ceur-ws.org/Vol-2063/sisiot-paper4.pdf. Accessed 31 Mar 2018
9. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993). https://doi.org/10.1006/knac.1993.1008
10. Gyrard, A., Serrano, M., Atemezing, G.A.: Semantic web methodologies, best practices and ontology engineering applied to Internet of Things. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT) (2015). https://doi.org/10.1109/WF-IoT.2015.7389090
11. Hachem, S., Teixeira, T., Issarny, V.: Ontologies for the internet of things. In: 8th Middleware Doctoral Symposium (MDS 2011) (2011). https://doi.org/10.1145/2093190.2093193
12. Hartman, S.: Taking a 21st century look at Dimetrodon. Scott Hartman's Skeletal Drawing.com (2016). http://www.skeletaldrawing.com/home/21stcenturydimetrodon. Accessed 31 Mar 2018
13. Lee, B., Hurson, A.: Issues in dataflow computing. Adv. Comput. **37**, 285–333 (1993). https://doi.org/10.1016/S0065-2458(08)60407-6

14. NEXCOM: IoT Development Tool Selection Guide (2016). http://ebook.nexcom.com/Catalog/2016_IoT_Development_Tool_Selection_Guide/IoT_Development_Tool_Selection_Guide_opf_files/pdfs/IoT_Development_Tool_Selection_Guide.pdf. Accessed 31 Mar 2018
15. nuPSYS: nuGEN^{TM} 3D visualization of IoT networks. nuPSYS (2017). http://www.nupsys.com/nugen.html. Accessed 31 Mar 2018
16. Puri, D.: Coping with IoT data overload? Try 3D visualization. NetworkWorld (2016). https://www.networkworld.com/article/3140712/internet-of-things/coping-with-iot-data-overload-try-3d-visualization.html. Accessed 31 Mar 2018
17. Rahul: 100 IoT open source development tools and resources. Internet of Things Wiki (2016). https://internetofthingswiki.com/100-iot-open-source-development-tools-and-resources/422/. Accessed 31 Mar 2018
18. Reas, C., Fry, B., Maeda, J.: Processing: A Programming Handbook for Visual Designers and Artists. The MIT Press, Cambridge (2007)
19. Rivi, M., Calori, L., Muscianisi, G., Slavnic, V.: In-situ visualization: state-of-the-art and some use cases. In: PRACE (2011). http://www.prace-ri.eu/IMG/pdf/In-situ_Visualization_State-of-the-art_and_Some_Use_Cases-2.pdf. Accessed 31 Mar 2018
20. Rose, K., Eldridge, S., Chapin, L.: The internet of things: an overview. Technical report. The Internet Society (ISOC) (2015). https://www.internetsociety.org/resources/doc/2015/iot-overview. Accessed 31 Mar 2018
21. Ryabinin, K.: Methods and means for the development of adaptive multiplatform systems to visualize scientific experiments. Ph.D. thesis, Perm State University (2014)
22. Ryabinin, K., Chuprina, S.: Development of ontology-based multiplatform adaptive scientific visualization system. J. Comput.Sci. **10**, 370–381 (2015). https://doi.org/10.1016/j.jocs.2015.03.003
23. Ryabinin, K., Chuprina, S.: Tackle lightweight hardware robotic devices data monitoring problems by means of scientific visualization systems. In: 27th International Conference on Computer Graphics and Vision GraphiCon 2017, pp. 142–146 (2017)
24. Ryabinin, K., Baranov, D., Belousov, K.: Integration of scientific visualization toolset SciVi with information system semograph. In: 27th International Conference on Computer Graphics and Vision GraphiCon 2017, pp. 138–141 (2017)
25. Serrano, M., et al.: IoT semantic interoperability: research challenges, best practices, recommendations and next steps. Technical report, European Research Cluster on The Internet of Things (2015). http://www.internet-of-things-research.eu/pdf/IERC_Position_Paper_IoT_Semantic_Interoperability_Final.pdf. Accessed 31 Mar 2018
26. Seydoux, N., Drira, K., Hernandez, N., Monteil, T.: IoT-O a core-domain IoT ontology to represent connected devices networks. In: Blomqvist, E., Ciancarini, P., Poggi, F., Vitali, F. (eds.) Knowledge Engineering and Knowledge Management, vol. 10024, pp. 561–576. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49004-5_36
27. Texas Instruments Inc.: The top 6 challenges facing IoT - and how we are overcoming them. TI E2E^{TM} Community (2016). http://e2e.ti.com/blogs_/b/thinkinnovate/archive/2016/02/02/the-top-6-challenges-facing-iot-and-how-we-are-overcoming-them. Accessed 31 Mar 2018
28. Tractica: Consumer robotics. Technical report. Tractica (2017). https://www.tractica.com/research/consumer-robotics/. Accessed 31 Mar 2018
29. W3C: IoT-lite ontology. W3C (2015). https://www.w3.org/Submission/iot-lite/. Accessed 31 Mar 2018