EFFICIENT ENGINEERING OF IIOT SYSTEMS BASED ON INTEROPERABILITY STANDARDS

Mario Thron, Holger Zipper, Thomas Bangemann

Institut f. Automation und Kommunikation e.V., Werner-Heisenberg-Str. 1, 39106 Magdeburg, Germany

Abstract

Implementation of interoperable devices, as they are used in IIoT and Industry 4.0 networks, is currently a time consuming and difficult work. This article describes methods and technologies for improving the efficiency of this process. The approach proposes to define data structures, groups of equipment component variables and later on component descriptions by use of formal languages. Further on, forms-based on-line editing of specifications is proposed instead of using word processing software. Code generators are then introduced to transform the formal descriptions into parts of the firmware code of the IIoT devices. The proposed approach provides support for standardization organizations and equipment manufacturers.

Keywords: *IIoT, Industry 4.0, Arrowhead Framework, production equipment, automation component, code generation*

1. INTRODUCTION

The market of industrial products has shifted in last years from mass production to production of more individualized products and this trend continues. That situation required the production systems to become more flexible. Suppliers of industrial production equipment reacted with the development of production components, which can be connected and controlled easily via industrial communication systems. Moreover, components have been designed, which can emerge and are dynamically allocated in a production process, even if that process spans over several sites of a company or over several companies. The resulting new possibilities of organizing production processes is called the 4th industrial revolution (Industry 4.0 – I4.0) or in other context Industrial Internet of Things (IIoT).

The elements of an industrial production system are called production equipment components or simply components. They can provide the possibility to communicate with other components via fieldbus systems defined in IEC61158 (2019) or increasingly via Internet-based communication protocols like HTTP-based REST services (Fielding 2014, section 4) or the MQTT standard (OASIS 2019).

Efficient interaction between production components requires exchange of data between components. A data sink component has to be able to interpret at least parts of the data sent by or requested from data source components. Therefore, interface or message descriptions are necessary for implementation of data sink components.

A modular design of production systems leads to excellent adaptability regarding the needs of several customers of the production system. But a modular design requires exchangeable components in order to stick with the original operation strategies. Especially the use of the original control programs requires exchangeable components to provide and consume the same data structures and encodings. Standardization organizations (like VDMA, PROFIBUS & PROFINET International (PI), CAN in Automation (CiA) or IEC) define such data structures, which describe e.g. sensors, fluid power technologies or electrical drives.

Currently, standards are often written in paper-like form e.g. for electrical drives (IEC 61800-7-201 2015) or for fluid power technology (CiA 408 2017). More recently projects have been initiated in order to describe production components more formal, e.g. the VDMA, a non-profit organization of machine manufacturers in Germany, provides OPC UA Companion Specifications, which implies interface descriptions as formal XML files too. But those formal descriptions are limited to describe component

interfaces for a specific communication system (e.g. to OPC UA). A communication system agnostic description technology would allow to transform specifications either into several communication system specific interface descriptions or directly into source code for data sources and sinks.

Manufacturers of components want to have insight into the specification process to have influence on the content and quality of those specifications. In case of formal specifications, it is a common approach in the information technology domain (IT) to use version control systems and public repositories for collaborative source code development, which could also be used for specification of production component interfaces. Examples are GitHub (https://github.com) or SourceForge (https://sourceforge.net).

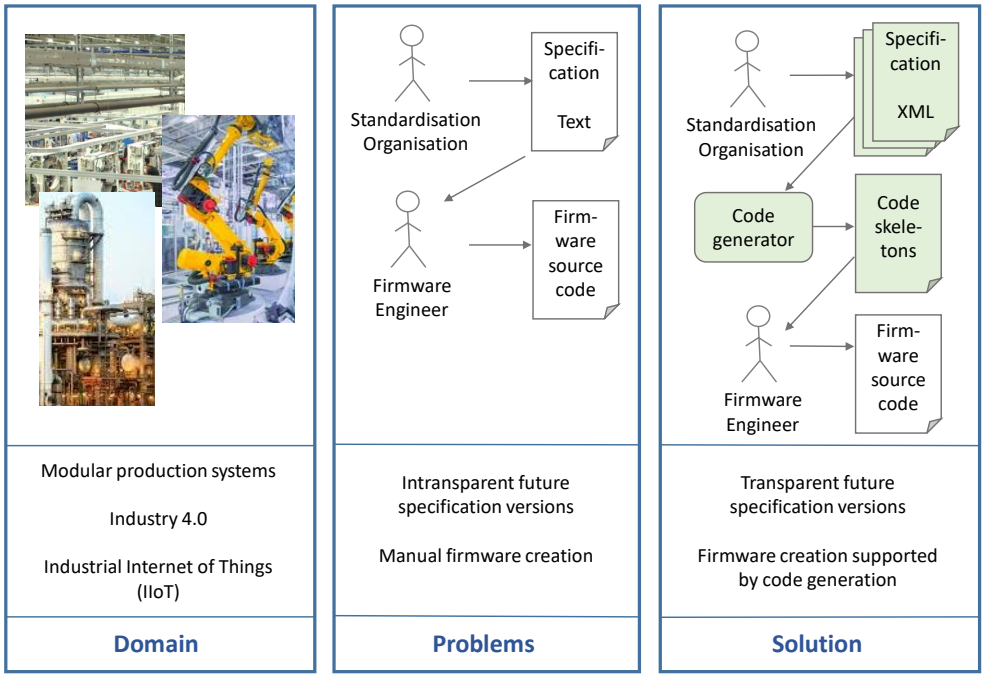


Fig. 1. Problems and solutions

This article describes methods and technologies for efficient editing of formal interface descriptions of production components and versioning of the specifications (see Fig. 1). On-line editing of specifications is proposed instead of using word processing software and examples of code generators are introduced. The proposed methods and technologies provide support for standardization organization and equipment manufacturers compared to SotA solutions.

2. METHODS AND TECHNOLOGIES

2.1. Improvement of workflows

***Workflow 1**: A standardization organization creates a standard for component interfaces.*

The current workflow for the standardization of interface descriptions for automation components is as follows: an interested company submits an application to a standardization organization for the cross-vendor standardization of interfaces for one or more types of automation components. The standardization organization organizes a working group and appoints the editor. Thereupon a text document is designed, which is successively extended and improved. The exchange of different versions of the document often takes place via a document exchange platform of the standardization organization, in rare cases also via e-mail. The editor has the task of merging different versions of the document.

The whole procedure can be improved with regard to various criteria. For example, a shorter time to complete the document is aimed for, as well as a reduction in errors when merging different document versions and a reduction in the amount of work required for this.

These goals can be achieved with a modified workflow:

- The working group is formed as described above.
- However, the standardization organization has a tool for the formal description of interfaces which replaces the previously used word processing software. This should be a Web-application so that authors do not need to install any software on their personal PCs. The standardization organization administrates access for all authors.
- The authors create their proposals for extending and changing the standard document directly in this Web-application. The editor only plays a moderating role in the decision-making process for accepting the changes. The input of free text sections is at least partially restricted. For the selection of certain characteristics of data to be transferred, authors can only choose from a set of permissible values. Such value sets are usually defined on a long-term basis by the standardization organization
- The writing of the standard ends with the release of the document. The Web application should make the version statuses publicly available. This can be ensured, for example, by storing the versions in a publicly accessible version management repository.

***Workflow 2**: A design engineer of an equipment manufacturer creates an interface description for a component, which refers to standards and proprietary extensions.*

A document that defines a standard for interfaces of an abstract equipment component type contains at least the following important sections: a definition of complex data types and a description of equipment component variables as an assignment of variable names and a corresponding meaning to data types and other properties, such as the type of data transmission (cyclic/acyclic). Simple data types are often already defined in other guidelines of the standardization organization. Component variables describe, for example, parameters for changing the component behavior, internal states of the component or different sensor values available within the component.

When developing a concrete equipment component type, the design engineer of an equipment manufacturer determines which of the component variables declared in the standard should be available in a concrete component type. It must be taken into account which variables are described in the standard as mandatory and which as optional variables. In many cases, component manufacturers also define proprietary extensions to enable additional functions that distinguish these components from those of competitors. All these specifications by the design engineer are defined in a document, which is referred to as the component or equipment description in the following.

There are several ways to define component descriptions. The component description can be formulated as text, which is read and translated into source code by the developer of the equipment component firmware. However, it can also be defined as a formal description that can later be translated into source code using a compiler. The use of one or the other method depends on the respective equipment manufacturer. The descriptions from the standard are then often copied and pasted into the component description.

However, the aim is to include the components to be referenced from the standard as directly as possible in the component description. This goal can be achieved with a modified workflow:

- The design engineer of an equipment manufacturer uses a software application to download the standards. In the best case, this tool is also implemented as a Web application in order to reduce the overall effort for software installations.
- The design engineer of an equipment manufacturer uses the Web-application for creation of a new component description.

- The declarations of data structures and the definitions of variables from the standard are referenced directly (i.e. not by copy and paste). This enables a very fast adaptation of the component description to changes in the standard.
- With the Web application, the design engineer of an equipment manufacturer has similar possibilities for the definition of data structures and component variables as the authors of the standards. He or she uses these for necessary proprietary additions to the standards and references them as in the step of this workflow described above.

Workflow 3: *A software engineer of an equipment manufacturer creates the firmware of a component according to an equipment description.*

An intelligent, networkable equipment component has a firmware that controls the component itself and its data exchange with other components. The creation of this software is often complex and time consuming. Therefore, the automatic generation of at least some parts of the code out of the component description is aimed at. This goal can be achieved with the following work steps:

- The tool used by the design engineer in best case supports the creation of some source code. Thus the firmware engineer downloads the code directly from the Web application hosted in the company.
- The generated part of the firmware code can only contain so-called skeletons for the interface functions. The business logic code for the calculation of the internal states of the component or for the acquisition and processing of data from the component sensors is then added by the firmware engineer.

2.2. Formalized language for description of IIoT components

To improve the workflows mentioned above, Web applications are required that support authors of standards, component design engineers and firmware engineers. The structure of the required data models is defined by a formal language. In the project Productive 4.0 (https://productive40.eu), such a formal language was developed, which is presented in the following sections.

Basis for the formal language is XML (see W3C 2006). This format has also become accepted for other descriptions of technical systems, e.g. for the Generic Station Description Markup Language (GSDML, see PNO 2020) or the IO-Link Device Description Specification (IODD, see IO-Link Consortium 2009). The specification of XML elements and attributes has been done by using the specification language XML schema (see W3C 2012a and W3C 2012b). It is easily possible to check instance documents for completeness and correctness by using such a schema and a validating parser. There are validating parsers for almost all programming languages.

Two different types of specifications are mentioned in the workflows described above:

- Data structure library: These are files in which data structures and variable definitions are specified for an abstract class of equipment components. Such files are created and maintained by the working groups of the standardization organizations.
- Equipment interface description: Such files are created and maintained by the design engineer of an equipment manufacturer. They reference data structures and variable agreements from standard libraries, but also from proprietary data structure libraries. These files describe exactly one type of equipment produced by the component manufacturer.

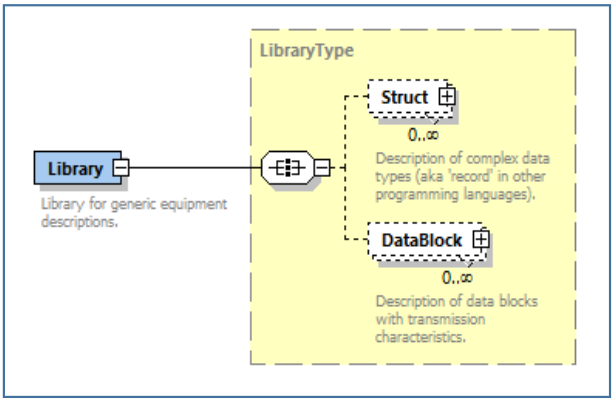


Fig. 2. Basic structure of a data structure library

Within a data structure library, it is possible to define data structures and data blocks (see Fig. 2). Structures form the basis for the specification of complex data types (see Fig. 3). The elements of data structures can be defined by simple scalar data types, such as numeric or character string types. But data structures can also be defined by references to other data structures or as lists of simple scalar data types, references to other data structures or further nested lists.

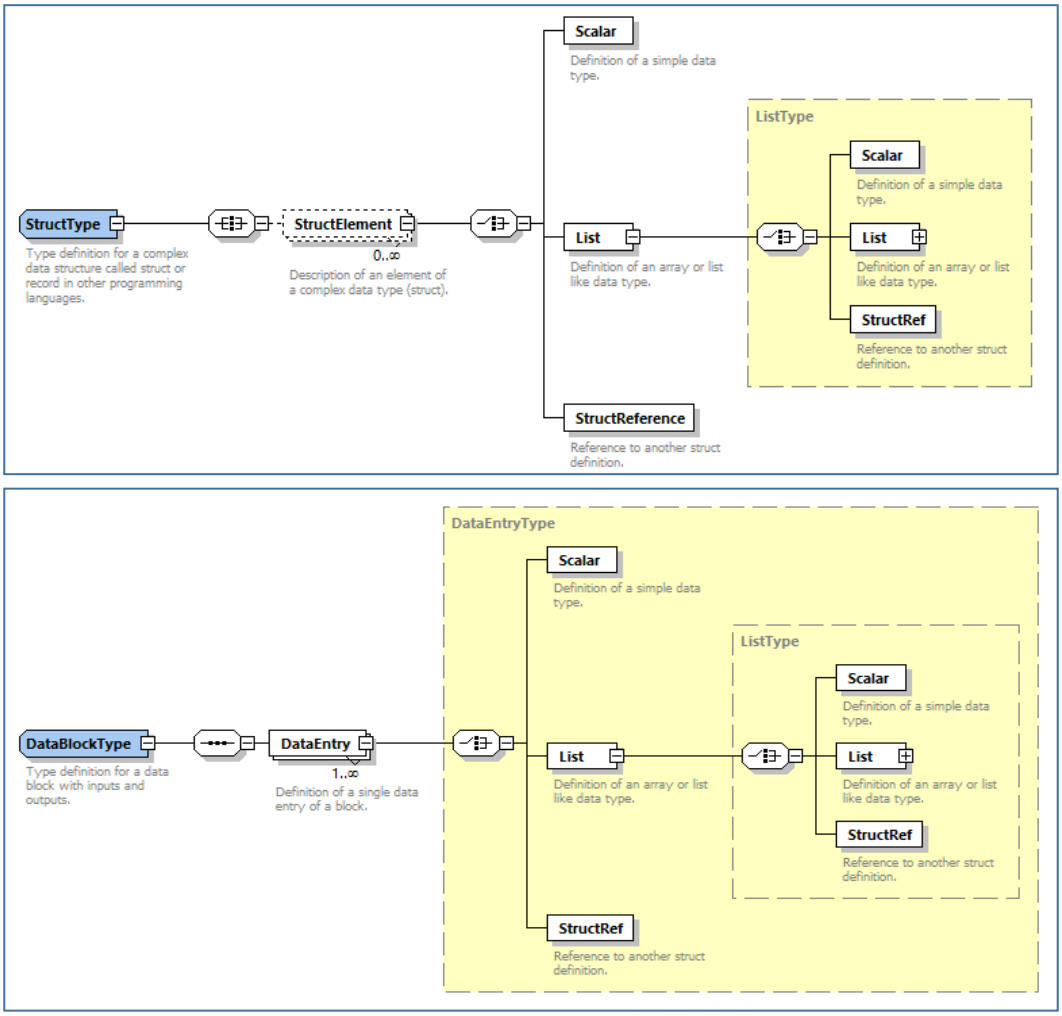


Fig. 3. Elements for describing data structures and data blocks

Data blocks contain entries that represent corresponding component variables (see Fig. 3). Such block entries are constructed similarly to structural elements. However, they have very different attributes.

Attributes that describe both struct elements and data block entries are as follows:

- ID: identifier of the struct element or data entry, usually a UUID (see also Leach 2005)
- Name: name of the struct element or data entry
- Description: description of the struct element or data entry
- Comment: comment of the struct element or data entry

Attributes for the exclusive definition of data entries:

- Direction: indicates, whether it is an input, output or in-/output of the equipment component
- TransmissionCharacteristic: describes, whether this data entry is part of the cyclic or acyclic data transfer
- Access: describes read/write characteristics
- Persistence: indicates, whether the value is safe even in case of power fail (retain) or not (non-retain)
- ValueRange: describes value range of permitted values
- DefaultValue: the preset value
- SubstituteValue: this value is transmitted in case of a substitution strategy is performed
- PhysicalUnit: an identifier for the physical unit of the value (makes sense for numerical values)
- Conditions: Description of conditions under which the value is valid, e.g. only in a certain operation mode of the component
- Support: statement, whether the data entry is mandatory, conditional or optional
- ConditionalSupport: if the Support is conditional, then this attribute contains a statement about the condition
- ValueSemantics: a reference to the definition of the parameter (e.g. a reference to another standard or a parameter dictionary like eCl@ss)

Those attributes have been derived from the Fieldbus Neutral Reference Architecture for Condition Monitoring in Factory Automation (VDMA 24582 2014), which provided a very powerful set of attributes for description of component variables.

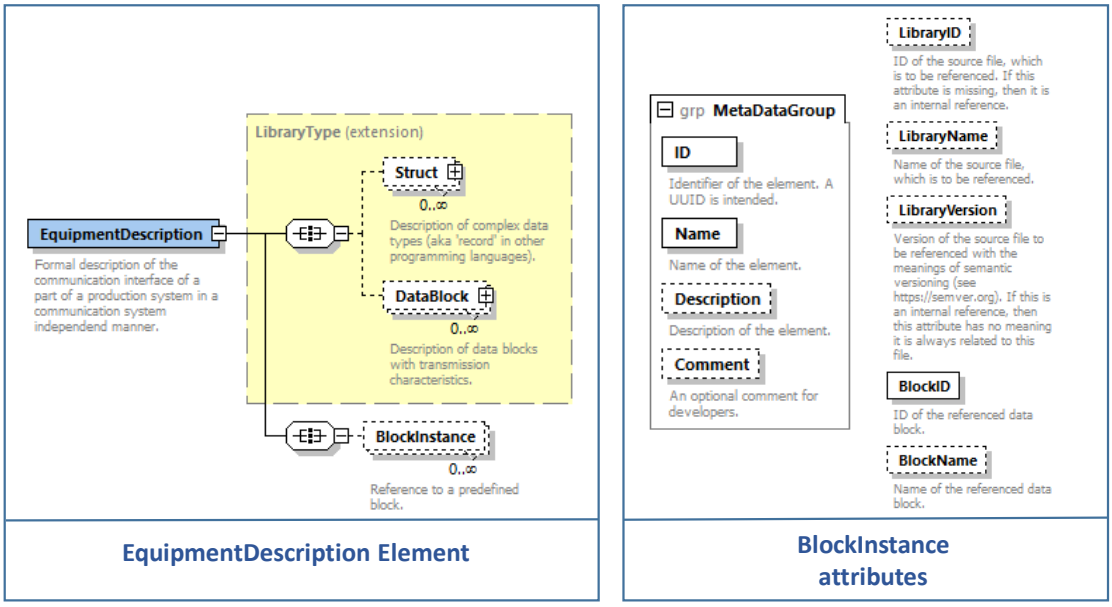


Fig. 4. Equipment descriptions and data block instances (component variables)

The structure of an equipment component description is partially the same as that of a library. It is possible to define Structs and DataBlocks as well. But the essence of the equipment component description is the definition of BlockInstances. There are two groups of BlockInstance attributes:

- Attributes, which are related to the instance itself (ID, Name, Description and Comment)
- Attributes referring block types defined within the EquipmentDescription or externally in a library file (LibraryID, LibraryName, LibraryVersion, BlockID, BlockName).

3. RESULTS

3.1. Architecture of the description and code generation framework

As part of the Productive 4.0 project, it was examined whether the implementation of the workflows described above is possible. For this purpose, it was necessary to implement three sub-applications:

- Data Structure Editor (DSE): This tool serves the members of the working groups for the standardization of component interfaces to edit the corresponding formal specifications.
- Equipment Type Interface Editor (ETIE): This tool is used by component designers of equipment manufacturers to create formal component descriptions.
- Skeleton Code Generator: This tool can be used by the firmware engineers to create skeletons for functions of the component interfaces based on the component descriptions.

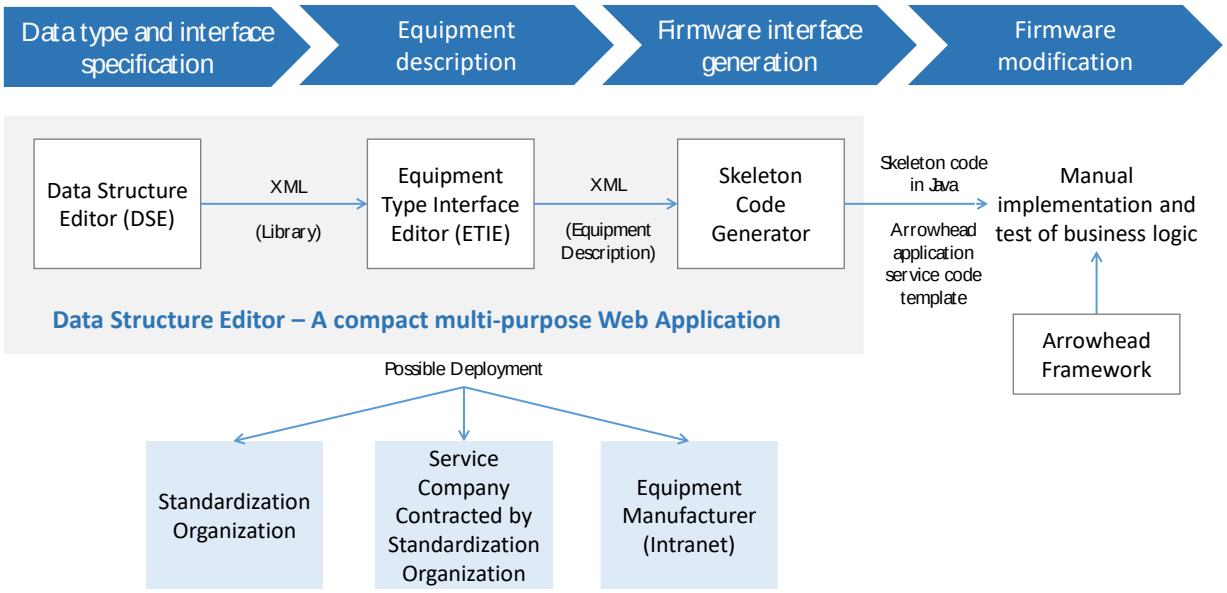


Fig. 5. Architecture of the code generation framework

Within the project, a compact multi-purpose Web application (Data Structure Editor) was developed. This application is deployable on different sites. This can be the standardization organization itself or a company commissioned by it. It offers the possibility to manage libraries with data structures and data blocks.

Furthermore, the Data Structure Editor can be installed in the intranet of an equipment manufacturer. There it is used to create and maintain proprietary libraries and equipment descriptions. The firmware developers can also download source code for the skeletons of the interface functions from the Web application installed in the equipment manufacturer's intranet. This code can then be used as a basis for implementing the necessary business logic of the component.

Within the Productive 4.0 project, the Arrowhead Framework was chosen as the central communication platform. This must be installed in order to test the firmware, which is implemented as an Arrowhead application service.

3.2. Tools for editing component descriptions

The Data Structure Editor is a Web application. This means, it consists of a client and a server application part. The server was implemented on the basis of the programming language Python (https://www.python.org) and the micro web framework bottle (https://bottlepy.org). The server provides a single HTML page with embedded logic in the programming language JavaScript (single page application). The framework Vue.js (https://vuejs.org) in combination with the extension Vuetify.js (https://vuetifyjs.com) was used as the basis for the graphical elements of the application interface. The communication between client and server was realized via an HTTP-based REST interface. Since the server-side data storage was done in XML, but on the other hand the processing of JSON in the JavaScript-based client application is much easier, on the server side a transformation between XML- and JSON-based representation forms of the data structure libraries and the equipment descriptions was done.

The versioning of the XML files on the server side was done with the help of the version management program Git (https://git-scm.com). This makes it possible to use local repositories as well as private and public repositories hosted by any service provider in the Internet.

Fig. 6 shows screenshots of the Data Structure Editor. It is illustrated that editing formal descriptions is optimized for editing on both desktop-oriented PCs and mobile devices.

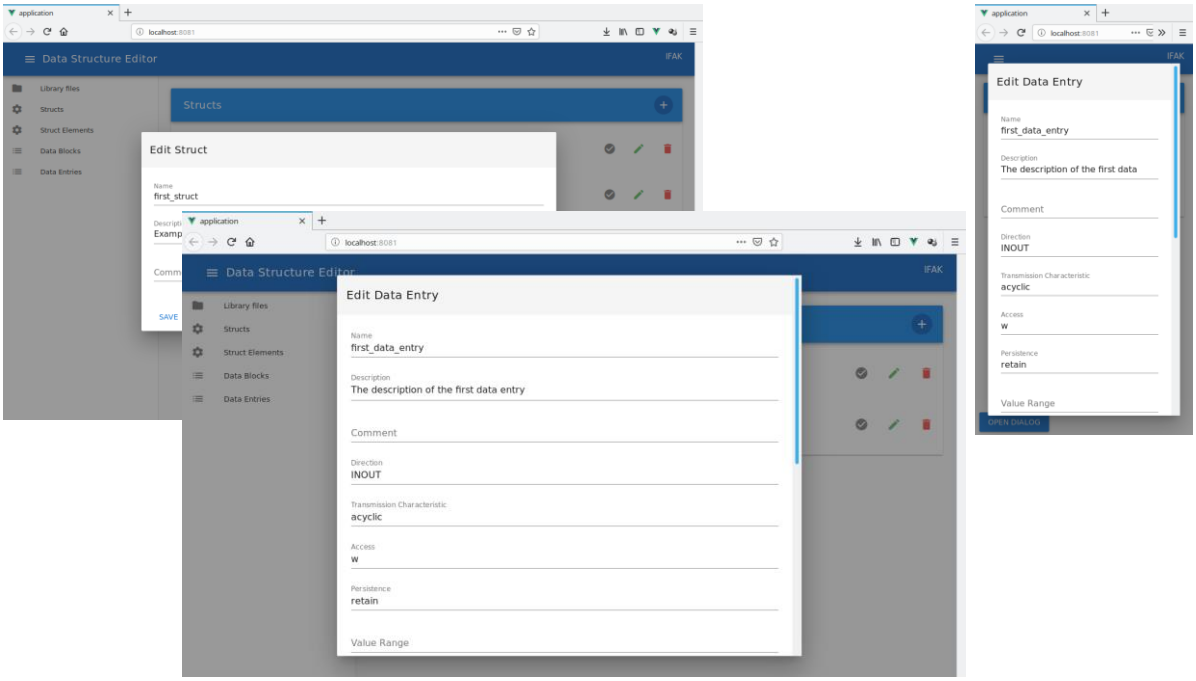


Fig. 6. Screenshots of ifak Data Structure Editor

3.3. Code generator

The code generator implemented on the server generates a ZIP file with several components from the formal component description:

- A plain old Java class (POJO), which creates a JSON object that conforms to the formal specification based on the addressing via the names of the block instance and the block entry. However, the values in this JSON object must be supplemented with code that must be developed manually. The developer has no effort to learn the special properties of the Arrowhead Framework.
- A template for an Arrowhead Application Service that was developed based on the example code for the Arrowhead Framework (https://github.com/arrowhead-f/sos-examples-spring).
- A Python based script with following features:
    - Automated download of the Arrowhead Framework
    - Build of the Arrowhead Framework
    - Start of all necessary Arrowhead Framework core services
    - Build and run of the generated Arrowhead Application service.

The development of the firmware, which is based on the Arrowhead Framework in the selected context, is therefore very easy.

4. CONCLUSION

The article showed an efficient approach for a versioned and formalized specification of data structs and blocks by standardization organizations. Further on, this can be used for versioned formalized specifications of equipment interfaces in scope of industrial equipment manufacturers. Finally, some code for the interfaces of IIoT components can be generated and thus the overall approach saves time and efforts in establishing IIoT and Industry 4.0 communication networks.

The developed formal languages and tools are prototypes, which served the purpose to evaluate the approach. After the end of the Productive 4.0 project (October 2020), they will be released under an open source license in a public repository on GitHub (https://github.com). Language and tools can then be continuously improved.

ACKNOWLEDGEMENTS

The research work leading to these results has received grants as project Productive 4.0 from the European H2020 research and innovation programme and the ECSEL Joint Undertaking under grant agreement no. GAP-737459 – 999978918 as well as from the German Federal Ministry of Education and Research (FKZ 16ESE0185). We are grateful to have been given that opportunity for our research activities.

REFERENCES

1. CiA 408 2017, “CANopen device profile for fluid power technology proportional valves and hydrostatic transmissions”, version 2.1.0, CAN in Automation.
2. Fielding, R (ed.) 2014, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”. Internet Engineering Task Force (IETF). RFC 7231. June 2014.
3. IEC 61158 2019, “Industrial communication networks – Fieldbus specifications”, part 1-7, International Electrotechnical Commission.
4. IEC 61800-7-201 2015, “Adjustable speed electrical power drive systems – Part 7-201: Generic interface and use of profiles for power drive systems – Profile type 1 specification”, International Electrotechnical Commission.
5. IO-Link Consortium 2009, “IODD - IO-Link Device Description”, version 1.0, order no. 10.012.
6. PNO 2020, “GSDML Specification for PROFINET”, Version 2.41, PROFIBUS Nutzerorganisation e.V.
7. Leach, P (ed.) 2005, “A Universally Unique IDentifier (UUID) URN Namespace”. Internet Engineering Task Force (IETF). RFC 4122. July 2005.
8. OASIS 2019, “MQTT Version 5.0”, 07 March 2019. OASIS Standard.
9. VDMA 24582 2014: “Fieldbus Neutral Reference Architecture for Condition Monitoring in Factory Automation”, VDMA.
10. W3C 2006): Extensible Markup Language (XML) 1.1 (Second Edition), World Wide Web Consortium, 2006.
11. W3C 2012a: XML Schema Definition Language (XSD) 1.1 Part 1: Structures, World Wide Web Consortium, 2012.
12. W3C 2012b: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, World Wide Web Consortium, 2006.