Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 104 (2021) 1771–1776

www.elsevier.com/locate/procedia

54th CIRP Conference on Manufacturing Systems

Design and implementation of a holistic framework for data integration in industrial machine and sensor networks

Jonas Hillenbrand[a,*], Philipp Gönnheimer[a], Eduard Gerlitz[a], Jürgen Fleischer[a]

[a]*wbk Institute of Production Science, Gotthard-Franz-Strasse 5, 76131 Karlsruhe, Germany*

* Corresponding author. Tel.: +49 1523 950 2582; fax: +49 721 608-45005. *E-mail address:* jonas.hillenbrand@kit.edu

Abstract

Digitalization and connectivity trends in industrial plants and production equipment create vast and heterogeneous networks of data sources, data sinks and various communication protocols. Data fusion and evaluation of these resources result in high costs for data integration and maintenance. Therefore, we propose a new framework, called *MyGateway*, enabling effortless integration of heterogeneous data sources, their fusion within the framework and publication to data sinks as needed. For easy integration, deployment, and expansion of the framework we provide an implementation in JAVA using open-source adapters for common industrial protocols and a simple API for usage in user specified setups.

Keywords: system architecture; monitoring; network

1. Introduction

Industry 4.0 and network enabled sensors gave rise to highly digitized production environments with an abundance of data sources, communication technologies with need for storage and processing possibilities. Growing interest also exists in making production internet ready with the Industrial Internet of Things (IIoT) or a postulated *Internet of Production* [1].

Production relevant data sources and data sinks are heterogeneous and manifold. An excerpt of sources and sinks is shown in Table 1.

For companies whose core competency does not reside within communication technologies, it can be tedious to manage the growing number of IT and OT infrastructure required for the application of industry 4.0 solutions.

Smart devices and sensors, digitalization solution, platforms and protocols grow faster than standardized industrial communication is created. In order to cope with the abundance of different systems application-specific middleware is needed for protocol translation, consolidation of data sources and their administration in data sinks.

Beyond that, the integration of legacy devices or systems is just as important. A key enabler for IIoT or cloud manufacturing is interoperability. This issue has been addressed in [2] with a proposal for a cloud-based CAD-CAM framework for manufacturing with CNC machines. Other drivers are usage of communication stacks, that are interoperable by design, such as Open Platform Communications - Unified Architecture (OPC-UA). [3] propose a multiscale digital twin for modeling machine shops, including components, processes, and entire machines. A hierarchical semantic of the system is represented within the OPC-UA node tree. This enables a central allocation of data associated with a machine and its corresponding machine shop, that can be accessed independent of vendors.

Still, recent automation protocols and interfaces can only be considered, while developing new devices and machines, but usually brownfield environments are found,

Table 1. List of protocols for data sources and sinks

Category	Protocols
ethernet based protocols	TCP/IP, UDP/IP, HTTP(S), MQTT
object oriented automation standards / stacks	OPC DA, OPC-UA, Beckhoff ADS, MTConnect
bus systems	PROFI-BUS, PROFI-NET, EtherCat Can-Bus, ModBus/TCP Serial, IO-Link, I2C, SPI, AS-Interface, EIB, KNX
wireless networks / protocols	Bluetooth, Bluetooth Low Energy, ZigBee
flat files (MIME types)	video files (WMA, WMV, MPEG, AVI…), image files (JPG, PNG, GIF, TIF…), audio files (MP3, WAV, OGG ...), text based files (CSV, XML, JSON, …), binary files (HDF5, MAT, *.bin, …)
audio and video stream protocols	Real-Time Messaging Protocol (RTMP), Real-Time Streaming Protocol (RTSP), Dynamic Adaptive Streaming over HTTP, Microsoft Smooth Streaming (MSS), HTTP Dynamic Streaming (HDS), HTTP Live Streaming (HLS)
database implementations	SQL, MySQL, SQLLite, PostgreSQL, InfluxDB, MongoDB, NoSQL,

where existing machinery must be integrated into new reconfigurable production systems. To target brownfield applications, usually additional hardware is required to access the legacy communication protocols at the device level. Here, [4] introduced a modular smart controller for industry 4.0 functionalities in machine tools. It is well-suited for applications, where real-time and best-effort applications meet. So far, only Controller Area Network (CAN) was implemented as communication protocol, but the modular design allows for integration of other protocols and periphery.

With *MyGateway* we address the problem of interoperability by introducing a universal, protocol and platform independent framework to integrate any digital data sources in industrial plants. In the following, the structure of the paper is introduced.

In *Related Works* state of the art and research is described and used to define the novelty of features in *MyGateway*.

In chapter 3 the framework is introduced, and its elements and functionalities are described. It is followed by chapter *Experimental Results,* with a demonstration of the application's accuracy and an example use case. Finally, the results are discussed, and we close with a *Conclusion* and further future work.

2. Related Work

There is a huge market for industry 4.0 and digitalization solutions. A lot of vendors and data pipeline providers focus on bringing their customers' data to the cloud or centralized storage facilities, while providing drivers and bridges for sensors and devices. Usually, they promote these functionalities within the setup of their own system, requiring customers to change or install new architectures. An example is the centralized measurement management center from Delphin Technology [5].

Less intrusive are vendors that provide their own gateways as middleware as mobile or stationary application. The Sick AG offers an open end-to-end IIoT architecture for integration of bus and network protocols, but also offers wireless interfaces for communication [6]. Another solution in the IIoT sector is *ThingOs* representing an edge-technology platform for smart devices, sensors, and actuators [7]. Core concept is a technology-neutral abstraction layer (TNL) enabling their users to focus on the business logic instead of protocol handling. Their target market are smart home and smart factory appliances.

The mentioned solutions stand exemplary for a collection of highly integrated and customizable frameworks. However, as those applications are not open source, users are restricted to their application programming interface (API) and usually have to use the companies' products within their installations.

Here, open-source software or communication frameworks provide tools to cope with the various industrial protocols and an open system architecture to develop own applications or integrate existing infrastructure.

In form of *Apache Kafka* an event streaming platform was introduced for a variety of use cases, including capturing, and analyzing sensor data from IIoT device or other equipment [8]. It is a distributed network of servers and clients, scaling from on premise to cloud environments. *Apache Kafka* targets already smart or network connected devices. Another project from The Apache Software Foundation is *PLC4X*, it consists of a set of libraries for communication with programmable logic controllers (PLC) using different protocols but a shared API [9].

Kuhn et al. introduce the concept of a *virtual automation bus* (*VAB)*, a peer-to-peer communication solution for industrial assets managed within asset administration shells according to VDI/VDE [10,11]. They enable inter-network communication with existing infrastructures, using a common language and HTTP/REST for communication within the bus. Protocol specific gateways are used to map heterogeneous protocols into the *VAB*.

Other works address the problem by defining domain-specific languages for sensor integration. Bodenbenner et. al proposes the *SensOr Interfacing Language (SOIL)* to decouple sensor data and properties from the underlying protocol [12]. It targets sensor equipment with network access and an open API. Applying *SOIL* users can create a code-free meta model of their sensor, which is then translated into executable python code. In order to be used with *SOIL*, the equipment must implement an HTTP interface.

Common ground of above frameworks and works are technology neutral layers, that allow reducing the programming effort and help reuse code. Another core functionality is the use of bridges and adapters to solve the issue of incompatible protocols or communication schemes.

Our proposal of *MyGateway* also incorporates the idea of an abstract layer for translation between different protocols. Where applicable we make use of existing solutions, such as

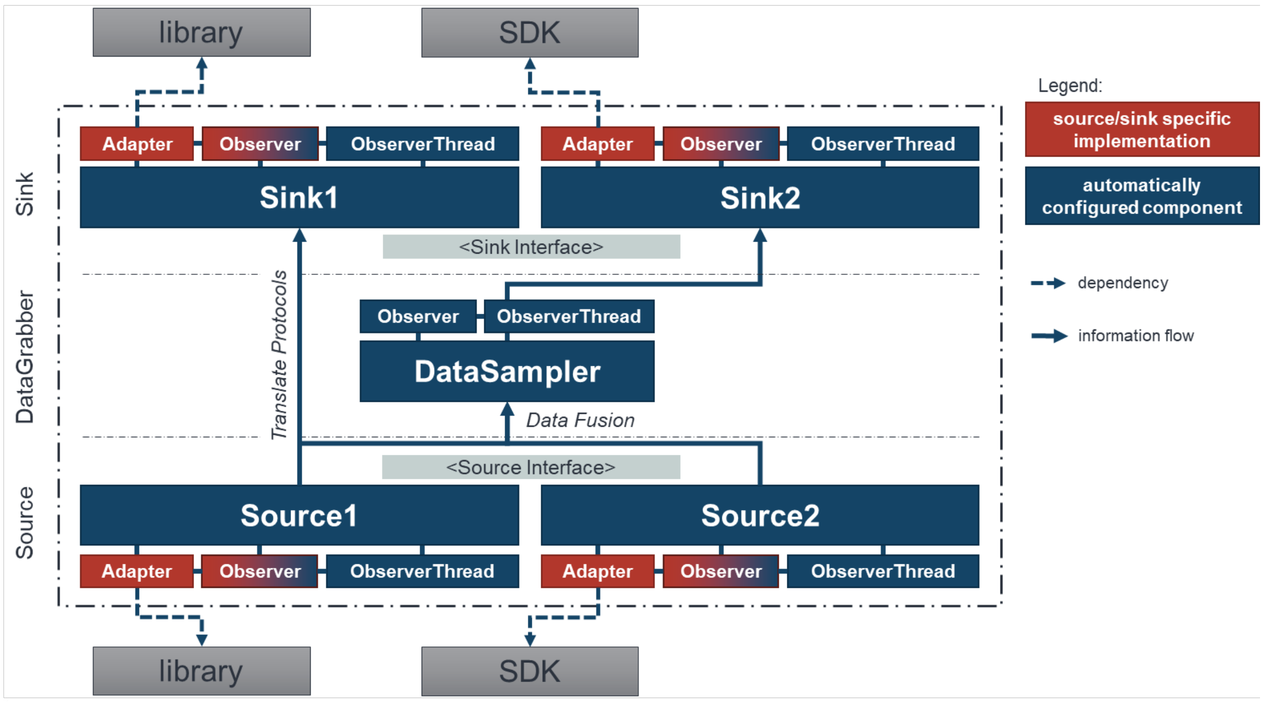


Fig. 1. MyGateway architecture

PLC4X, if PLC communication is required. Instead of defining a new language or network architecture for industrial communication, *MyGateway* must be understood as framework of wrappers for protocols and a technology neutral layer within which sensor data streams can be processed.

MyGateway specifically addresses data logging applications and condition monitoring scenarios, where legacy equipment must be incorporated, and high-speed data streams must be processed.

It operates as gateway, similar to installations like the *VAB* or an *Apache Kafka Connect* instance [8,11]. It therefore helps in retrofitting projects where usually a lot of programming expenses go into programming of adapters and protocol translation.

We realize aggregation of data sources as close to the hardware as possible, so that data fusion and later on context-based feature extraction can be done as close to the source as possible. Also, high configurability and effortless expansion are ensured using code templates and encapsulation wherever it is possible. Once a protocol or device was implemented, it can be automatically configured, and executable code can be exported to run the application on any target machine with a Java Runtime Environment (JRE).

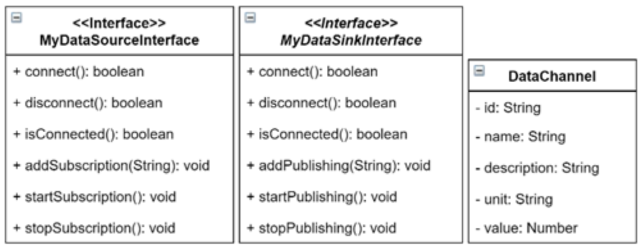


Fig. 2. MyGateway interfaces and the DataChannel definition

3. MyGateway Framework

The proposed framework *MyGateway* targets brownfield shopfloors and machinery, which are not up to date in terms of modern communication architectures, or due to historic growth an abundance of different systems can be found, with varying communication standards and no common ground for interconnection. For those environments *MyGateway* poses an opportunity to translate and consolidate, respectively synchronize, data sources of any kind, while staying highly configurable and expandable for new or changing sources. In terms of the automation pyramid *MyGateway* is located between field and control level and the planning or management level. Besides vertical integration of data sources, it also enables horizontal integration and communication between devices and machines with different communication standards through mapping. In the current version *MyGateway* is conceived for protocol translation, data fusion and data logging of digital sources and streaming or storing them in data sinks, e.g. databases or webservers for visualization.

The architecture of *MyGateway* consists of three main layers, namely *MyDataSource*, *DataSampler* and *MyDataSink*. The data flow direction within this framework goes from *MyDataSource* into *DataSampler* and leaves the framework through *MyDataSink*. Fig. 1 displays these layers and also shows the dependency of the framework on other libraries or software development kits (SDK). The goal in this setup is to reduce the protocol and vendor specific code for processing industrial data streams. The blue elements in Fig. 1 represent core *MyGateway* components, that are automatically created based on required connections and use case specific architecture. Red highlighted elements require a one-time effort of programming to cope with vendor specific or protocol specific implementations. Once they are created for a project, they can be reused in new applications. In the following subsections the core components are further explained in detail.

3.1. Data sources

The component *MyDataSource* is an abstract class describing any data source that can be connected to the framework. The term data source in the context of this work refers to any kind of network participant, that produces a data stream other entities can connect to. MyGateway is hosted in such a network. Examples are OPC DA and UA Servers, TCP and UDP sockets or flat files with comma-separated values (CSV). In *MyDataSourceInterface* we define basic workflow and minimum methods each class of *MyDataSource* should adhere to. A simple workflow consists of connecting to the source, checking if it is connected, then subscriptions are added in form of *DataChannels* and then those subscriptions can be started and stopped and if no longer required, the connection can be disconnected (see Fig. 2).

Data within *MyGateway* is stored in *DataChannel* objects, containing an *id* and *name* property, fields for describing the data and a numeric property (*value*) that is updated via an observer pattern (also see Fig. 2). These values are received using the underlying *adapters* of the corresponding *MyDataSource* object. An exemplary implementation of this relation for OPC UA is the setup in schematic in Fig. 3. As adapter for the communication with an OPC UA server we use the *Eclipse Milo*, an open source implementation of the OPC UA stack [13]. We use the SDK to build our own *MyOpcUaClient* that is able to connect, read and write server nodes. If one of those nodes shall be subscribed, the *MyOpcUaClientDataSource* maps those nodes into a *DataChannel* object, where they can be accessed from the rest of the *MyGateway* elements.

3.2. Data Sampling

The next layer in *MyGateway* is represented by *DataSamplers*. The *DataSampler* inherits its functionality from *MyDataSource* and implements the same interface. As mentioned before, one task of our system is consolidation of different data sources, this is solved by additional methods inside *DataSampler*, who subscribes to all *MyDataSources*

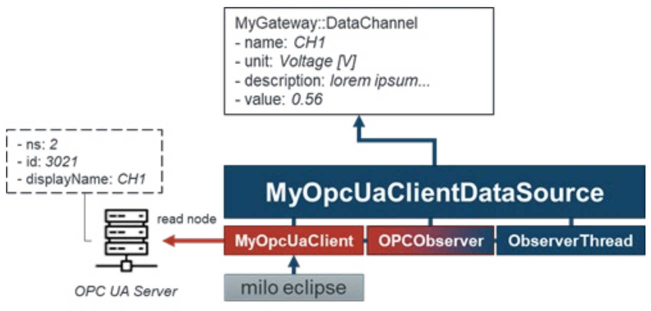
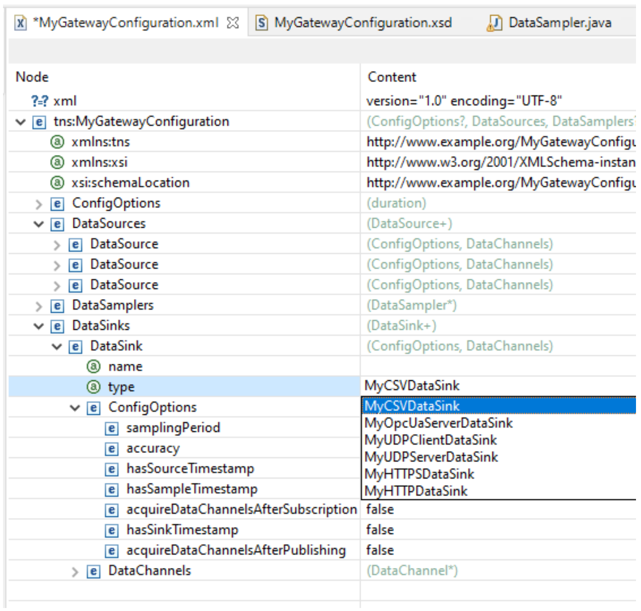


Fig. 3. Setup of OPC UA data source



*MyGatewayConfiguration.xml   MyGatewayConfiguration.xsd   DataSampler.java

Node	Content
?-? xml	version="1.0" encoding="UTF-8"
tns:MyGatewayConfiguration	(ConfigOptions?, DataSources, DataSamplers
xmlns:tns	http://www.example.org/MyGatewayConfigu
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instan
xsi:schemaLocation	http://www.example.org/MyGatewayConfigu
ConfigOptions	(duration)
DataSources	(DataSource+)
DataSource	(ConfigOptions, DataChannels)
DataSource	(ConfigOptions, DataChannels)
DataSource	(ConfigOptions, DataChannels)
DataSamplers	(DataSampler*)
DataSinks	(DataSink+)
DataSink	(ConfigOptions, DataChannels)
name	
type	MyCSVDataSink
ConfigOptions	MyCSVDataSink
samplingPeriod	MyOpcUaServerDataSink
accuracy	MyUDPClientDataSink
hasSourceTimestamp	MyUDPServerDataSink MyHTTPSDataSink
hasSampleTimestamp	MyHTTPDataSink
acquireDataChannelsAfterSubscription	false
hasSinkTimestamp	false
acquireDataChannelsAfterPublishing	false
DataChannels	(DataChannel*)

Fig. 4. Graphical interface for XML configuration in eclipse

that need to be fusioned. A second task of this element is the provision of data buffers for high-speed data streams. As *MyGateway* runs on non-real-time operating systems (Non-RTOS), continuous sampling of data sources cannot be guaranteed without a buffer. But, if the underlying data source (device or sensor system) already provides an interface for buffering data, those buffers can be read and transferred to *MyGateway* while preserving the continuously sampled data.

3.3. Data Sinks

When the data sources are consolidated within *MyGateway*, we use data sinks to transfer the data acquired to its target destination. Example data sinks are OPC UA servers, MQTT publishers, database servers like InfluxDB, web servers, plain text or binary files for offline storage purposes. Adapters for those targets are used for the implementation of *MyDataSinks*. The interface and definition of *MyDataSink* resembles the one from *MyDataSource*, as shown in Fig. 2, but instead of incoming data, it describes the handling of outgoing data. By transitioning all three layers of *MyGateway*, we reach protocol translation, synchronize data sources under a common timestamp and establish a highly flexible setup for the aggregation and distribution of data streams.

3.4. Graphical user interface for MyGateway Configuration

Our current version of *MyGateway* supports building applications within any integrated JAVA development environment (IDE) or being configured by specifying an extensible markup language (XML) configuration file. We provide XML templates for all *MyDataSources*, *DataSamplers* and *MyDataSinks* supported so far. Possible configuration options are based on an XML scheme.

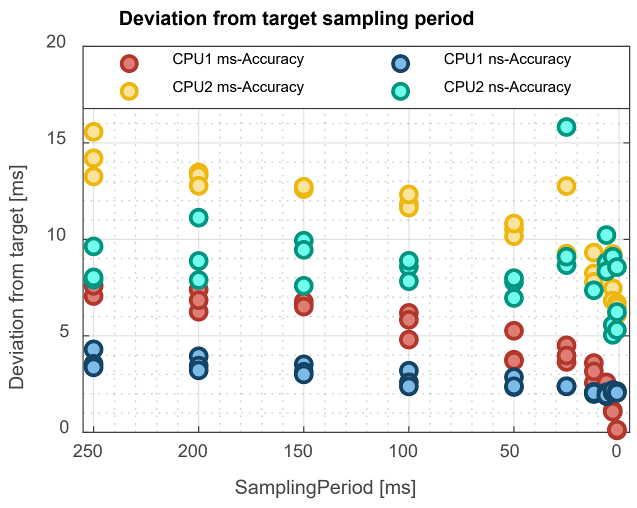


Fig. 5. Sampling accuracy tests with different CPU's

This scheme ensures that the user is guided through the configuration and no invalid values are submitted. Any XML editor with validation routines can be used to configure *MyGateway*. We use the XML editor provided in eclipse, as it is also the IDE used for development (see Fig. 4).

4. Deployment Studies

In the following chapter, we present results from sampling accuracy tests with different CPUs and an implementation use case on testbenches at our institute.

4.1. Sampling accuracy tests on non-RTOS systems

As the framework operates on non-RTOS systems, the sampling is controlled from the CPU of its system, a fixed sampling rate can only be ensured for low sampling rates. In Fig. 5 we outline the usability of *MyGateway* for sampling tasks of a few milliseconds.

The plot shows the results for two different CPU and the same *MyGateway* configuration, consisting of an OPC UA client that publishes its subscribed server nodes into CSV files. CPU1 is an Intel Core i7-8565U CPU at 1.80GHz and 4 cores, representing an office grade laptop, CPU2 is a quad core Cortex-A72 at 1.5GHz on a Raspberry Pi 4B+. On both machines *MyGateway* configurations with different sampling periods (between 250 and 1ms) for the subscribed data were run. The resulting time differences in milliseconds are gained by computing the difference in CPU system time from one sample to the next. The system time is stored in an additional *DataChannel*, that is also published into the CSV file.

The sampling period is controlled within the *ObserverThread* and its accuracy can be set to either milliseconds or nanoseconds. The tests show the dependency of the accuracy on the available CPU, as results gained from CPU1 (higher clock rate) deviate less from the target sampling period. Also, choosing nanoseconds as resolution for the sampling helps reducing the deviation further. However, deviations fluctuate and can be off the target sampling period by multiple milliseconds. Usage scenarios where continuous sampling rates are required cannot be addressed yet, with the unbuffered data processing.

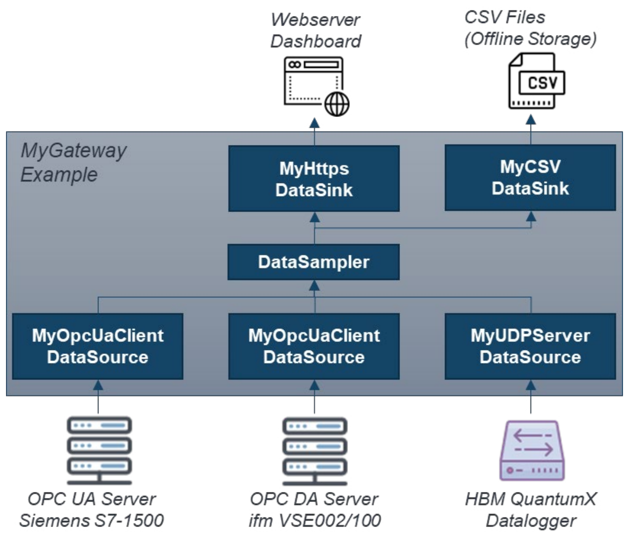


Fig. 6. Schematic of implementation example (icons taken from [14])

4.2. Implementation example

To test the configuration and operation capabilities in a real environment, we deployed a *MyGateway* application on a PC that is used to control a testbench in our institute's component laboratory for machine tools. The purpose of this testbench is conduction of lifetime experiments on ball screws and hence requires long-term observation and robust logging of sensor data. In this particular case, the system of data sources and sinks is depicted in . It consists of three data sources in form of an OPC UA server running on a S7-1500 PLC CPU from Siemens, an OPC DA server running on the host PC capturing data from a measurement device of ifm electronics and a strain gauge data logger from HBM, called QuantumX, with a vendor API that allowed us to program a UDP client, that sends measurements as continuous data stream.

The data was transferred through a *DataSampler* and consolidated, in order to publish the data for a web server hosting a monitoring dashboard. This was done by using the framework's *MyHttpsDataSink,* which allows querying data from a *MyGateway* instance over HTTPS requests. Additionally, the testbench data was logged into CSV files using *MyCSVDataSink*. Customization of code was only required for the UDP client, as the vendor API did not natively talk in any already implemented *MyDataSource* protocol. Here, we programmed a C# UDP client, that uses the vendor API and talks with a corresponding UDP server on the framework side. Otherwise, the configuration can be completely setup with graphical XML modeling.

4.3. Other MyGateway features

Besides the aforementioned features of *MyGateway* we want to mention several other functionalities available in this framework.

Instances of *MyGateway* can be run on 32- and 64-bit platforms using different JAVA runtimes. That enables bridging between legacy systems newer architectures by coupling more than one *MyGateway* instance.

By compiling *MyGateway* as a JAR file (an executable Java program), it can also be executed within MATLAB. This way we facilitate multi-threaded data acquisition in the background, while MATLAB's graphical frontend can be used to do signal processing. So far, we implemented buffered data sources and data sinks for the network protocols UDP and TCP, allowing for the processing of high-speed data streams. General application of buffered sources and sinks are still under development.

5. Discussion

In chapter three we show the *MyGateway* architecture and its capabilities to solve data logging problems for heterogeneous data sources and data sinks. The framework is able to bridge between different commonly used industrial protocols, such as OPC UA and OPC DA. Additionally, in different use cases we used the *PLC4X* libraries to communicate with PLC controllers without OPC interfaces. Vendor specific API can be used to integrate devices into the network using TCP / UDP clients or server on the *MyGateway* side. We showed test results for data sampling task in the framework for different CPU's. Calculated accuracies show differences between target sampling period and actually received timestamps. As MyGateway is run on non-RTOS systems this must be taken into account, considering possible transformations after data acquisition, e.g. fast Fourier transformation, that requires data with constant sample rate. This issue is already being addressed with the extension of *MyGateway* by buffered data sources and sinks. Currently, the largest effort is building adapters and wrappers for state-of-the-art protocols. We want to further automate the process by defining a wrapper or adapter template, so that code for *MyDataSource* and *MyDataSink* elements can be generated automatically. To ensure easy configuration and interaction while creating *MyGateway* applications, we make use of graphical XML modeling. To increase the usability, the modeling of MyGateway can be ported to a web-based configuration tool.

6. Conclusion and Outlook

We have presented our framework *MyGateway* for the purpose of integrating heterogeneous data sources and their consolidation into data sinks for industrial applications. *MyGateway* specifically targets legacy devices or retrofit applications, where currently no gateway solution or integration into higher level data pipelines (manufacturing execution (MES), enterprise resource planning systems (ERP) or cloud environments) exists. By using open-source libraries and frameworks, we created a technology neutral layer, that translates protocols and consolidates data streams for data logging purposes.

In future works, we concentrate on three new functionalities: Automatic identification of sources within the *MyGateway* environment (network) related to the works in [15]. We analyze the network the framework is connected to for participants that publish data or listen to ports. This enables less user defined configuration and automates the data integration process.

Running on architectures like the Raspberry Pi allows interfacing with devices that do not operate on Ethernet based protocols, like Bluetooth or WiFi, but also enables communication on embedded level using I2C or SPI communication.

Finally, we will extend *MyGateway* by data processing capabilities for condition monitoring, such as the inclusion of feature extraction functions and use of machine learning models within the framework.

References

[1] J. Pennekamp, et al., Towards an Infrastructure Enabling the Internet of Production, in: 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS 2019): Howards Plaza Hotel Taipei, Taiwan, 06-09 May, 2019, Taipei, Taiwan, IEEE, Piscataway, NJ, 2019, p. 31–37.

[2] M. Mourad, A. Nassehi, D. Schaefer, Interoperability as a Key Enabler for Manufacturing in the Cloud, Procedia CIRP 52. https://doi.org/10.1016/j.procir.2016.07.051, 2016, p. 30–34.

[3] D. Mourtzis, N. Milas, N. Athinaios, Towards Machine Shop 4.0: A General Machine Model for CNC machine-tools through OPC-UA, Procedia CIRP 78. https://doi.org/10.1016/j.procir.2018.09.045, 2018, p. 301–306.

[4] D. Barton, P. Gönnheimer, F. Schade, C. Ehrmann, J. Becker, J. Fleischer, Modular Smart Controller for Industry 4.0 Function in machine tools // Modular smart controller for Industry 4.0 functions in machine tools, 2019.

[5] Delphin Technology AG, Delphin Data Center: Zentrales Messdatenmanagement. https://www.delphin.de/produkte/software/delphin-data-center.html.

[6] Sick AG, Gateway-System TDC-E: Multi-Sensor-Vernetzung mit Mobilfunk-Kommunikation. https://www.sick.com/de/de/gateway-system-tdc-e-multi-sensor-vernetzung-mit-mobilfunk-kommunikation-daten-sammeln-auswerten-speichern-und-uebertragen-in-mobilen-und-stationaeren-anwendungen/w/press-2018-SMM-TDC-E/, 2018.

[7] ThingOS, ThingOS – The Smart Things Integrator. https://thingos.io/, 2018.

[8] The Apache Software Foundation, Apache Kafka: A Distributed Streaming Platform. https://kafka.apache.org/intro, 2017.

[9] The Apache Software Foundation, PLC4X: The universal protocol adapter for Industrial IoT. https://plc4x.apache.org/, 2017.

[10] T. Kuhn, P.O. Antonino, M. Damm, A. Morgenstern, D. Schulz, C. Ziesche, T. Müller, Industrie 4.0 virtual automation bus, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, Gothenburg Sweden, ACM, New York, NY, USA, 05272018, p. 121–122.

[11] T. Kuhn, P.O. Antonino, F. Schnicke, Industrie 4.0 Virtual Automation Bus Architecture, Software Architecture, Springer International Publishing, [S.l.], 2020, p. 477–489.

[12] M. Bodenbenner, M.P. Sanders, B. Montavon, R.H. Schmitt, Domain-Specific Language for Sensors in the Internet of Production, Proceedings of the 10th Congress of the German Academic Association for Production Technology (WGP), Springer Berlin, 2020, p. 448–456.

[13] Eclipse Foundation, https://github.com/eclipse/milo. https://github.com/eclipse/milo.

[14] Icons 8 LLC, icons8.de. https://icons8.de/icons, 2020.

[15] P. Gönnheimer, A. Puchta, J. Fleischer, Automated Identification of Parameters in Control Systems of Machine Tools, Proceedings of the 10th Congress of the German Academic Association for Production Technology (WGP), Springer Berlin, 2020, p. 568–577.