ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/325999583

Connecting PLCs with their Asset Administration Shell for Automatic Device Configuration

Conference Paper · July 2018

DOI: 10.1109/INDIN.2018.8472022

CITATION
1

READS
382

3 authors, including:

Monika Wenger
fortiss GmbH
36 PUBLICATIONS **201** CITATIONS
SEE PROFILE

Alois Zoitl
Johannes Kepler University Linz
204 PUBLICATIONS **2,041** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Model-Driven Embedded System Design Environment for the Industrial Automation Sector (MEDEIA) View project

IKT-Wandel - Digitale Transformation View project

All content following this page was uploaded by Monika Wenger on 26 October 2018.

The user has requested enhancement of the downloaded file.

Connecting PLCs with their Asset Administration Shell for Automatic Device Configuration

Monika Wenger, Alois Zoitl
fortiss GmbH
Guerickestr. 25, 80805 München
Email: {wenger, zoitl}@fortiss.org

Thorsten Müller
SMS group GmbH
Wiesenstrasse 31, 57271 Hilchenbach
Email: thorsten.mueller2@sms-group.com

***Abstract*—Current industrial automation systems integrate structural information and component configuration data within the code of their programmable logic controllers. This prevents the demanded flexibility and adaptability of Industrie 4.0 systems. The asset administration shell provides a defined mechanism for finding, accessing and interpreting standardized and vendor specific data of an asset. Within this work the asset administration shell has been analyzed according to its structure, its location, and its implementation forms. Based on that a concept is proposed to provide event driven run-time access for components to the asset administration shell, which enables an automatic self-configuration infrastructure for component instances. The proposed concept is applied on the logistic part of an aluminium cold rolling mill, provided by the SMS group. For demonstration the structural information of the logistic components has been described in terms of an AutomationML file, which has been loaded into a BaseX Database. The database content represents the asset administration shell, which is accessed during runtime by XQuery requests.**

I. Introduction

Current industrial automation systems are mostly designed in a rigid way. Structural information (e.g., arrangement of components) and component configuration data (e.g., speeds, acceleration) is embedded into the control applications hosted on the programmable logic controllers (PLCs) controlling them. Changing the production system or the product typically means that this configuration information has to be adjusted in a manual process. This kind of system architecture prevents the need for the more and more increasing flexibility, adjustability and need for faster commissioning of production systems.

A promising concept for reducing this effort is to assemble a production facility from a set of mechatronic components. These components can be arranged freely to the current needs. In order to remove the manual configuration effort these components need to retrieve information on their neighbor components or product relevant data (e.g., maximum speed for product). In the research project *Basissystem Industrie 4.0*[1] we are investigating software infrastructures which are supposed to support this. The key questions to be solved are, where this data origins, where and how it is stored as well as how it can be accessed.

[1]https://www.basys40.de/

10.1109/INDIN.2018.8472022$31.00 ©2018 IEEE

In the *Reference Architecture Model Industrie 4.0* [1] the concept of a so-called (asset) administration shell (AAS) acting as software entity representing an asset, has been introduced. An asset in this context can be a single component but also an assembly of components (e.g., module, cell, machine, plant). The AAS provides a defined mechanism for finding, accessing and interpreting standardized as well as vendor specific data of an asset (i.e., engineering as well as run-time data). This greatly reduces the number of interfaces and also relieves the user from the need to know where this data is stored. [6]

This opens completely new ways of how to engineer automation systems. With the AAS concept acting as central element different engineering tools can interact via a common AAS database allowing collaborative engineering across different engineering domains. Tools can automatically detect available plant configurations and data models for other tools. Through well defined access schemes even different companies can collaborate easily in engineering projects. [3]

But the AAS concept does not only change the engineering phase of a production facility. We think even more benefits can be achieved utilizing it during run-time. Assets can register themselves at its AAS and therefore the AAS can also be the access point for run-time data, either as proxy or as direct run-time data storage. Assets can also access the AAS for retrieving information about the plant structure, product information and also component instance specific configuration [2].

Assuming the availability of engineering data (e.g., [10]) this work investigates what information a component would need from an AAS implementation, how this information can be provided and how it can be accessed from the PLC code. Solving this questions provides an automatic self-configuration infrastructure for component instances.

The article is structured as follows: in the next section an overview on the AAS concept is given. Based on that information in Section III a concept for accessing data in the AAS shell from the PLC is presented. Section IV shows how this connection can be utilized in the logistic part of an aluminum rolling mill use case. A discussion on these first results and potential next steps conclude this work.

II. The Administration Shell

The AAS is supposed to be a virtual representation of a real component [2]. Therefore, it can be understood as a synonym of the term *digital twin* [3]. The AAS sufficiently describes all assets of an Industrie 4.0 Component related to an Industrie 4.0 use case [4], where an asset is an *"item which has a value for an organization"* [5]. An I4.0 component is a *"globally uniquely identifiable participant with communication capability consisting of administration shell and asset within an I4.0 system which there offers services with defined quality of service (QoS) characteristics"* [5]. QoS considers services as real-time capability, timing synchronization or reliability. *"An I4.0 component can represent a production system, a single machine or station, or even an assembly within a machine"* [5].

A. Structure of an Administration Shell

According to [2] the administration shell utilizes a type-instance model. The connection between a component's type and instance definition should be maintained through its whole life cycle. Also the definitions and data of one component has to be maintainable through the whole life-cycle, if required by the use case. The administration shell can be related to one or more objects. The security features of an object have to conform to the security features of an administration shell.

The administration shell, as illustrated in Figure 1 consists of the following main elements:

- **Administration Interface** is platform independent and provides services and properties, which are associated to one or more assets and contains references to relevant data.
- **Header** contains identifying details regarding the asset administration shell and the represented assets [4].

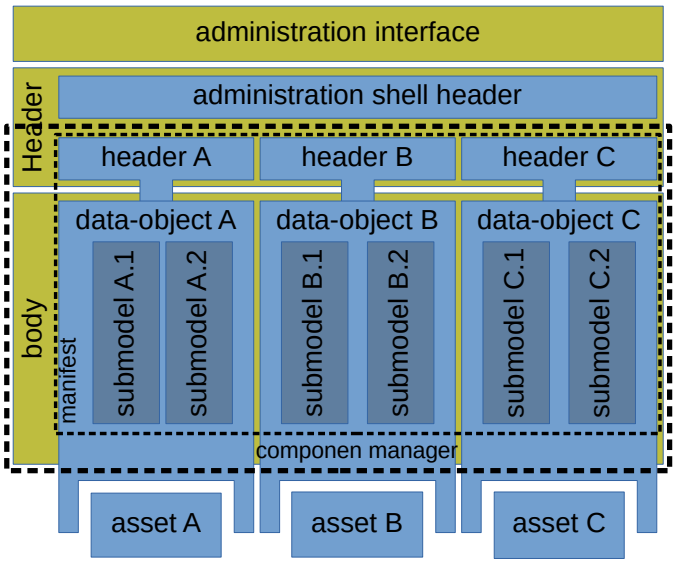

Fig. 1: General structure of an (asset) administration shell.

- **Body** contains a certain number of sub-models for an asset-specific characterization of the asset administration shell [4].
- **Submodels** represent the different aspects of an asset according to a corresponding standard. The submodels contain technical/specialized functionality of an Industrie 4.0 component, as computer-aided design (CAD) models, or structural information like neighbours.
- **Component-Manager** connects the administration shell to a repository, and provides interfaces that allow external access to the virtual representation and technical/specialized functionality. It therefore manages the sub-models dependent on the assets.
- **Manifest** is supposed to be a directory for the data of a virtual representation.
- **Data-Object** is a virtual representation (requirements, assurances, type information, instance data, services) of an asset, which is supposed to be a real device and can be updated through the administration interface.

The standard IEC 62832 – Digital Factory, which is currently under development, will define a template for structuring the data within the AAS.

B. Location of Administration Shells

There are several possibilities investigated to host administration shells of an Industrie 4.0 system. This influences the required mechanisms for accessing, integrating, and updating administration shells. An Industrie 4.0 System consists *"of I4.0 components and components of a lower communication and presentation classification (CPC)"* [5]. One Industrie 4.0 *"system may be present as a component in a further I4.0 system"* [5]. The different submodels of one administration shell also do not have to be located at the same physical place as its corresponding administration shell. The location of administration shells can be divided into central administration shells, component-centric administration shells, and distributed administration shells.

The **central administration shells** can be divided into two sub groups, a single central administration shell for the whole system or all administration shells of the system at one central location. The central administration shell for the whole systems contains its components in terms of assets. Each new component is added to the system's central administration shell. This has the advantage that the one administration shell has all information of the whole system. In contrast to that, each component can also be represented with an own administration shell at one central location. Each new component is added in terms of a creating a new administration shell at the same location as the existing ones. In both cases the administration shell(s) is(are) located at a central point within the Industrie 4.0 system. Having all administration shells and its submodels at one place, on the one hand provides a single information point, but on the other hand lets the central administration shell become a bottleneck for communication connections.

For the **component-centric administration shell** each component of a system hosts its own administration shell together with its submodels. This supports structuring of administration shells that corresponds to the structure of the components within the system. Each hierarchy needs to add glue between the administration shells of its lower level, while not repeating information but adding value. To host its own administration shell, each component needs enough memory to store the corresponding data, which is often not available. To get information from the administration shell the component has to be accessed, which might influences the execution behavior of the system due to lots of network traffic or fast component accessing rates.

For the **distributed administration shell** neither the administration shell nor its submodels need to be hosted on the component itself. The component therefore only links to its administration shell or its submodels. Components therefore only need to be accessed to fetch the location of the desired information, which saves memory on the components and reduces the payload of the messages from and to the component.

C. Implementation of Administration Shells

Sofar the available work on AAS is mainly theoretical, and only a few implementation attempts exist. [7] investigated different standards to be used as submodels in an AAS. They provided device description submodel for Industrie 4.0 components based on field device integration (FDI). Furthermore they investigated the suitability of AutomationML as generic data exchange format as it is able to carry a wide range of descriptions and contain references to other submodels or submodel contents.

The ZVEI together with the Chair of Process Control Engineering RWTH Aachen University initiated the open source project openAAS[2], which is an open source implementation of an AAS [11]. They build an AAS utilizing OPC UA information models to store the data as well as using the OPC UA communication infrastructure to access it. A generic model for structure and a detailed property model define the frame for the data in an openAAS-based AAS.

In this work we will also investigate the possibilities of AutomationML and directly accessing AutomationML data. However as AutomationML can easily be represented in OPC UA information models [9] a migration to OPC UA is possible at a later stage.

III. Access an Administration Shell

Memory is still a limited resource on PLCs. Therefore, it might not be possible to store the whole content of a component's AAS directly within the PLC. Typically also real-time capabilities are required for the PLC program execution. Real-time execution might be influenced by AAS requests and updates to an AAS located on the PLC. However real-time data produced by the PLC should also be easily accessible through AAS mechanisms. Within this contribution we only consider the distributed administration shell approach, since from our point of view the AAS combines data distributed within different types of systems due to its different purpose and time of creation.

Considering a distributed administration shell, each component needs to access the information linked within its corresponding AAS through the network. Any PLC controlling a specific component also might need to access a remote AAS to, for example, initialize with structural information or configuration data. This use-case is especially of interest in plug and play scenarios where generic components are plugged into a system and retrieve their instance specific configuration from the remote part of the AAS. PLCs therefore need to request information during run-time from a remote database.

Due to the distributed nature of Industry 4.0 systems but also the message based communication with a remote database during run-time, event triggered runtime environments (RTEs) approaches are better suited than cyclic RTEs. Events are used to explicitly define the execution order of function blocks (FBs) in distributed systems, but also to trigger the production of network messages and the react on received network messages. The remote database access for any RTE has to be a non blocking service, in order to keep the reactive real-time behavior of PLCs, so that the system is still able trigger alarms, immediately react on emergency stops, and provide the required control quality of service.

A. Run-time Support for Administration Shell Access

To realize a non blocking AAS access during run-time, a *query client* functionality is needed, which sends a request to the AAS. For distributed control systems the IEC 61499 standard [12] defines an architecture and a modeling language, relevant in the Industry 4.0 context. For the implementation a IEC 61499 based RTE is proposed. The IEC 61499 also defines a communication model for bidirectional transactions. One part of this model is the `CLIENT` FB, as illustrated in Figure 2.

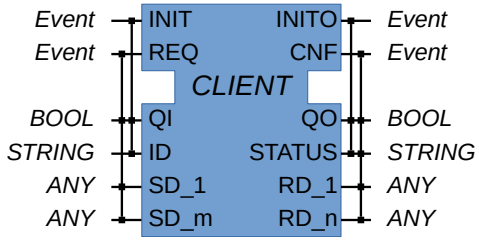

Fig. 2: IEC 61499 standard client function block, according to [12].

According to IEC 61499-1 the generic behavior of a `CLIENT` FB consists of two main functions:

1) on initialization (i.e., `INIT` event) connect to the remote entity
2) on a `REQ` event take the data inputs `SD_x`, generate based on their values a data packet, send it to the remote entity, wait for a response, process the response,

[2]https://github.com/acplt/openAAS

apply it to the data outputs RD_x, and finally notify the application with an CNF event.

To realize the AAS access also configuration data to establish a connection to the AAS is required. In the generic IEC 61499 model the client's ID data input is used to configure the connection to the remote entity, in our case the AAS. In IEC 61499 no special definitions on the structure or content of the configuration parameters is given. In [13] a very flexible and extensible approach for configuring and handling communication is presented. We will follow this approach also for our AAS connection configuration. Listing 1 illustrates the structure of a configuration ID for different kinds of query languages. By parsing the configuration ID the *query client* FB get the necessary information to open a connection to the desired AAS, and authenticate for read and write operations.

```
queryLanguage[IP:port; AASname; user; psw]
```

Listing 1: Structure of a configuration ID for a query client.

The IEC 61499 standard client FBs are able to handle any number of send inputs SD_x and receive outputs RD_x of any standard compliant data type. For the *query client* only one send input, containing the query's data, is required, since the *query client* is also able to process any new query at every new request event. Currently we investigate request which expect only one result. Therefore the *query client* only requires one output for the received data. The *query client* therefore gets the query for the AAS through its SD_1 input, and provides the result of the query through its RD_1 output.

Since each query needs to be associated to its result, the *query client* would block until the result arrived. To prevent this blocking a *query handler* is introduced, which runs within an own background thread. The *query handler* contains a *list of requests*, which collects the requests from the clients. When a request event is received by the *query client* an element is registered in this *list of requests*. The entire request is set through the SD_1 data input of the *query client*. As long as there are elements registered within the *list of requests*, the elements of this list are processed one after each other according the first come first serve principle. The *query handler* processes the *list of requests*, by *sending each request* through the network to the AAS and returning the query result to the calling *query client*. The corresponding query element is then removed from the *request list* by the *query handler*.

After receiving the query result, the *query client* confirms by updating its RD_1 data output and sending a CNF event. Figure 3 illustrates the proposed concept.

B. Design Time Support for Administration Shell Access

The AAS or one of its submodels is represented by the *data* part of the *database server*. To access the information provided by any AAS, data-object or submodel, a language is needed to query the desired data. The language used to query data from the administrations shell should be able not only to gather data but also to calculate data from the queried data.

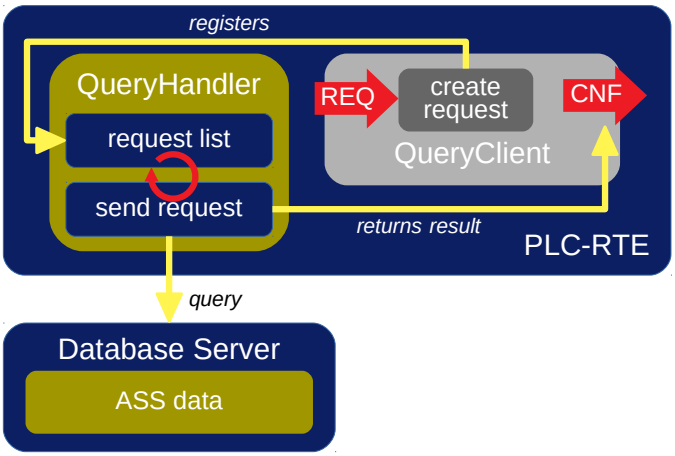

Fig. 3: Concept for an event driven access of a run-time environment to an asset administration shell during run-time.

For modeling a submodel of a plant automation markup language (AML) has been used, since it is able to connect information of different formats. Furthermore AML has the advantage that it utilizes data that is anyhow generated during the engineering process from different engineering tools. The AML is then loaded into BaseX[3], which is a light-weight, high-performance, robust, and scalable extensible markup language (XML) Database engine. This database engine also provides a powerful Client/Server Architecture to handle concurrent read and write operations of multiple clients. Clients are provided in many different languages, such as C/C++ which is suitable for real-time capable PLC RTEs.

BaseX supports the XQuery 3.1 processor, which suggests XQuery [8] as query language. The IEC 61499 standard allows to use other complying programming languages within its algorithms. XQueries can therefore be supported as algorithm language. For the IEC 61499 implementation Eclipse 4diac™[4] XQuery support can be achieved by defining an additional language type. Besides that for each algorithm of this language type a data output has to be set. This data output then contains the query, which is supposed to be connected to the SD_1 data input of a *query client*. The query should also be able to contain placeholders, where information from the run-time can be added to the query.

XQuery allows to define variables with the following syntax: `let $variableName := x`. The XQuery variable concept has to be merged with data input usage inside IEC 61499 algorithms. This means that a variable `$variableName` should be a local XQuery variable, in case it has been definded by the `let` keyword or it is a replacement of the FB's data interface element.

For each FB type code needs to be generated, which has to be compiled for the RTE, that the new FB can be used on the RTE. During the code generation phase the defined query

[3]http://basex.org/
[4]http://www.fordiac.org

string has to be adapted, whenever variables are supposed to be inserted that change during run-time. This proposed concept, consisting of a RTE part as well as a integrated development environment (IDE) part, enable a direct use of query languages for PLC programming.

IV. Demonstration

The starting point of the demonstration scenario is the logistics part of an aluminum cold rolling mill, provided by the SMS Group. The logistics components of the aluminum cold rolling mill are described in a single AML file, where component properties are not all explicitly represented but calculateable. This AML file is the content of our BaseX data base supposed to be the implementation of an administration shell.

The component under consideration is a roller conveyor, which is part of the transportation system and transports pallets with aluminum coils up to 20t. Each roller conveyor component is supposed to be a mechatronic entity, with its own PLC, controlling and monitoring the operation of the component. Roller conveyors can have two neighbors, one on its left and one on its right side. Properties such as the maximum rotation speed of the motor are not explicitly added but calculateable from other specified properties.

To extend the transportation system, an engineer adds a new roller conveyor within the AML file as well as its neighbors. The newly added roller conveyor is then put at the desired place. During startup each generically programmed component connects to the system's administration shell and configures itself with instance specific values such as maximum velocity and neighboring components.

Figure 4 illustrates an IEC 61499 application that initializes a component during startup. In the first step the three clients are initialized. Afterwards the component triggers each client by its `INITO` event. The clients connect to an BaseX database to fetch the desired information described by a corresponding XQuery request applied at their `SD_1` inputs. The result from the BaseX database is transmitted to the roller conveyor component via the `RD_1` output of the specific client.

A proper XQuery for the clients is provided by the corresponding roller conveyor component (top right FB in Figure 4). Figure 5 illustrates a simplified execution control chart (ECC) for the initialization of the roller conveyor component. When an `INIT+` event arrives, four algorithms are executed. Each produces one of the desired XQueries for fetching the maximum velocity of the roller conveyor's motor as well as its two neighbors, respectively. Afterwards an `INITO` event is sent, which triggers the corresponding clients with the specific XQuery.

```
xquery
let $component := //InstanceHierarchy/*/InternalElement[
    @Name=$name]
let $compType := tokenize($component/RoleRequirements/
    @RefBaseRoleClassPath/data(),'/')[last()]
let $rollerTable :=  $component/InternalElement/
    RoleRequirements[contains(@RefBaseRoleClassPath,'
    RollConveyor')]/..
```

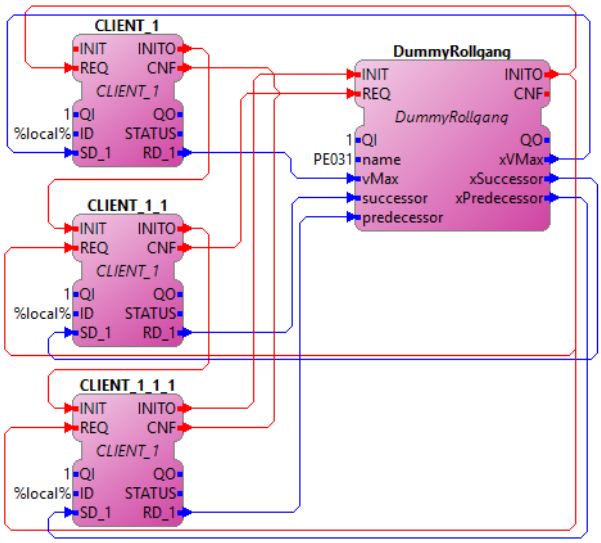

Fig. 4: IEC 61499 application for the initial configuration of a roller conveyor.

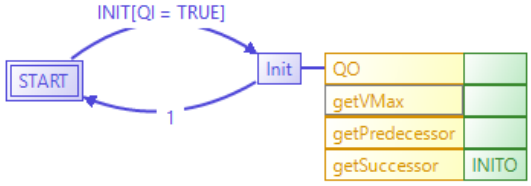

Fig. 5: Simplified execution control chart for the initial configuration of the roller conveyor.

```
let $rollDiameter := $rollerTable/Attribute[@Name='
    RollDiameter']/Value/data()
let $link := tokenize($component/InternalLink/
    @RefPartnerSideA/data(),':')[1]
let $motor := //InternalElement[@ID=$link]/
    RoleRequirements[contains(@RefBaseRoleClassPath,'
    ElectricMotor')]/..
let $gearFactor := $motor/Attribute[@Name='Gearfactor']/
    Value/data()
let $rotSpeed := $motor/Attribute[@Name='Rotational
    Speed']/Value/data()
let $linSpeed :=  $rotSpeed div (60.0 * $gearFactor) *
    (2.0 * 3.1416 * 0.5 * $rollDiameter)
return concat(round-half-to-even($linSpeed, 3),' m/s')
```

Listing 2: Get maximum speed of a roller conveyor from an administration shell.

Listing 2 illustrates the XQuery used as algorithm to get the maximum speed of a specific roller conveyor. The roller conveyor is specified by the `name` data input, which has been set to `PE031`, as shown in Figure 4. This data input value is used during run-time to replace the `$name` variable of the XQuery. Even if BaseX is an XML Database engine the results of an XQuery do not need to be XML, but can be formated by the XQuery, which means that no XML decoding is required on the RTE.

For scenarios where submodels of the administration shell have to be changed during run-time additional commands

are needed. During run-time it should be possible to add information into an existing administration shell or delete information from an existing administration shell. Listing 3 illustrates how a node <myNode> could be added to the root element of an administration shell accessible by XQueries.

```
xquery
insert node <myNode> into /root
```

Listing 3: Insert component description into administration shell.

```
xquery
delete node //root/MyNode[@name='myName']
```

Listing 4: Delete component from administration shell.

Listing 4 illustrates how to delete a specific node according to its name myName from an administration shell accessible by XQueries.

V. Conclusion

This work analyzed the term (asset) administration shell to provide a mechanism, which enables PLCs accessing content of its corresponding AAS during run-time. The proposed mechanism implements a non blocking client for PLCs to query information from a database server during run-time, where the database contains the AAS. Even if the proposed mechanism has been implemented for an IEC 61499 based RTE it can also be applied to IEC 61131 based RTEs or any C++ application.

For demonstration an AAS for the logistic part of an aluminum cold rolling mill has been defined in terms of an AML file. The AML file described the logistic components of the aluminum cold rolling mill as well as their connections. The AML has been loaded into an XML database, which can be accessed by XQueries. The PLC has been enabled to send XQueries during run-time to query, create and delete data from the AAS implementation. With that we could show that an AML-based AAS is suitable for the task of providing configuration information to generic automation components.

Compared to other approaches as provided by openAAS for example, this approach does not provide an information model for components, but uses an information model provided by AutomationML. It also supports collecting data and calculating the desired information from the collected data. The XQuery itself does not have to be interpreted on the device itself, since it is sent as a composed String literal to the database server. The returning result can be formated by the XQuery to get a PLC conforming data type, such as String, Boolean or Integer.

The experiments showed that XQueries require detailed knowledge of the underlying XML structure and can get quite large. Especially for queries that determine a pallet path through the system is expected to be more complex than the calculation of the maximum speed of a roller conveyor. Especially for AML files the XML structure gets quite complex. This complexity is directly reflected in the XQueries. It can be confirmed that the BaseX XML database meets its proposed performance for processing XQueries and the result is therefore available on the PLC immediately, also for requests with calculation as shown in Listing 2.

To overcome the complexity of XQueries caused by the AML structure, SPARQL Protocol And RDF Query Language (SPARQL) will be investigated as query language. Since SPARQL also provides math functions, it might reduce the complexity of queries, especially for queries which determine pallet paths through the system. Using SPARQL for future experiments requires an ontology based AAS implementation, for future experiments.

Furthermore we will investigate how to link the structural AAS data stored in the database with run-time data in the PLC, and where to store this run-time data, for a full-fledged implementation of an AAS.

Acknowledgment

This work is funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS16022N through the project BaSys 4.0 (Basic System Industry 4.0) (http://www.basys40.de/).

References

[1] DIN SPEC 91345: *Reference Architecture Model Industrie 4.0 (RAMI4.0)*. April 2016.

[2] P. Adolphs, S. Auer, H. Bedenbender, M. Billmann, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele-Dunsche, G. Koschnick, H. Koziolek, L. Linke, R. Pichler, F. Schewe, K. Schneider, B. Waser: *Structure of the Administration Shell*, 3rd ed. Federal Ministry for Economic Affairs and Energy (BMWi), April 2016.

[3] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister: *The role of the Industrie 4.0 Asset Administration Shell and the Digital Twin during the life cycle of a plant*, International Conference on Emerging Technologies and Factory Automation (ETFA), 2017.

[4] H. Bedenbender, M. Billmann, U. Epple, T. Hadlich, M. Hankel, R. Heidel, O. Hillermeier, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele-Dunsche, G. Koschnick, H. Koziolek, L. Linke, S. Lohmann, F. Palm, R. Pichler, S. Pollmeier, B. Rauscher, F. Schewe, K. Schneider, B. Waser, I. Weber, M. Wollschlaeger, M. Zinn: *Examples of the Asset Administration Shell for Industrie 4.0 Components - Basic Part*, ZVEI - German Electrical and Electronic, April 2017.

[5] VDI Verein Deutscher Ingenieure e.V., VDI/VDE-Gesellschaft, Mess- und Automatisierungstechnik (GMA): *Industrie 4.0 Terms*. [Online] Available: www.vdi.de/industrie40

[6] VDI/VDE Gesellschaft für Mess- und Automatisierungstechnik: *Industrie 4.0 Service Architecture Basic concepts for interoperability*. 2016.

[7] C. Diedrich, M. Riedl: *Engineering and integration of automation devices in I40 systems*. at - Automatisierungstechnik, 64(1): 41–50. January 2016.

[8] W3C XML Query Working Group: *XQuery 3.0: An XML Query Language*. [Online] Available: https://www.w3.org/TR/xquery-30/

[9] M. Schleipen, A. Lüder, O. Sauer et al: *Requirements and concept for Plug-and-Work*. Special Issue: Industrie 4.0 / Jürgen Beyerer, Jürgen Jasperneite, Olaf Sauer. at - Automatisierungstechnik, 63(10), pp. 801–820, October 2015.

[10] B. Brandenbourger, M. Vathoopan, A. Zoitl: *Engineering of Automation Systems using a Metamodel implemented in AutomationML*. Industrial Informatics (INDIN), 2016 IEEE 14^th International Conference on, pp. 363–370, 2016.

[11] F. Palm, U. Epple: *openAAS - Die offene Entwicklung der Verwaltungsschale*. In: Automation 2017: 18. Branchentreffen der Mess- und Automatisierungstechnik, Baden-Baden, Juni 2017.

[12] IEC SC65B: *IEC 61499-1: Function blocks for industrialprocess measurement and control systems – Part 1: Architecture*. International Electrotechnical Commission. 2005.

[13] M. Hofmann, M.N. Rooker, A. Zoitl: *Improved Communication Model for an IEC 61499 Runtime Environment*, 16^th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2011.

View publication stats