Industry 4.0 Resource Monitoring - Experiences With Micrometer and Asset Administration Shells

Miguel G. Casado[a,b], Holger Eichelberger[b]

[a]*University of Valladolid, 47002 Valladolid, Spain*
[b]*Software Systems Engineering, Universität Hildesheim, 31141 Hildesheim, Germany*

Abstract
Industry 4.0/IIoT setups are often distributed and consist of hundreds of sensors, devices or machines. Nowadays, powerful edge devices enable the execution of software services like Artificial Intelligence close to production machines to reduce latency. Although several supporting Industry 4.0 platforms do exist, industrial stakeholders aim for more open, interoperable and vendor-neutral solutions.

In this paper, we focus on the monitoring of runtime properties in Industry 4.0 for services and (edge) devices. We propose a realization based on existing components like Micrometer, MQTT and upcoming standards such as the Asset Administration Shell. We discuss patterns for their integration, the impact of the patterns on performance in terms of an experiment as well as experiences that we made so far.

Keywords
Industry 4.0, IIoT, monitoring, Micrometer, performance, Asset Administration Shell, BaSyx, MQTT

1. Introduction

The digitization of industry increases the performance of technical systems and their processes, but also their complexity. Particular complexity arises as Industry 4.0 installations are inherently distributed. In such settings, edge devices allow for handling the high data frequencies and volumes provided by sensors and production machines. Moreover, modern edge devices enable the safe execution of hard real-time machine control operations and (soft real-time) IT functionality, such as Artificial Intelligence (AI).

In the BMWi-funded project IIP-Ecosphere[1], we research approaches to address typical shortcomings in Industry 4.0 systems regarding openness, interoperability, or vendor-neutrality. At the heart of IIP-Ecosphere, we develop a prototypical AI-enabling Industry 4.0 platform that allows for exploring the approaches developed in the project. As a core functionality, the platform shall automatically deploy and manage (AI-)services in a uniform manner on heterogeneous resources such as edge devices, on-premise servers, or cloud resources.

Within this context, we focus in this paper on a runtime monitoring approach for (AI-)services. Our research question is: *How to design and realize efficient resource and service monitoring in an open, interoperable and vendor-neutral manner?* Our approach combines a recent Industry

SSP'21: Symposium on Software Performance, November 09–10, 2021, Leipzig, Germany
miguel.gomez@alumnos.uva.es (M. G. Casado); eichelberger@sse.uni-hildesheim.de (H. Eichelberger)
http://www.sse.uni-hildesheim.de (H. Eichelberger)
0000-0002-2584-5558 (H. Eichelberger)

[1]https://www.iip-ecosphere.eu/

4.0 information/interface model, the Asset Administration Shell (AAS) [1], with protocols such as MQTT or AMQP[2] and a vendor neutral monitoring frontend (Micrometer[3]).

For these components, we introduce three integration patterns and analyze them in a performance experiment. While a local integration of monitoring and AAS performs best, it implies also higher resource usage, a potential obstacle for resource-constrained edge devices. In contrast, a distributed installation increases response times. We show that a combination with publish-subscribe protocols helps balancing resource usage and response time.

Structure of the paper: In Section 2, we provide background information on used standards and technologies. We present our approach to Industry 4.0 resource and service monitoring in Section 3 and discuss initial experimental results in Section 4. In Section 5, we briefly outline related work, and in Section 6 we conclude the paper and give an outlook on future work.

2. Background

In this section, we introduce standards and technologies that we selected to meet the goals of IIP-Ecosphere. For the realization, we aim at maximizing the use of Open Source.

Micrometer is an Open Source vendor-neutral application metrics facade with support for monitoring tools like Prometheus or Dynatrace. Micrometer defines so-called meters such as timers, gauges, or counters, which also carry their unit of monitoring, e.g., items/s.

A range of protocols is used in IIoT, including (legacy) machine protocols such as Fieldbus (the field level is not in our scope as stated in [2]) to Message Queuing Telemetry Transport (MQTT) or Advanced Message Queuing Protocol (AMQP). **MQTT** and **AMQP** are publish-subscribe protocols for the transport of binary payloads via a broker server.

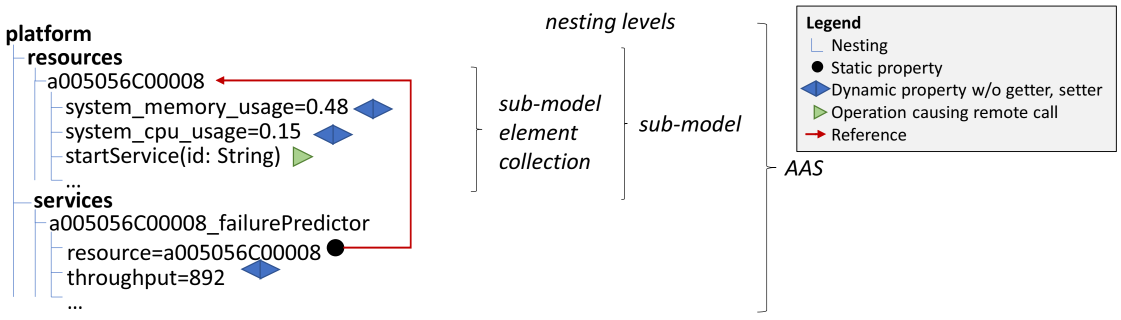

Figure 1: Structure of an Asset Administration Shell, here for compute resources and services.

Recently, the **Asset Administration Shell (AAS)** [1] was defined to uniformly model Industry 4.0 "assets", i.e., products, machines, their digital twins, or even software components. AAS is currently in a standardization driven by industrial groups in different countries, e.g., the Industrial Digital Twin Association (IDTA) in Germany or the Smart Manufacturing Profiles initiative in the U.S. Although AAS is rather new and still in development, the IIP-Ecosphere

[2]https://mqtt.org/, https://www.amqp.org/
[3]https://micrometer.io/

partners opted for an extensive exploration. Essentially, as illustrated in Figure 1, an AAS is a nested structure containing typed properties, operations or references between elements. An AAS can be static, dynamic regarding values or structures, or active through callable operations.

Eclipse BaSyx[4] is the de-facto AAS reference implementation. In BaSyx, an AAS is represented as a REST structure that can be served by a local process or deployed to a remote server. Properties and operations can be realized through functors attached to an AAS that perform local or remote method invocation, e.g., via the BaSyx Virtual Automation Bus (VAB) protocol.

3. Approach

Our goal is to collect runtime monitoring data reflecting the resource usage and performance of devices and services in Industry 4.0 installations. A realization shall provide access to the monitored information for individual devices and services, but also serve as a basis for efficient aggregation over all elements in an installation. Moreover, IIP-Ecosphere imposes further requirements [2], among them in particular: 1) AAS shall be used for all component/service interfaces, 2) IIoT protocols such as MQTT or AMQP can be used instead of AAS, e.g., for soft realtime streaming, 3) monitoring operations shall be faster than a typical 8 ms machine pace, 4) (de facto) standards shall be used wherever possible to foster interoperability.

To realize the monitoring approach, we made some basic decisions (cf. [3] for more details): For representing runtime measures in an uniform fashion, we rely on Micrometer. However, Micrometer focuses on local meters, i.e., there is limited built-in support to access remote/published meters. Thus, we propose a set of proxy instances, which can be loaded with JSON information obtained from published meters, e.g., through a REST client. Further, the monitoring results shall be represented in terms of AAS structures as illustrated in Figure 1, which may also contain JSON. This allows client implementations, e.g., the AASX Package Explorer[5], to access monitored properties through polling as desired by our stakeholders. Alternative forms, e.g., pushing of the data through publish-subscribe, can be realized side-by-side.

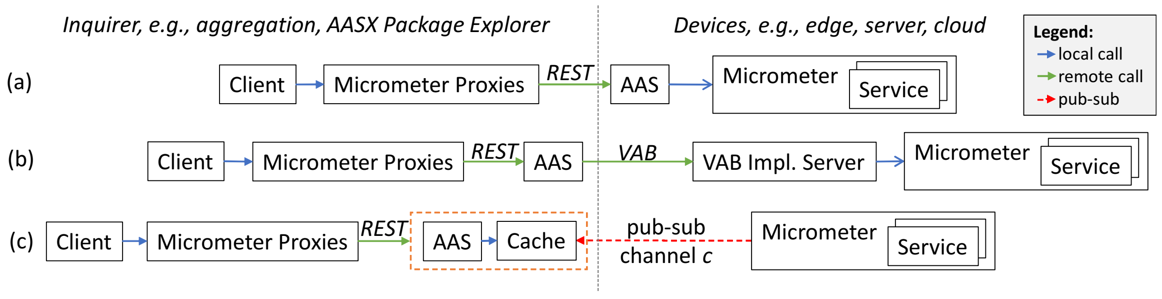

Figure 2: AAS integration patterns for AAS-based runtime monitoring.

The mentioned components can be combined in several ways, implying individual performance impacts. Figure 2 illustrates some alternative patterns, all including services with defined

[4]https://www.eclipse.org/basyx/
[5]https://www.eclass.eu/anwendung/verwaltungsschale.html

meters on the device side, either a local or remote AAS to represent the monitored data, and a client reading the remote meters via Micrometer proxies. Our patterns are:

a) A **local AAS** executing the Micrometer environment, which, in turn, observes the services. The AAS polls the monitored data when a property is accessed, i.e., its functors perform local calls. To publish the AAS, an HTTP server process must be running on the device.

b) A **remote AAS** communicating through a (light-weight) AAS implementation server on the device. As in a), the data is polled, here through (VAB) remote method calls.

c) **Remote pub-sub AAS:** A scheduled task on the device publishes the data on a channel *c*. The AAS subscribes to *c* and holds the received data in a local cache bound to the AAS property functors. If communication from the AAS to the device is needed, e.g., to reconfigure the monitoring, a protocol as in b) can additionally be used.

As a BaSyx AAS is represented as a REST structure, all patterns access the AAS from the client through REST. Patterns a) and b) exclusively rely on AAS functionality and particularly follow the examples provided by BaSyx. In contrast, in pattern c) we connect device and AAS/client using asynchronous communication and transform the polling from pattern b) into an approach that relies on local calls to a cache instead of remote method invocations. In this pattern, further components can subscribe to the channel and operate even without access to the AAS. Patterns a) and b) may also be combined with some form of pushing, which we do not detail here.

4. Results

To analyze the efficiency of the patterns introduced in Section 3, we present now an experimental evaluation of the individual response times. We realized the patterns as variants of the same Java code[6]. As basis, we use a simple workload in terms of two interconnected Spring Cloud Stream services (Spring version 3.1.1). The Micrometer (version 1.6.4) instances represent more than 50 default Spring and custom meters. A BaSyx AAS (github version of January 2021) defines a representative structure of properties and operations for all variants. A client reads the AAS through our Micrometer proxies and records the individual response times.

We use the local AAS pattern as baseline. However, Spring and BaSyx conflict when being executed in the same JVM. Thus, we use corresponding meters without service workload as fallback baseline. For the remote AAS pattern, the client polls meter data via VAB. As the meters are published by Spring in terms of REST by default, we opted for a VAB implementation server that accesses the meter information through an internal REST client [3]. For the pub-sub AAS, a regular task publishes a JSON summary of the meters via AMQP (HiveMQ client 2020.4).

As experimental environment, we use Windows 10 (Dell 7490, Intel i7-8650U, 1.9MHz, 32 GByte RAM, Open JDK 13+33) to emulate a development setting and Ubuntu Linux 20.04 (VMWare VM with 6 vCores on a Xeon E5-2650v4, 2.2GHz, 12 GByte RAM, Open JDK 13.0.7) as a server environment. Both setups use local network only. A single iteration of a variant reads 62 AAS properties representing gauges, counters and timers. Initial experiments in [3]

[6]Replication package https://doi.org/10.5281/zenodo.5571817

indicated, that the measurements stabilize at around 175 iterations, probably due to caching or JVM/JIT settling operations. Here, we perform 200 iterations for each variant. To prevent warm-up outliers, we exclude the results of first two iterations per variant.

		Windows		Linux	
		avg [ms]	σ [ms]	avg [ms]	σ [ms]
Gauges	local a)	1,3	0,3	0,4	0,1
	VAB b)	5,5	1,1	2,2	0,7
	AMQP c)	1,2	0,6	0,7	0,3
Timer	local a)	0,8	0,3	0,4	0,2
	VAB b)	5,5	1,2	2,6	1,0
	AMQP c)	1,6	0,8	1,0	0,8
Counter	local a)	0,6	0,3	0,3	0,2
	VAB b)	3,6	0,9	1,7	0,8
	AMQP c)	1,1	0,5	0,6	0,3

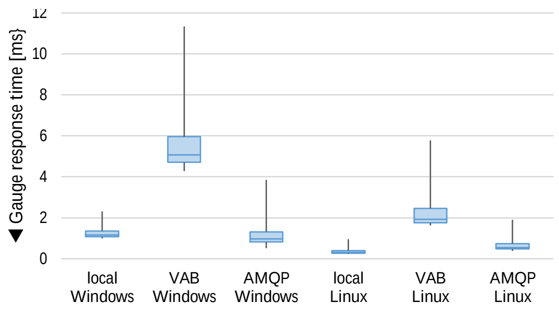

Figure 3: Evaluation results, response times for all patterns (left) and box plots for gauges (right).

Figure 3 summarizes the results, in particular response times of the three variants for both, Windows and Linux, as well as representative boxplots (min/max-whiskers) for the gauges, currently the most relevant meter for us. As the VAB-based implementation of pattern b) involves three protocols (AAS-REST, VAB, Spring-REST), it is not surprising that the response times are factor 4-7 higher than the baseline. Additional experiments show that this difference can roughly be attributed in equal amounts to VAB and Spring-REST. If we replace those two protocols by publish-subscribe in pattern c), we can achieve significantly better response times (factor 1-2.5 of the baseline) and less fluctuations, while still not serving the AAS on the device. Further, we found that gauges may incur a response time drop of factor 2-3, if they are updated on request rather than asynchronously, e.g., by a schedule. Moreover, each AAS access in BaSyx creates a temporary network connection. A high access frequency may cause an outage of system resources requiring a cool-down of 120 seconds between two experiment runs.

The measures depend on the used components and the setup, which may limit generalizability. Although the use of REST or VAB may be rather specific here, we believe that the remote pub-sub pattern can be beneficial in similar (REST) settings. Comparable results can also be expected for MQTT, as other work in IIP-Ecosphere indicates that MQTT and AMQP behave rather similar for the load applied here. Moreover, the Windows setup is less representative, as it was mainly intended to support the judgement of ad hoc measures during development.

5. Related Work

We addressed a combination Industry 4.0 protocols, Micrometer and AAS. For IIoT protocols, a body of work is published with a certain focus on MQTT. Two interesting related works are [4] on broker performance for IoT edge computing and [5] on MQTT protocol latency over different gateways. Regarding Micrometer, publications typically focus on applications, e.g., in the context of Spring [6]. Further, there is an increasing amount of publications on AAS and, in particular, BaSyx. Currently, the focus there is on evaluating maintainability [7] or on use cases/demonstrators [8]. According to our knowledge, there is currently no work that combines

AAS, Micrometer and IIoT protocols for the monitoring of resource or software services.

6. Conclusions

Industry 4.0 systems do not only require complex, distributed setups, but sometimes also the use of prescribed protocols and standards. Moreover, future Industry 4.0 systems aim at increased openness, interoperability and vendor-neutrality. In this context, we analyzed whether Asset Administration Shells, Micrometer and protocols such as MQTT or AMQP can be combined to realize efficient runtime monitoring of resource and service properties. We proposed three basic integration patterns and discussed the impact on the performance in terms of an experiment. In essence, a more complex publish-subscribe implementation can outperform techniques that are (mostly) shipped with the utilized implementation frameworks. Moreover, efficient updates of meters may require specific attention, e.g., asynchronous schedules.

In future, we plan to explore the scalability of the approach when monitoring a larger set of devices. On the one side, an overview of all monitored values through an Asset Administration Shell is demanded by our stakeholders and can be provided as shown in this work. On the other side, we expect that a realization of further platform components, e.g., a monitoring aggregator, can efficiently be achieved through an integrated publish-subscribe approach.

Acknowledgments

IIP-Ecosphere is partially supported by the German Ministry of Economics and Energy (BMWi), grant 01MK20006D.

References

[1] S. Bader, E. Barnstedt, H. Bedenbender, et al., Details of the Asset Administration Shell, 2020. URL: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html.

[2] H. Eichelberger, C. Sauer, A. S. Ahmadian, M. Schicktanz, A. Dewes, G. Palmer, C. Niederée, IIP-Ecosphere Platform Requirements, 2021. doi:10.5281/zenodo.4485774.

[3] M. G. Casado, Service and device monitoring on devices in IIP-Ecosphere, 2021. URL: https://sse.uni-hildesheim.de/studium-lehre/beispiele-guter-ausarbeitungen/.

[4] H. Koziolek, S. Grüner, J. Rückert, A Comparison of MQTT Brokers for Distributed IoT Edge Computing, in: Software Architecture, 2020, pp. 352–368.

[5] S. B. Kenitar, S. Marouane, A. Mounir, A. Younes, A. G. Gonzalez, Evaluation of the MQTT Protocol Latency over Different Gateways, in: Intl. Conf. on Smart City Applications, 2018.

[6] F. Gutierrez, Monitoring, Apress, 2021, pp. 383–397.

[7] K. Esper, F. Schnicke, Evaluation of the Maintainability Aspect of Industry 4.0 Service-oriented Production, in: Intl. Conf. on Industry 4.0, Artificial Intelligence, and Communications Technology, 2020, pp. 8–14.

[8] F. Schnicke, T. Kuhn, P. O. Antonino, Enabling Industry 4.0 Service-Oriented Architecture Through Digital Twins, in: Software Architecture, 2020, pp. 490–503.