Interacting with the Arrowhead Local Cloud: On-boarding Procedure

Ani Bicaku[1,2], Silia Maksuti[1,2], Csaba Hegedűs[3], Markus Tauber[1], Jerker Delsing[2] and Jens Eliasson[2]

[1]*University of Applied Sciences Burgenland* - Eisenstadt, Austria
[2]*Luleå University of Technology* - Luleå, Sweden
[3]*AITIA International Inc.* - Budapest, Hungary

***Abstract*—Industrial automation systems are advancing rapidly and a wide range of standards, communication protocols and platforms supporting the integration of devices are introduced. It is therefore necessary to design and build appropriate tools and frameworks that allow the integration of devices with multiple systems and services. In this work we present the Arrowhead Framework, used to enable collaborative IoT automation and introduce two support core systems, SystemRegistry and DeviceRegistry, which are needed to create a chain of trust from a hardware device to a software system and its associated services. Furthermore, we propose an on-boarding procedure of a new device interacting with the Arrowhead local cloud. This ensures that only valid and authorized devices can host software systems within an Arrowhead local cloud.**

I. Introduction

High product variability and – at the same time – shortened product life-cycles require agile and flexible production structures that can be reconfigured rapidly for new product demands [1]. This degree of flexibility, as well as other fast growing market demands such as sustainability, efficiency and competitiveness cannot be achieved by traditional automation. The existing technologies can not fully address these requirements, that is why a lot of work is being done to transform the potentials of the upcoming digital revolution, known as Industry 4.0, into business success. Industry 4.0 is driven by ongoing developments and supported by emerging technologies and concepts, including the Internet of Things (IoT), System of Systems (SoS), Cyber Physical Systems (CPS), cloud computing and Service Oriented Architectures (SOA) [2]. Whilst these technologies already exist, they have to be adapted to meet specific requirements. Here, serious effort is needed in the scientific community to address issues like security, latency, etc.

Traditional hierarchical automated architectures, such as ISA95 [3] based process control systems, are migrating to SOA-compliant control systems. First migration steps have been successfully applied and many traditional systems already integrate communication technologies that make them compatible for SOA based architectures. Further improvements in automation systems demand more real-time data, which is beyond the capability of existing systems. The use of cloud technologies has become popular in production environments and thus, enterprises can move or migrate the load within and between their own data centers [4].

These above mentioned technologies have been in focus of the Arrowhead project, aiming to address the capability of building large system of systems, based on IoT by using SOA fundamentals. The Arrowhead Framework [5], developed as part of this project, is used to facilitate the creation of local automation clouds, which include devices, various application-specific systems and services to perform the automation tasks and provide a boundary to the open internet and the outside activities. This way Arrowhead is enabling local (on-site), real time performance and security, paired with simple and cheap engineering, while enabling scalability through multi-cloud interactions. The Arrowhead Framework architecture is composed of a number of systems: mandatory core systems, automation support core systems and application systems.

To assure that an Arrowhead local cloud is not compromised upon the introduction of new devices hosting a number of software systems (SW-systems), it is important to establish a chain of trust from the hardware device to a hosted SW-system and its associated services. To support this approach, in this paper, we introduce two additional Arrowhead support core systems, the SystemRegistry system and the DeviceRegistry system, used to provide a local cloud storage for SW-systems and devices, respectively.

One of the fundamental criteria of IoT-platforms is the need to integrate efficient security mechanisms [6]. To address this, we further introduce an on-boarding/bootstrapping procedure including both, the SystemRegistry and the DeviceRegistry systems. A new device that wants to interact with the Arrowhead local cloud should authenticate itself using a manufacturer issued certificate, which can be stored e.g. in its Trusted Platform Module (TPM). Each SW-system hosted in this device should be provided with an Arrowhead issued runtime certificate. Thus, every local cloud should have a central Certificate Authority (CA) that issues and signs these SW-systems runtime certificates. This local cloud CA system is the root of trust within its local cloud.

The remainder of this work is organized as follows:
Section II presents an overview of existing IoT platforms. This is followed by the description of the Arrowhead Framework, the SystemRegistry and the DeviceRegistry systems in Section III. In Section IV, we present the on-boarding procedure of a new device interacting with the Arrowhead local cloud and we conclude this work in Section V.

II. Related Work

Various approaches and frameworks have been proposed in support of the IoT. A recent review of the literature [7], summarizes and compares the existing frameworks and platforms used for IoT applications with the aim to identify gaps in the current state of the art. In this review the platforms are classified in: (i) IoT frameworks for home automation including the IPSO Alliance [8], the IoTivity [9] and the LWM2M framework [10], (ii) cloud-based IoT platforms including ThingWorx [11] and Xively [12], (iii) device to device platforms such as IzoT [13] and ThingSquare [14] and (iv) platforms for cloud gateway integration including Intel, Microsoft and IBM. The authors highlight that a successful IoT framework should support secure APIs for third party systems, communication protocol interoperability and should enable management of heterogeneous networks.

LISA2.0 [15] introduces a lightweight, distributed and embedded IoT bus architecture using a node centric networking architecture. It is designed to provide interoperability and mobility for facilitating the implementation of SOA in resource-constrained devices. LISA offers service discovery, registration and authentication features, but in comparison with the Arrowhead Framework it does not provide system or device registry to create a chain of trust from the hardware device to the software system and its associated services.

Nimbits [16] is a cloud-based IoT platform used to control, manage and monitor IoT devices from the cloud. The framework consist of: (i) Nimbits Server, which enable the user to record, store and process data from IoT devices, (ii) nimbits.io, a library for building IoT software, (iii) Nimbits Android, an application in GoogleStore and (iv) Nimbits public cloud, which is an instance of the Nimbits Server. The communication is possible via XMPP messaging protocol and the web-services use HTML request under JSON format. It provides a data compression, an alert management and data calculation based on simple mathematic formulas.

Ericsson IoT-Framework [17] developed by Ericsson Research, the Swedish Institute of Computer Science (SICS) and Uppsala University is a PaaS with centralized architecture including a REST API, data storage functionality and OpenId access control. The purpose of this framework is to enable scalability and is designed to operate as a service in the backend. It provides an example with wireless sensors connected to the IoT framework by a gateway, where the sensors push time-stamped data (stream) to the framework or the data can be pulled periodically by the framework. These streams can be available via a RabbitMQ, allowing external users to push their data and subscribers to consume the needed data.

OpenIoT [18] is an IoT platform offering discovery and collection of data from mobile sensors within cloud computing infrastructures. This is achieved through a publish/subscribe middleware broker used for collecting data from the mobile sensors. In terms of secure authentication, OpenIoT offers a privacy and security module where users should provide username and password for authentication.

Authentication solutions of IoT devices relies on traditional machine-to-machine (M2M) systems, which makes the authentication difficult on a large scale. This challenge is addressed by the Generic Bootstrapping Architecture (GBA) [19] supporting device authentication at the transport layer. One technology for bootstrapping in resource-constrained environments is Light Weight Machine to Machine (LWM2M) from Open Mobile Alliance (OMA)[10]. OMA-LWM2M supports management of a device in terms of configuration, security and connectivity. LWM2M can be used over SMS, CoAP and HTTP. LWM2M is a core building block for the IPSO Smart Objects framework from IPSO Alliance. Multiple frameworks and projects have considered using secure hardware, such as TPM to provide a root of trust for identifying inconsistencies and security issues. Several studies, for e.g. [20], [21], and [22] analyzed the opportunities to provide root of trust by investigating different types of hardware for bootstrapping trust, security mechanisms, techniques for recording platform state and existing proposed or deployed approaches.

Most of the identified frameworks have a REST API, meaning that IoT platforms are working towards web services. Only a few of the identified platforms (e.g. LISA2.0, HAT, Ericsson IoT and OpenIoT) support service discovery mechanisms. In comparison with the identified frameworks, mostly based on centralized solutions, Arrowhead Framework is an industrial framework connecting distributed applications with the aim to integrate legacy systems with communication and distributed SoA. Additionaly, we describe a step-by-step on-boarding procedure, build on our previous work [23], including SystemRegistry and DeviceRegistry systems, which are needed to create a chain of trust from a hardware device to a software system and its associated services.

III. Arrowhead Framework

A. Arrowhead Framework Architecture

The objective of the Arrowhead Framework architecture is to facilitate the creation of local automation clouds and enable local real time performance and security, interoperability, simple and cheap engineering and scalability through multi cloud interaction [5]. The architecture addresses the move from large monolithic organisations towards multi-stakeholder cooperations, thus addressing the high level requirements in today's society such as sustainability, flexibility, efficiency and competitiveness. The architecture is build based on the SOA fundamentals: (i) loose coupling, which supports autonomy and distributed services, (ii) late binding, which makes possible to use the information any time by connecting to the correct resources and (iii) lookup that is used to publish services to notify others about endpoints, which information then can be used to discover already registered services.

In terms of the Arrowhead Framework, a service is an information exchange from a providing system to a consuming system. A SW-system provides and/or consumes multiple services. A hardware device is a piece of equipment, machine or hardware with computational, memory and communication capabilities, which can host one or several SW-systems.

The Arrowhead Framework architecture is composed of a number of systems, including mandatory and automation support core systems and application systems. The mandatory core systems include ServiceRegistry, Authorisation and Orchestration systems. The ServiceRegistry system is used to provide storage of all active services registered within a local cloud and enables the discovery of them. The Authorisation system is used to provide authentication, authorisation and optionally accounting of service interactions. The Orchestration system is used to provide a mechanism for distributing orchestration rules and service consumption patterns, thus providing service endpoints to specific requests. The application systems are used to implement application functionalities and services aiming to fulfill application requirements. They should be consuming at minimum the three mandatory core services of the Arrowhead local cloud, thus ServiceDiscovery produced by ServiceRegistry system, AuthorisationControl produced by Authorisation system and OrchestrationStore produced by Orchestration system. In order to define an Arrowhead local cloud the three mandatory core systems and at least one application system deployed are required, as shown in Figure 1.

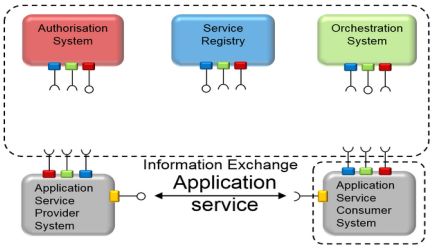

Fig. 1. Arrowhead Framework architecture

The automation support core systems such as, PlantDescription, EventHandler, Configuration, etc., are used to facilitate automation application design, engineering and operation. They should be able to support the implementation of plant automation, housekeeping within the local cloud, inter-cloud service exchange, system and service interoperability and secured on-boarding/bootstrapping procedure of a local cloud.

To assure that the Arrowhead local cloud is not compromised upon the introduction of new devices and systems an on-boarding/bootstrapping procedure is needed to make possible their registration to the specific registries and to be available within the local Arrowhead network. In order to support this procedure, we introduce two additional support core systems, the SystemRegistry system and the DeviceRegistry system. They are necessary to create a chain of trust from a hardware device to a hosted SW-system and its associated services. Both systems are explained in the following sections.

B. SystemRegistry System

An Arrowhead Framework SW-system is what is providing and/or consuming services. A system can be the service provider of one or more services and at the same time the service consumer of one or more services. A system is implemented in software and executed on a hardware device.

The SystemRegistry system is used to provide a local cloud storage holding the information on which systems are registered within a local cloud, meta-data of these registered systems (vendor, requirements, Service Level Agreement (SLA) etc.) and the services these systems are designed to consume. The SystemRegistry holds for the Arrowhead local cloud unique system identities for systems deployed within it.

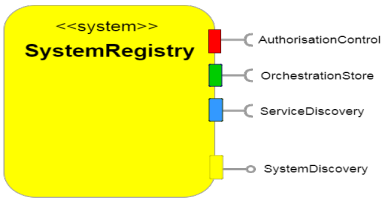

Fig. 2. SystemRegistry system in the Arrowhead Framework. It consumes the Arrowhead mandatory core services and produces SystemDiscovery service used to register, unregister and find systems within the local cloud

The SystemRegistry system produces one service, SystemDiscovery, and consumes the three mandatory core services, as shown in Figure 2. All Arrowhead Framework SW-systems within a local cloud shall register within the SystemRegistry system by using the SystemDiscovery service. As such SystemDiscovery is regarded as a well known service, and shall be accessible using a multitude of SOA protocols (e.g. REST, CoAP, MQTT, etc). The SystemDiscovery service is used to register and unregister systems and their produced services, as well as to find systems among the registered systems.

The SystemDiscovery service interface is defined according to Figure 3. The interface is defined using three main methods. The *publish* method is used to register a system. The systems will contain a symbolic name as well as a physical endpoint. The instance parameter represents the endpoint information that should be registered. The *unpublish* method is used to unregister a system that no longer should be used. The instance parameter contains the necessary information to find the system to be removed. The *lookup* method is used to find and translate a symbolic system name into a physical endpoint, IP address and a port. The query parameter is used to request a subset of all the registered systems in the SystemRegistry system based on a specified criteria, for e.g. meta-data.

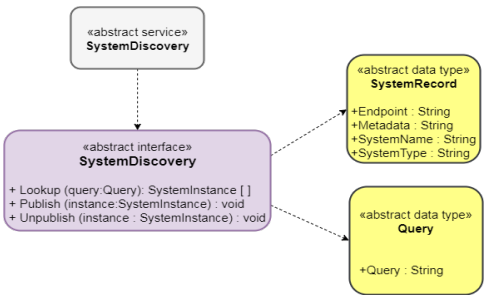

Fig. 3. SystemDiscovery service interface and data types

The information model holds two data types: (i) SystemRecord and (ii) Query, as shown in Figure 3.

The SystemRecord data type contains information of a system endpoint, such as name and physical address. System metadata can also be added here. The metadata should be provided using key pairs such as, encode = syntax, *e.g. encode = xml*, compress = algorithm, *e.g. compress = exi*, semantic = XX, *e.g. semantic = senml*. The Query datatype represents a query that can be sent to the system provider in order to filter the stored systems to return a proper subset.

C. DeviceRegistry System

An Arrowhead Framework compliant device is a piece of equipment, machine, hardware, etc. with computational, memory and communication capabilities, which hosts one or several Arrowhead Framework SW-systems and can be bootstrapped in an Arrowhead local cloud. Any other device, equipment, machine, hardware, component etc., is non-Arrowhead compliant.

The DeviceRegistry system is used to provide a local cloud storage holding the information on which devices are registered within a local cloud, meta-data of these registered devices including a list of the systems that are deployed to each of them. The DeviceRegistry system holds for the Arrowhead local cloud unique device identities.

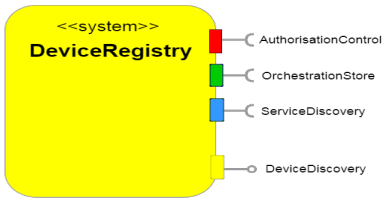

Fig. 4. DeviceRegistry system in the Arrowhead Framework. It consumes the Arrowhead mandatory core services and produces the DeviceDiscovery service used to register, unregister and find devices within the local cloud

The DeviceRegistry system shall be accessible using different SOA protocols (e.g. REST, CoAP, MQTT). It provides one service, DeviceDiscovery and consumes the three mandatory core services, as shown in Figure 4. The DeviceDiscovery service is used to register and unregister devices as well as to find devices among the registered devices.

The DeviceDiscovery service interface is defined according to Figure 5. The interface is defined using three main methods. The *publish* method is used to register a device. The device will contain a symbolic name. The instance parameter represents the endpoint information that should be registered. The *unpublish* method is used to unregister a device that no longer should be used. The instance parameter contains the necessary information to find the device that should be removed. The *lookup* method is used to find and translate a symbolic device name into a physical endpoint, such as Media Access Control (MAC) address, IP address, hostname and port.

The information model holds two data types: (i) DeviceRecord and (ii) Query, as shown in Figure 5. The DeviceRecord data type contains information of a device, such as name, IP address, port and MAC address. Device metadata can also be added here, same as explained for SystemDiscovery service. The Query data type represents a query that can be sent to the provider in order to filter the stored devices and to return a proper subset.

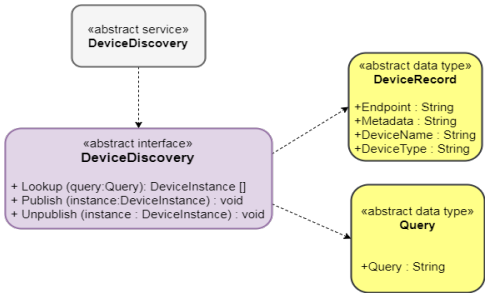

Fig. 5. DeviceDiscovery service interface and data types

As mentioned above, both systems shall be accessible using SOA protocols in order to increase interoperability via Uniform Resource Identifiers (URI) for identifying services by bringing them to the browser, meaning that the services can be retrieved by using a standard URI (e.g. $http://app.arrowhead.eu/ahf/Temp1/8090/$). This enables the communication with any device, system or service in the Arrowhead local cloud while neither requiring to know about the exact details of the underlying infrastructure.

IV. On-boarding Procedure

In the previous section we have presented the characteristics of the Arrowhead Framework, as well as SystemRegistry and DeviceRegistry systems in general. In this section, the on-boarding procedure and its underlying steps are described. To illustrate the proposed approach, in Figure 6 is provided a simple use case where a new device interacts with the core systems of the Arrowhead local cloud.

A. Use Case

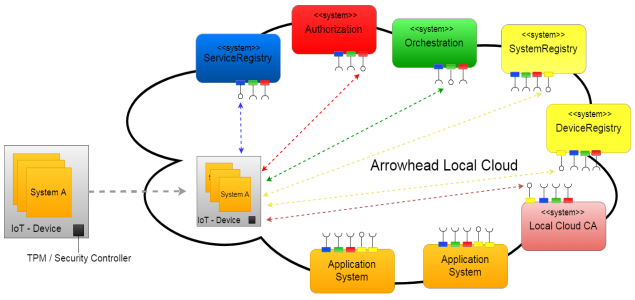

Fig. 6. The on-boarding procedure of an IoT device, with IoT system A requesting to interact with the Arrowhead local cloud

Here, a new device produced by a specific vendor (e.g. Siemens), containing a security controller (e.g. TPM), wants to interact with the Arrowhead local cloud. In general, a TPM hardware is a special variant of a security controller with a standardized feature set for the integration into various

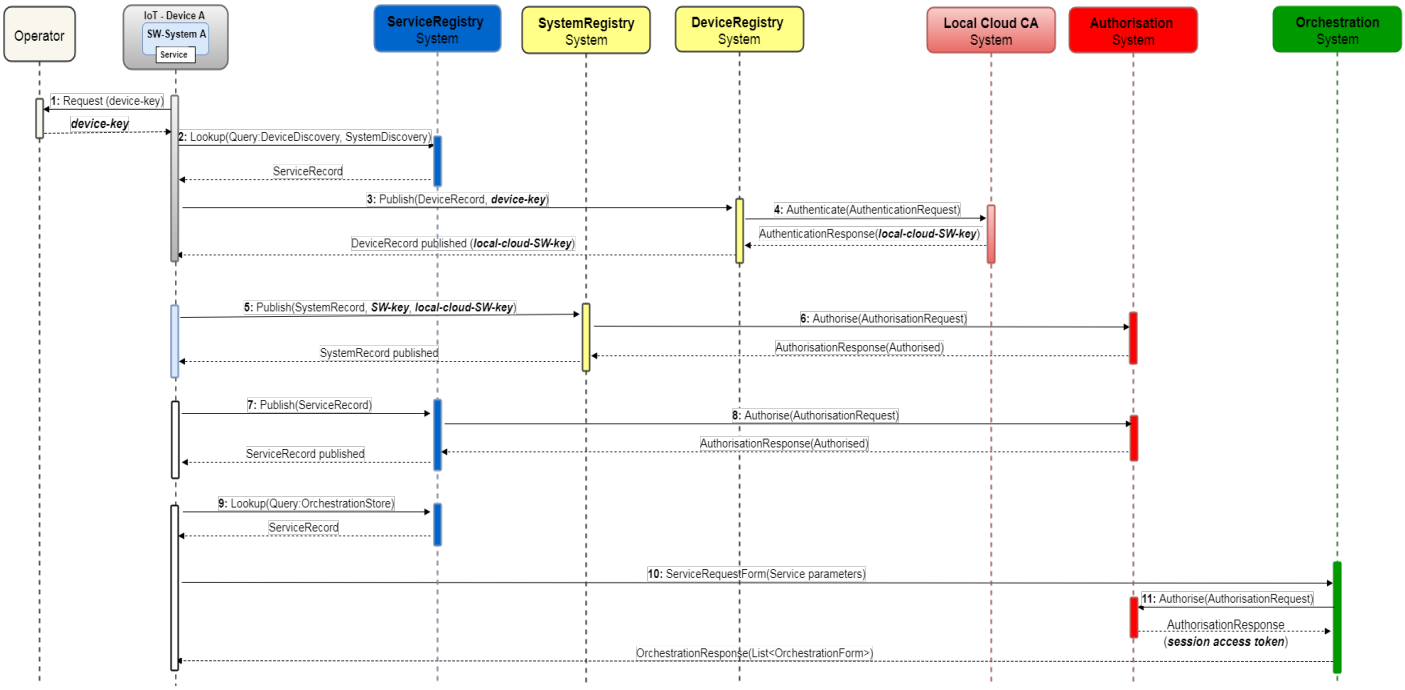

Fig. 7. The sequence diagram for the on-boarding procedure of a new device interacting with the Arrowhead local cloud

computing platforms [24]. TPMs can be used to identify a device and to also check the software of the device on startup to ensure that the OS and programs are not manipulated.

The root of trust is a non-volatile memory able to securely store cryptographic material. This material can be used for encryption or signing of data to ensure the integrity, confidentiality and authenticity. Furthermore, TPMs provide a standardized interface, which makes it very easy to integrate them in any gateway. The TPM certificates for the new device might not be Arrowhead issued but own manufacturer issued.

B. On-Boarding/Bootstrapping Procedure

The bootstrapping procedure is needed between a device and a local cloud, that have no previous knowledge of each other. This is especially needed when the device that wants to connect to the Arrowhead local cloud is a very resource constrained device and may not have an UI.

Here, we are assuming that the ServiceRegistry system, the DeviceRegistry system and the mandatory core systems are established in the network in a secure way. To assure that the cloud is not compromised upon the arrival of a new device, it is important to establish a chain of trust from the new hardware device, to its hosted SW-systems and their services. For this purpose a secured initiation or bootstrapping process starting from the device is needed. Bootstrapping procedure makes possible that the device, systems and services are authenticated and authorized to connect to the Arrowhead local cloud.

As described in the use case above a new device with at least one SW-system, installed in a secure way with a software key (SW-key) and associated services shall be introduced to the Arrowhead local cloud. To enable a device to be trusted, it has to have specific hardware providing storage and computation of pre-shared authentication keys, which shall be secure and tamper free. Such hardware security controllers (e.g. TPMs), are provided by a couple of vendors (e.g. Siemens, Infineon, etc.). The device further needs both a network interface and some type of short range communication (e.g. Near-Field Communications, NFC [25]). This will enable an operator identification of a device via key authentication over the near-field communication link. Such authentication will allow the generation of a device-key to be transferred to and stored in the security controller. Following we provide a step-by-step on-boarding procedure as shown in Figure 7.

1) We are assuming that a new device has a device-key (e.g. a manufacturer issued certificate). This might be acquired through an operator identification in a two-way communication schema or the device has a device-key stored in the TPM.
2) The new device uses the DNS-SD of Arrowhead (e.g. auto-configured through DHCP) as ServiceDiscovery and looks up the endpoints of the DeviceDiscovery and SystemDiscovery services.
3) Using the obtained endpoints the new device begins the registration with the DeviceRegistry through the DeviceDiscovery "publish" interface authenticating itself with the device-key obtained previously, in step 1).
4) The DeviceRegistry verifies the received device-key with the local cloud CA system and registers the device if authententication is successful. The Arrowhead local cloud CA issues local-cloud-SW-keys (i.e. runtime certificates) for each SW-system hosted on the device.
5) The SW-system on the new device begins the registration

with the SystemRegistry through the SystemDiscovery "publish" interface authenticating itself with its new, Arrowhead issued local-cloud-SW-key obtained in step 4).
6) The SystemRegistry verifies both the local-cloud-SW-key and the SW-key with the Authorisation system and registers the SW-system if authorised. This is repeated for all SW-systems residing on the new device.
7) The SW-system next registers its produced services with the ServiceRegistry using the "publish" interface of the ServiceDiscovery service over a secure channel.
8) The ServiceRegistry verifies with the Authorisation system and, if authorised, allows the registration of the service(s). This is repeated for all services running on the SW-system.
9) Every SW-system starts its execution and look up the endpoint of the Orchestration service in the ServiceRegistry.
10) Every SW-system requests Orchestration using the obtained endpoint for every connection it wants to establish, services it wants to consume.
11) The Orchestration system cross-checks with authorisation and an Arrowhead session access token [26] might be generated for every connection attempt during the orchestration process.

When any of the SW-systems hosted on the device shuts down, the device contacts the DeviceRegistry system and requests a destroyal of the local-cloud-SW-keys. The local cloud CA system puts them in the blacklist and removes them from the registry.

Such on-boarding procedure requires that the cloud operator has already configured the mandatory core systems, for e.g., orchestration rules, authorisation status and other support core systems such as, PlantDescription system. It uses the meta-data information stored in the SystemRegistry and DeviceRegistry to compare the existing plant model with the current implementation.

V. Conclusion

In this paper we have introduced two additional automation support core systems of the Arrowhead Framework: SystemRegistry and DeviceRegistry systems. These are both used to provide a special storage, the SystemRegistry system stores SW-systems registered within the local cloud and the DeviceRegistry system stores devices registered within the local cloud. These systems are needed to create a chain of trust from the hardware device, to its hosted SW-systems and their services whenever a new device wants to interact with the Arrowhead local cloud. To show how this is done, we have presented a step-by-step on-boarding procedure, including the Arrowhead mandatory core systems and the two additional systems presented in this paper.

Acknowledgment

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 737459 (project Productive4.0) and from the partners' national programmes/funding authorities.

References

[1] S. Weyer, M. Schmitt, M. Ohmer, and D. Gorecky, "Towards industry 4.0-standardization as the crucial challenge for highly modular, multi-vendor production systems," *IFAC-PapersOnLine*, vol. 48, no. 3, 2015.
[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
[3] B. Scholten, *The road to integration: A guide to applying the ISA-95 standard in manufacturing*. Isa, 2007.
[4] P. Friess, *Digitising the industry-internet of things connecting the physical, digital and virtual worlds*. River Publishers, 2016.
[5] J. Delsing, *IoT Automation: Arrowhead Framework*. CRC Press, 2017.
[6] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "Iot security: ongoing challenges and research opportunities," in *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 230–234.
[7] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, "A survey of commercial frameworks for the internet of things," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–8.
[8] Z. Shelby and C. Chauvenet, "The ipso application framework draft-ipso-app-framework-04," *Avaiable online: http://www. ipso-alliance. org/wp-content/media/draft-ipso-app-framework-04. pdf*, 2012.
[9] A. Subash, "Iotivity–connecting things in iot," *TIZEN Development Summit*, 2015.
[10] S. Rao, D. Chendanda, C. Deshpande, and V. Lakkundi, "Implementing lwm2m in constrained iot devices," in *Wireless Sensors (ICWiSe), 2015 IEEE Conference on*. IEEE, 2015, pp. 52–57.
[11] P. E. I. Solutions, "Platform technology: Thingworx. 2016," *URL: https://www. thingworx. com/(cited on page 25)*.
[12] N. Sinha, K. E. Pujitha, and J. S. R. Alex, "Xively based sensing and monitoring system for iot," in *Computer Communication and Informatics (ICCCI), 2015 International Conference on*. IEEE, 2015, pp. 1–6.
[13] "Izot platform," https://www.echelon.com/izot-platform, (Accessed on 01/03/2018).
[14] Y. J. Heo, S. M. Oh, W. S. Chin, and J. W. Jang, "A lightweight platform implementation for internet of things," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2015, pp. 526–531.
[15] B. Negash, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Lisa 2.0: lightweight internet of things service bus architecture using node centric networking," *Journal of Ambient Intelligence and Humanized Computing*, vol. 7, no. 3, pp. 305–319, 2016.
[16] "Nimbits platform," https://www.nimbits.com/, (Accessed on 01/03/2018).
[17] "Github-ericssonresearch/iot-framework-engine," https://github.com/EricssonResearch/iot-framework-engine, (Accessed on 01/03/2018).
[18] J. Kim and J.-W. Lee, "Openiot: An open service framework for the internet of things," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 89–93.
[19] G. Bajko and T. K. Chan, "Method, apparatus and computer program product providing bootstrapping mechanism selection in generic bootstrapping architecture (gba)," Dec. 27 2011, uS Patent 8,087,069.
[20] B. Parno, J. M. McCune, and A. Perrig, "Bootstrapping trust in commodity computers," in *Security and privacy (SP), 2010 IEEE symposium on*. IEEE, 2010, pp. 414–429.
[21] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security & Privacy*, vol. 12, no. 4, pp. 29–37, 2014.
[22] Z. Malik and A. Bouguettaya, "Reputation bootstrapping for trust establishment among web services," *IEEE Internet Computing*, vol. 13, no. 1, pp. 40–47, 2009.
[23] O. Carlsson, P. P. Pereira, J. Eliasson, J. Delsing, B. Ahmad, R. Harrison, and O. Jansson, "Configuration service in cloud based automation systems," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*. IEEE, 2016, pp. 5238–5245.
[24] T. Morris, "Trusted platform module," in *Encyclopedia of cryptography and security*. Springer, 2011, pp. 1332–1335.
[25] R. Want, "Near field communication," *IEEE Pervasive Computing*, vol. 10, no. 3, pp. 4–7, 2011.
[26] S. Plosz, C. Hegedus, and P. Varga, "Advanced Security Considerations in the Arrowhead Framework," in *DECSoS*, September 2016, pp. 1–13.