

Enhancements of the Arrowhead Framework to Refine Inter-cloud Service Interactions

Csaba Hegedűs

AITIA International Inc.

Telecommunication Division

48-50 Csetz János str., Budapest, Hungary, H-1039

Email: hegeduscs@aitia.ai

Dániel Kozma, Gábor Soós and Pál Varga

Dept. of Telecommunications and Media Informatics

Budapest University of Technology and Economics

2 Magyar Tudósok krt., Budapest, Hungary, H-1117

Email: {dkozma, gsoos, pvarga}@tmit.bme.hu

Abstract—Solving interoperability and integrability issues for the Internet of Things – and especially for the automation domain – is a great challenge. The Arrowhead Framework addresses these issues by providing the innovative concept of Local Clouds, within which some well-defined core systems support the applications, based on the principles of Service Oriented Architectures. Several additions and refinements of the Arrowhead Framework are introduced in this paper – from system and service hierarchies through core system capabilities, as well as interfaces for application systems. The presented enhancements support the advanced functionalities for System-of-Systems, including Quality-of-Service and resource allocation capabilities, inter-cloud servicing, advanced authentication and authorization, and event handling. All mandatory core systems of the Arrowhead Framework are affected somehow in the process: the Service Registry, the Orchestrator and the Authorization System.

I. INTRODUCTION

Arrowhead aims at nothing less than to create a full-scoped framework that enables collaborative automation of networked embedded devices. Arrowhead is an Artemis Innovation Pilot Project [1], with the grand challenge of solving the interoperability and integrability of almost any device.

In its evolution, Generation 1 of the Arrowhead Framework provided the fundamental principles and preliminary implementation for the mandatory core services – whereas Generation 2 provided further core services and their mature implementations. The Generation 3.1 of the framework introduced the handling of events, security certificates, and Quality of Service constraints. Furthermore, it clarified the core services used by the application demonstrations for the Arrowhead project – and their implementations got further polished. The current paper focuses on the core elements of the Arrowhead Framework, and discusses its Generation 3.2 that encompasses new architectural elements. These enable to enhance the Arrowhead-compatible System-of-Systems' functionalities with ease. The main objective of this generation (3.2) is to integrate fundamental solutions to the newly arisen, but essential requirements:

- advanced authentication and authorization capabilities,
- dynamic service orchestration process,
- real-time communication (QoS) and resource allocation functionalities,

- event and alarm handling capabilities,
- the integration of the automation support services (such as the Historian or Translation Services [2]),
- and inter-Cloud servicing capabilities.

II. THE ELEMENTS OF THE ARROWHEAD FRAMEWORK

A. Local Automation Clouds

Having heterogeneous systems (or architectures) work together is not an easy problem to solve – especially in the automation domain. Legacy protocols, commercial off-the-shelf products and monolithic architectures often cripple interoperability and dynamic reconfigurability. Several frameworks have been created to solve such problems of the Industrial Internet of Things [3]. Among others, the Arrowhead Framework uses the approach of Service Oriented Architectures (SOA) [4] to tackle this problem: it aims at providing interoperability by facilitating the service interactions within closed or at least separated automation environments, called local clouds.

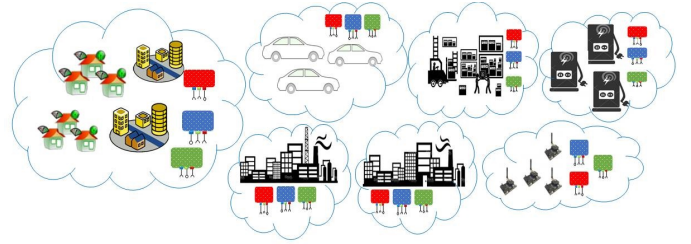


Fig. 1. Arrowhead Local Clouds deployed in different domains

These *Arrowhead Local Clouds* might fulfill various tasks and can have their own sets of appointed stakeholders (e.g. their operators or developers). These Local Clouds all have their operational boundaries, let those be functional, geographical or network-segmented. Nevertheless, they must be governed through their own instances of the *Arrowhead Core Systems*, as Figure 1 suggests. These are clouds in the sense that they use common resources: the Core Systems of that domain. These common Core System resources (e.g. related to Service Registry, Authorization or Orchestration) are used by all kinds of other entities – applications – in the network, and can also be implemented in a distributed way. This means

that the scalability of the Local Cloud mostly depends on the scalability of the Core Systems (e.g. by implementing them using virtualization and Web Services [5] technologies).

B. Systems and Services

In such Arrowhead Local Clouds there can be an arbitrary number of *Systems* that can provide and consume *Services* from one another: they create and finish *servicing instances* dynamically in run-time. Figure 2 depicts these relations. Services are defined so that loose-coupling, late binding, and service discoverability can be realized.

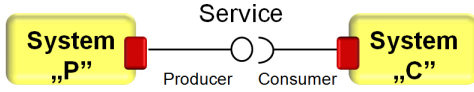


Fig. 2. Service providers and consumers

In this context, Arrowhead-compatible Systems are not just service endpoints, but can be realized by a wide range of devices: from a small temperature sensor, in certain use cases, up to very complex cyber-physical systems of a production plant. However, there might be operational limits of how these Systems can be handled, i.e. certain hard-wired service interactions cannot change at all, supposing that an actuator can only access certain sensors. The Arrowhead Framework has to abide by such implementational constraints that shape this world of the industrial Internet of Things.

Therefore, Arrowhead-compatible Systems must use the mandatory *Core Systems and their Core Services* provided by the Framework to realize their operational targets (as shown in Figure 3): to communicate with other Systems using dedicated application protocols. There are two main groups of the Core Systems: the mandatory ones that need to be present in each Local Cloud, and the automation supporting ones that further enhance the core capabilities of a Local Cloud.

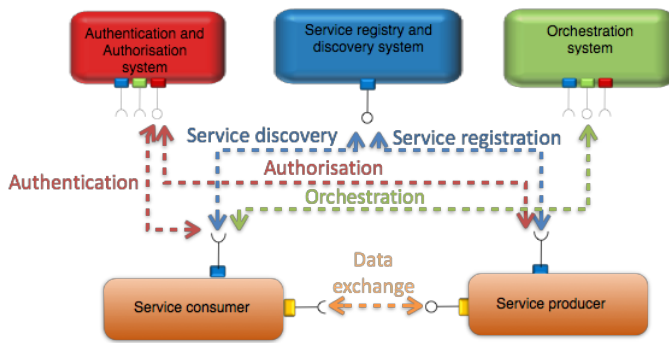


Fig. 3. Arrowhead Core Systems and their use

C. Core Systems and Services

In accordance with [6], this work attributes the same functionality to each of the Core Systems. There are three mandatory Core Systems (the row I. in Figure 4).

The *Service Registry* stores all the Systems (that are currently available in the network) and their service offerings.

Systems have to announce their presence, and the services they can offer. The registry takes note of this information when systems come on-line, and might have to revoke them when they go off-line.

The *Authorization System* – as its name suggests – manages authentication and authorization (AA) tasks, however, it covers some other security-related issues as well (e.g. certificate handling).

The *Orchestrator* is responsible for instrumenting each System in the cloud: where to connect and what to consume. It instructs Systems so by pointing towards specific Service Providers to consume specific Service(s) from. This has to be done by a simple request-response sequence, which ends with the requester System receiving a Service endpoint. After these, the System is obligated to consume from that Service instance.

There are further automation supporting Core Systems available. These either have been (II.) or currently being integrated in the framework (III.), as Figure 4 suggests.

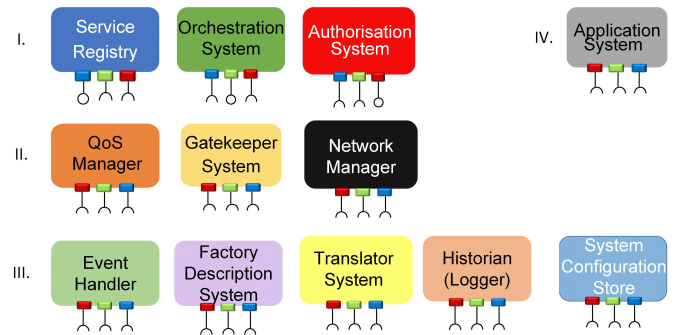


Fig. 4. Arrowhead Core Systems in scope

D. Inter-Cloud servicing

Arrowhead Local Clouds are generally autonomous and independent from each other, since they are deployed separately for various reasons. However, there are specific cases, where Systems from different clouds need to consume Services from one another. To this end, an inter-Cloud servicing architecture was introduced in [7] based on two, new automation supporting Core Systems: the Gatekeeper and the Network Manager.

This architecture builds heavily upon the Core Systems' exclusive involvement in facilitating and managing the connections (servicing instances) within their authority: in this sense, the Core Systems are responsible for all other Application Systems and their data handled in the Local Cloud. The Gatekeeper provides essentially two services for the mandatory Core Systems. The first is the Global Service Discovery (GSD) process, which aims at locating adequate service offerings in neighboring Clouds. The second is the Inter-Cloud Negotiations (ICN) process, in which mutual trust is established between two Clouds and the actual connection between endpoints is then built up.

After these processes, the Network Manager is responsible for creating the data path between the Service Provider and the

Consumer. We assumed here that the Core Systems are configured properly, so that they can securely establish connections with other Clouds using the services of the Gatekeeper. This setup includes some security considerations taking place, e.g. declaring inter-Cloud access rights (from and to other Local Clouds) in the Authorization Systems, or setting up the trusted neighborhood domains.

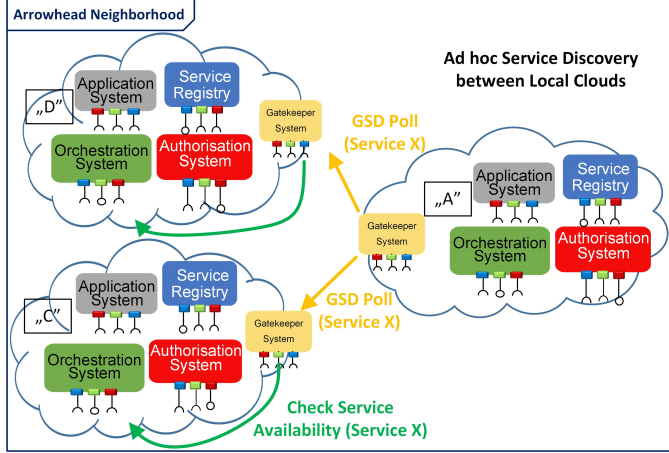


Fig. 5. Inter-Cloud service discovery between Gatekeepers

III. CURRENT ENHANCEMENTS OF THE ARROWHEAD FRAMEWORK

This section describes how the Core Systems operate, in order to support the requirements set up in the introduction. Since this generation 3.2 of the framework is aiming at merging several supporting Core Systems in one modular Framework, implementation-wise it uses numerous placeholders that are reserved for future use. For the reasons discussed in section II-B, the proof-of-concept implementation uses HTTP Web Services. In order to comply with RESTful [8] requirements, the Core Systems themselves are separately deployable REST resources (with specific URL, certificate and JSON payload structures) created in Java EE Jersey [9] with Grizzly containers and using MySQL Server. This makes the Framework easy to deploy and operate with high scalability. The Web Services approach also alleviates several interoperability problems while making resource-constrained devices Arrowhead-compatible.

A. System and Service hierarchies

One of the key innovations here is the strict logical separation between the Systems and Services abstractions. This conforms with SOA and DNS-SD [10], and unifies the data structures in the Core Systems. Therefore, a new System and a new Service hierarchy has been introduced to support (not just inter-Cloud) interoperability and matchmaking, as shown in Figure 6. The Service hierarchy stems from the documentation structure of Arrowhead [11]: Services are identified with a name (identified in the Service Definition document) that is unique in its Service Group. This Service abstraction only

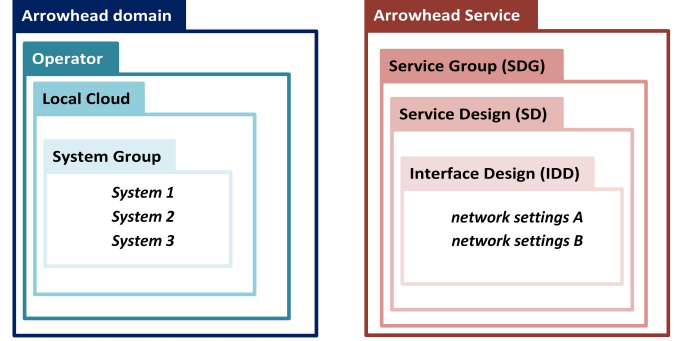


Fig. 6. Systems and Services defined for Arrowhead

defines the functionality provided, but a Service can be implemented many ways using appropriate application protocols (e.g. by SOAP, COAP or RESTful Web Services - WS). A Service Provider can therefore even implement multiple *interfaces* of the same Service. Metadata is also associated with Services (for every instance separately): currently key-value pairs can be added to describe them (e.g. "unit"="°C") embedded in JSON. This will enable the integration of standardized data descriptors, e.g. SenML [12] or IPSO Smart Objects [13]. It could also use case specific data ontologies (i.e. MIMOSA for describing maintenance related communications [14]), and can fit into other interoperability strategies, such as the Interoperability Repository System described in [15].

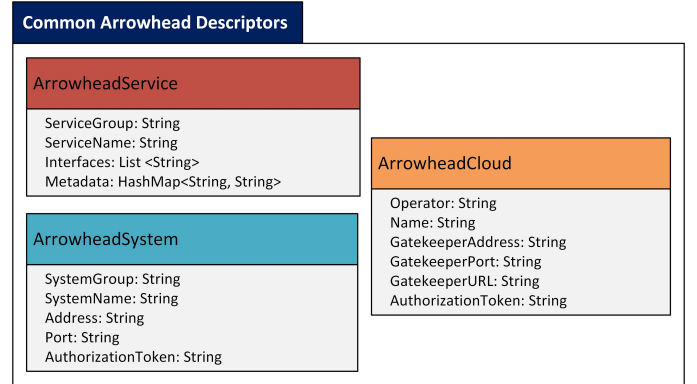


Fig. 7. Unique identifiers for the various Arrowhead elements

Meanwhile, the unified Cloud and System hierarchy follows a general, process-oriented principle. Local Clouds are bound to have an operator, and an operator might maintain several dedicated clouds depending on its use case. Similarly, Systems are bound to be grouped together logically even within their Local Clouds, e.g. if they are located in the same production line. Figure 7 depicts how an Arrowhead System, a Service and a Cloud is represented in the unified data structures. These can be used by all communications within the Core Systems.

As an example, let us consider temperature and humidity sensors in a refrigerator warehouse. They measure the indoor temperature at different locations and heights. Each sector has two sensors that measure the floor- and ceiling-level tem-

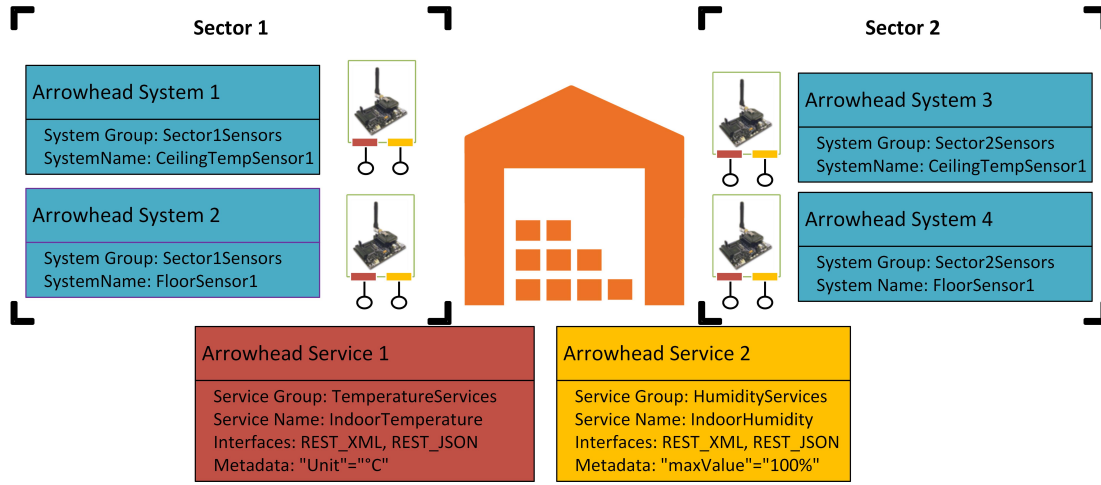


Fig. 8. Example for unique identifiers regarding the various Arrowhead elements

peratures. Figure 8 shows how these sensors are represented as Arrowhead Systems and how their measurement readout resources can be abstracted as Arrowhead Services.

B. Arrowhead Core Systems

1) *Service Registry*: The Service Registry uses DNS-SD for backwards compatibility, but utilizes a REST facade to communicate with the rest of the framework. It keeps track of the currently online Service Providers and what Services they have to offer (up to the Service Interface level). Since graceful shutdown is not a viable option in some use cases, this System runs a database clean-up daemon by periodically checking whether the registered Service Providers are still active or not.

Service metadata is also associated with *each and every* service offering instance *separately*. This provides the possibility to describe the offered Service instance with key-value pairs (e.g. if a sensor provides its readout in °C or °F). Based on these, advanced querying capabilities can and have been also realized: other Core Systems can fetch lists of suitable Service Providers, based on the criteria towards the Service that they are looking for (e.g. if we are explicitly seeking indoor temperature services measured in °C).

2) *Authorization and security considerations*: Typically, an AA system has various tasks to fulfil. In this case, the Authorization System holds access rights (service consumption rights) for each System in the Cloud. This authorization database describes where each System can consume Services from and what type of Services they are allowed to access there.

It also manages admission control and verification, based on certificates and the authorization database: a Service Provider has to be able to verify the authorization status of an inbound service request from another System. This is achieved by the introduction of an authorization token. This token is created in the Authorization System from the certificate information, and passed on to the Service Consumer in the orchestration process. This token can verify that the given Service Consumer

has been orchestrated properly, and therefore it is authorized by the Core Systems to invoke the Service from the given Service Provider.

3) *Resource Management*: An other invaluable aspect of such an automation support framework is the centralized management of resources. Resources in a Local Cloud can vary from e.g. processor task time slots of embedded Systems through real-time networking QoS up to setting client-server servicing requirements (e.g. transactions per second). This resource management is tasked with the QoS Manager [16]. Its two major processes are (i) the verification of the requirements set by new Service Requests at the early stage of the orchestration process, and (ii) the actual resource reservation process with configuring the affected components as the last step of the orchestration process. Therefore, this optional Core System is in tight cooperation with the other Core Systems, and it also participates in the orchestration process.

4) *Orchestration Store*: The Orchestration Store is a statically configured database that contains hardwired orchestration data for Systems in the Cloud. Operators can set up relations between Systems that cannot change runtime, and has to be respected in the orchestration process. One Orchestration Store entry consists of a Service Provider and a Service specification that must be enforced if the Service Consumer at question requests orchestration. These entries are set by the operators and developers of a Local Cloud suiting the physical processes and restrictions in the the Cloud.

5) *Orchestrator*: In Generation 3.2, the Orchestrator bears the biggest responsibility in a Local Cloud. It is the primary decision-maker that is aware of the current conditions in the SoS. Its primary task is to allocate Service Providers to the Service Requests sent in by Systems. During this *orchestration process* the Orchestrator consults with the other Core Systems and makes a decision based on the responses, as Figure 9 suggests. This orchestration process consists of the following:

- 1) Checking whether there is a hardwired orchestration rule for the requester System in the Orchestration Store;

- 2) Fetching a list of suitable on-line Service Providers from the Service Registry that offer the Service at hand;
- 3) Extending the Service discovery to other Clouds using the Gatekeeper if necessary;
- 4) Filtering this list of possible Service Providers based on the authorization status of the Service Consumer;
- 5) Further filtering this list to find out which Service Provider(s) can satisfy the QoS expectations;
- 6) Choosing one Service Provider from this filtered list (final matchmaking);
- 7) Then making the QoS resource reservations for this optimal (Provider - Service - Consumer) combination;
- 8) Responding to the requester System with an Orchestration Form.

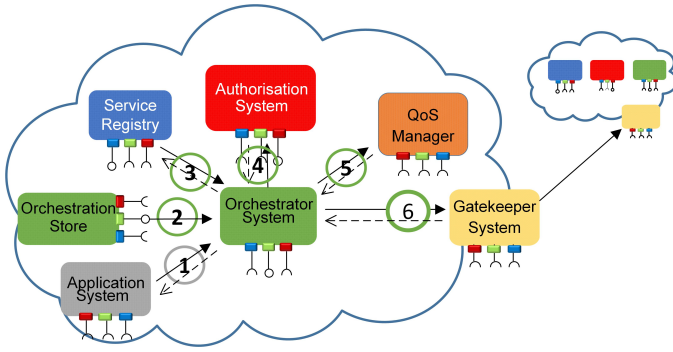


Fig. 9. The orchestration process

If inter-Cloud servicing is set up and enabled, then the Orchestrator can also look for adequate Service Providers outside of its Local Cloud. In this process the GSD and ICN Services of the Gatekeeper System are involved as described in [7]. The whole intra-Cloud orchestration process can also be skipped or triggered if there is explicit instruction for it either in the Service Request or in the Orchestration Store entry for the requester System.

Moreover, the orchestration process can also be affected directly by the requester System via setting the appropriate flags in the Service Request – as described in the next section, III-C. It is also worth noting, that the internal workings of the orchestration process can also be tailored to the deployment requirements. This way conditional and dynamical orchestration can be achieved by implementing use-case specific orchestration algorithms.

C. Interfaces for Application Systems

In general, Application Systems use the mandatory Core Systems to aquire the proper information required to consume Service from other Application Systems. These Core Services should be basic request-responses, and Systems should only have limited information about macro-lensed objectives of an SoS in order to keep them simple. The Core Systems themselves then bear all responsibilities and tasks in order to run the SoS smoothly. Since in Generation 3.2 Application Systems mainly communicate with the Orchestrator, this section will describe the Orchestration interface. It is intended to be a

simple request-response on the client-side; although a complex orchestration process runs in the background. Application Systems submit Service Requests and receive an Orchestration Response using the appropriate URL, payload and HTTP method in the Web Services implementation.

Service Request Form	OrchestrationFlags
RequesterSystem: ArrowheadSystem	ExternalServiceRequest: Boolean
RequestedService: ArrowheadService	TriggerInterCloud: Boolean
RequestedQoS: String	MetadataSearch: Boolean
OrchestrationFlags: HashMap <String, Boolean>	PingProvider: Boolean
PreferredProviders: List <ArrowheadSystem>	OverrideStore: Boolean
	Matchmaking: Boolean
	HasPreferences: Boolean

Fig. 10. The sturcture of the Service Request Form

The Service Request (Figure 10) describes what and how it is exactly requested: the color coding suggests the fields affecting various Core Systems in the orchestration process. This query information is either composed run-time on autonomous systems, or composed statically by the cloud operator and stored partially in the Orchestration Store (that is part of the Orchestrator System). Autonomous Systems can take advantage of this dynamical configurability of the orchestration process and can also affect it via setting simple flags.

This Service also utilizes the common descriptor objects described in III-A. The payload mainly describes which System is requesting orchestration (this information is also present in the certificate when using SSL) and what Service is requested. The remaining fields control the orchestration process, e.g. Systems can specify their servicing QoS expectations or can force certain preferred Service Providers they need to connect to. This structure also enables further functionalities to customize the sub-processes of orchestration, for example:

- Customizing the assembly of the list containing the Service Providers (e.g. switching between intra- and inter-Cloud servicing or fetching from the Orchestration Store),
- Customizing the matchmaking algorithm (e.g. weightings of certain Service Provider parameters),
- Enabling additional Services (e.g. logging or Service content translation),
- Providing contextual data (e.g. whether this Service Request is a retry attempt of error handling).

The Orchestration Form contains a simple response: basically gives the Consumer instructions on where to connect to. This is described by identifying the Service Provider (an Arrowhead System descriptor with the address and port), the Service instance (with an Arrowhead Service descriptor up to the interface level) and how the Service can be accessed within that System (e.g. its URL path for the WS implementation). However, there can be other optional parts of this message, e.g. passing an authorization token to the Consumer that will be validated by the Service Provider for admission control purposes.

IV. FURTHER WORK

This concept-paper has provided a general architecture for integrating optional modules that can further enhance a Local Cloud's capability, flexibility and security. The presented proof-of-concept generation of the Arrowhead Framework is not yet fully implemented and widely verified. On the other hand, there are already implemented System-of-Systems available [17] that show practical verification of the concepts presented here.

Nevertheless, there are still some expectations detailed in the introduction that require not just implementational, but further architectural considerations.

Firstly, advanced security capabilities will be realized by a unified certificate-based token system that will also enable secure inter-Cloud servicing as well, based on [19]. This will be further aided by the Network Manager in order to handle network visibility problems. Exploring the neighborhoods, building trust and authenticity between autonomously deployed Local Clouds are also of interest.

Secondly, the orchestration process has also potential to integrate translational capabilities (so that Systems implementing the same Service with different interfaces can communicate), and to utilize the Plant Description Service (a supporting Core System that maps out e.g. the physical and process logical connections between entities) [20].

Moreover, the shortcoming of error and event handling [21] is yet to be covered: detecting, identifying and solving events occurring in the Local Cloud. The roles and responsibilities (with the Event Handler System [22]) should be perfected.

V. SUMMARY

This paper presented the Arrowhead Framework briefly, then showed how some of its concepts are further enhanced.

Arrowhead is fulfilling the three principles of Service Oriented Architectures, namely (i) loose-coupling, (ii) late binding, and (iii) „lookup” (or in other words: service discovery).

In order to allow high flexibility as well as interoperability, and support some native security-related features, Arrowhead introduced the Local Cloud concept. This means that within some operational or geographical boundaries, there are core services – Orchestration, Authorization, Service Registry, and others – are provided within a Local Cloud.

The requirements for enhancing the framework have ranged from dynamical service orchestration and QoS-based resource allocation, through centralized event handling, to intra-cloud protocol translation and inter-Cloud servicing. These issues are tackled in four ways, namely

- introducing a strict separation in systems and services and their hierarchies;
- introducing new, non-mandatory core systems (such as the Gatekeeper or the QoS Manager);
- extending some of the internal functions of the mandatory core systems, especially for Orchestration; and
- introducing new interfaces for applications such as the Service Request Form and the Orchestration Form.

ACKNOWLEDGMENT

This work is supported by the EU ARTEMIS JU funding, within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD). The authors would also like to acknowledge the efforts of the Hungarian team who participated in this work: from evopro Inc. and from the Budapest University of Technology and Economics.

REFERENCES

- [1] Artemis, <http://www.arrowhead.eu>.
- [2] J. Delsing, ed. (2016) Arrowhead framework wiki. [Online]. Available: <http://forge.soa4d.org/arrowhead-f/wiki>
- [3] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, “A survey of commercial framework for the Internet of Things,” in *IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)*, 2015.
- [4] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.
- [5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services*. Springer, 2004.
- [6] P. Varga, F. Blomstedt, L. Lino Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. Martinez de Soria, “Making system of systems interoperable - the core components of the arrowhead technology framework,” *Journal of Network and Computer Applications*, 2016.
- [7] P. Varga and C. Hegedus, “Service interaction through gateways for inter-cloud collaboration within the arrowhead framework,” in *5th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace and Electronic Systems (Wireless VITAE)*, 2015.
- [8] L. Richardson and S. Ruby, *RESTful web services*. ” O'Reilly Media, Inc.”, 2008.
- [9] A. Rodriguez, “Restful web services: The basics,” *IBM developerWorks*, 2008.
- [10] S. Cheshire and M. Krochmal, *DNS-Based Service Discovery*, IETF Standard RFC-6763, 2013.
- [11] F. Blomstedt, L. Lino Ferreira, M. Klisics, C. Chrysoulas, I. Martinez de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for soa application development and documentation,” in *IEEE IECON*, 2014.
- [12] C. Jennings and Z. Shelby, “Media types for sensor markup language (senml) draft-jennings-senml-10,” IETF, Tech. Rep., 2013.
- [13] I. Allcance, “Ipso smart objects,” <http://www.ipso-alliance.org/wp-content/uploads/2016/01/ipso-paper.pdf>, 2016.
- [14] “The mimosa project site,” <http://www.mimosa.org/>, 2016.
- [15] P. Malo, T. Teixeira, B. Almeida, and M. Mateus, “Interoperability Repository System for the Internet-of-Things,” in *Green Computing and Communications (GreenCom), 2013 IEEE iThings/CPSCoM*, 2013.
- [16] L. L. Ferreira, M. Albano, and J. Delsing, “QoS-as-a-Service in the Local Cloud,” in *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2016)*, 2016.
- [17] B. Peceli, G. Singler, Z. Theisz, C. Hegedus, P. Varga, and Z. Szepessy, “Integrating an Electric Vehicle Supply Equipment with the Arrowhead Framework,” in *IEEE IECON*, 2016.
- [18] P. Punal, J. Eliasson, and J. Delsing, “An authentication and access control framework for coap-based internet of things,” in *40th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2014.
- [19] S. Plosz, C. Hegedus, and P. Varga, “Advanced Security Considerations in the Arrowhead Framework,” in *ERCIM/EWICS/ARTEMIS Dependable Embedded and Cyber-physical Systems and Systems-of-Systems Workshop (DECSoS)*, September 2016, pp. 1–13.
- [20] O. Carlsson, D. Vera, J. Delsing, and B. Ahmad, “Plant descriptions for engineering tool interoperability,” in *14th International Conference on Industrial Informatics*, 2016.
- [21] H. Derhamy, P. Varga, J. Eliasson, J. Delsing, and P. Punal, “Translation error handling for multi-protocol soa systems,” in *20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2015.
- [22] M. Albano, L. Lino Ferreira, and J. Sousa, “Event handler system: Publish/subscribe communication for the arrowhead world,” in *12th IEEE World Conference on Factory Communication Systems (WFCS)*, 2016.