Making System of Systems Interoperable - the Core Components of the Arrowhead Framework

Pal Varga[a], Fredrik Blomstedt[b], Luis Lino Ferreira[c], Jens Eliasson[d], Mats Johansson[d], Jerker Delsing[d], Iker Martinez de Soria[e]

[a]*Budapest University of Technology and Economics, Budapest, Hungary*
[b]*BNearIT Inc., Lulea, Sweden*
[c]*ISEP, Polytechnic of Porto - School of Engineering, Porto, Portugal*
[d]*Lulea University of Technology, Lulea, Sweden*
[e]*Tecnalia Research and Innovation, Bilbao, Spain*

Abstract

The objective of the Arrowhead Framework is to efficiently support the development, deployment and operation of interconnected, cooperative systems. It is based on the Service Oriented Architecture philosophy. The building elements of the framework are *systems* that provide and consume *services*, and cooperate as *systems of systems*. Some commonly used systems, such as orchestration, authorization or service registry are considered as core. These can be used by any system of systems that follow the guidelines of the Arrowhead Framework.

Within the framework, systems – using different information exchange technologies during collaboration – are helped through various approaches. These include the so called Interoperability Layer, as well as systems and services for translation. Furthermore, one of the main problems of developing such highly interoperable systems is the lack of understanding between various development groups. Adequate development and service documentation methodologies can help to overcome this issue.

The design, development and verification methodology for each service, system and system of systems within the Arrowhead Framework supports that these can be implemented, verified, deployed, and run in an interoperable way. This paper presents an overview of the framework together with its core elements – and provides guidelines for the design and deployment of interoperable, Arrowhead-compliant cooperative systems.

Preprint submitted to Journal of Network and Computer Applications August 28, 2016

Keywords: System Interoperability, System of Systems, Systems, Services, Service Oriented Architectures, Internet of Things

1. Introduction

The integration of large collaborative automation systems requires that the participating systems can exchange information during run-time. The orchestration of such information exchange can either be made at engineering time or engineered during run-time. Todays legacy systems are most often proprietary solutions, such as ABB 800xa [1] and Siemens' Simatic [2].

For scenarios of very large system of systems – whole cities – the present solutions do not scale favorably. Thus SOA (Service Oriented Architecture) approaches have been investigated to address large scale automation systems [3, 4, 5, 6].

In the SOA approach, the term Service is used as an entity for information exchange between a Service provider System and a Service consumer System. Furthermore, we can introduce the System of Systems, which are a set of systems working together to achieve a more complex target or a higher purpose.

There are two major strategies in building SOA automation systems. The first one is connecting all involved devices to a global cloud using one SOA protocol. In this case, the cloud provides communication and processing infrastructure, as Figure 1 suggests.

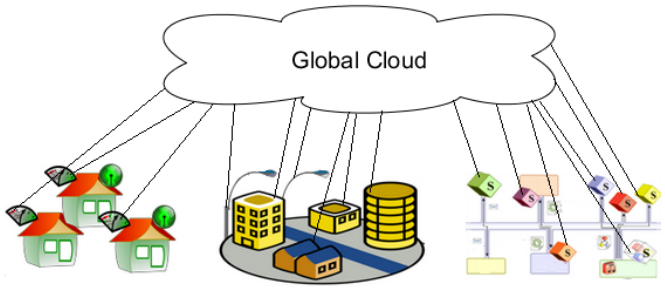

Figure 1: Collaboration through a global cloud

The second one is building local SOA clouds, where the local cloud is governed through the use of some core services. These are provided by one

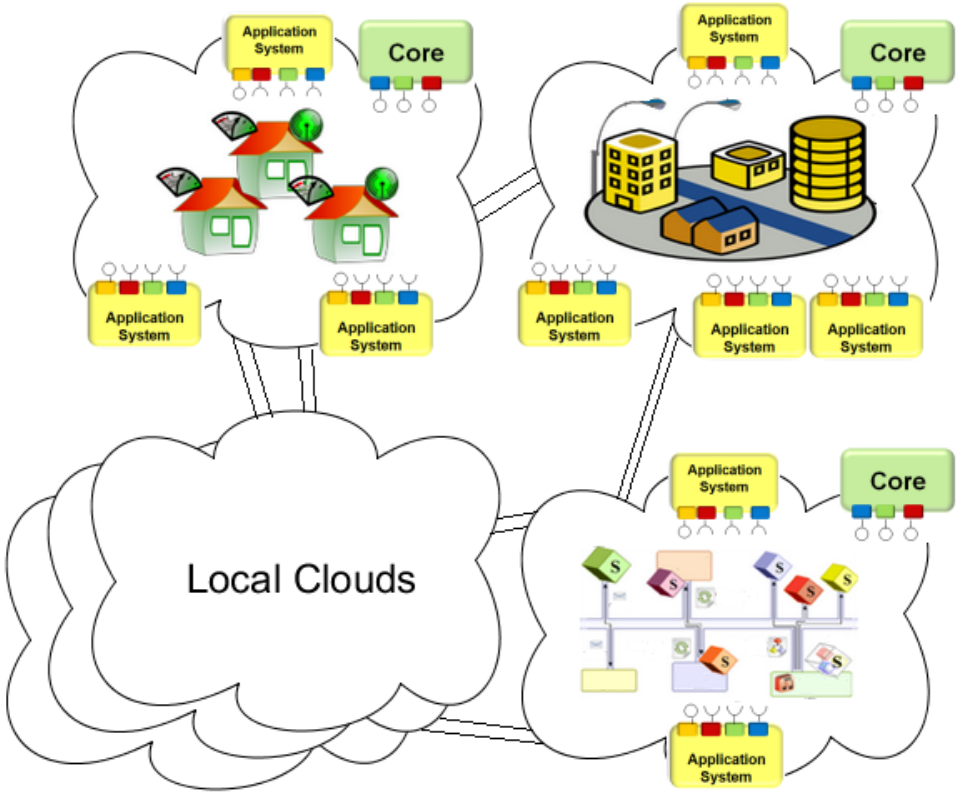

Figure 2: Interconnected local collaborative clouds

of the locally involved systems – the manner of interconnection is depicted by Figure 2.

In this case, the local cloud can provide further support functions through its core systems. Another way to differentiate these approaches is based on input drivers. The two major types here are the data driven and the event driven architectures.

The data driven global cloud already has a number of large commercial implementations, e.g. Facebook, Amazon and Google. The event driven local cloud approach is less investigated – although it has a number of advantages – and is the focus of this paper. Nevertheless, these are briefly discussed within the related work, Section 6.

The main attention is around the selection of core systems: which core

services are necessary in an event driven local cloud approach. As part of the Arrowhead project [7], this issue has been discussed based on requirements from 5 industrial domains: production, smart buildings and infrastructure, electro mobility, energy production and end user services, virtual market of energy. This lead to a number of core services, which are presented briefly in this paper, together with a discussion on how these core services will support in meeting industrial domain requirements. These core services take an essential part in providing a framework enabling interoperability in-between almost any service provided by heterogeneous systems.

The enabling of local SOA clouds requires us to consider a couple of fundamental questions, as follows.

- How does a system that is a service provider make its services known to service consumers?
- How does a system that is a service consumer discovers services it wants to consume?
- How does a system that is a service provider decides if a system that wants to consume its services, is authorized to do that?
- How to orchestrate system of systems, i.e. enabling an orchestration body to control which of the provided service instances a system shall consume?

These questions are here addressed by what we call the core services necessary to meet the requirements and enable a collaborative automation cloud.

In the following we will denote this cloud as the Arrowhead Framework, which also servers as a target architecture for SOA-based systems that ensures interoperability and integrability for the Internet of Things.

Arrowhead Framework is designed as a SOA Reference Architecture (RA). The current solution of this RA should be seen as a proof of concept for the Arrowhead Framework (and its partners applications). The world of IoT should have a SOA RA to ensure interoperability and integrability for common IoT target architectures. One of the goals of the Arrowhead Framework is to fill this gap, and pave the path for such SOA target architectures.

2. The Arrowhead Framework

2.1. Necessities for a loosely coupled event based automation cloud

There are a few important requirements that a local cloud automation functionality has to fulfill. These include the following:

- discovery of services;
- enabling loosely coupled data exchange between producer and consumer systems;
- authorization of service usage;
- orchestration of automation functionalities.

The Arrowhead Framework addresses these questions through the definition of Information Assurance services (IA), Information Infrastructure services (II) and System Management services (SM).

To create a minimal working SOA system based on the Arrowhead Framework, one mandatory core service is necessary from each of the three groups IA, II and SM. These are Service Discovery, Authorization and Orchestration services. Using these set of services, it is possible to design and implement a minimal local automation cloud. Each of the later mentioned services belong to either the IA, II, or SM group, contributing to their general purpose.

To further enhance the capability of the local automation cloud, the following services have been also regarded as important: Deployment service, User-System Registry service, Configuration service, Event Handler service, and Meta-Service Registry service.

In the next sections the core systems providing these services are described in detail. Furthermore, the way they support the operation of a single system and system of systems is also described.

2.2. Purpose and Benefits for System Developers

The Arrowhead Framework contains common solutions for the core functionality in the area of Information Infrastructure, Systems Management and Information Assurance as well as the specification for the application services carrying information vital for the process being automated.

A developer needs to know how to develop, deploy, maintain and manage Arrowhead-compliant systems. Therefore, it is crucial that there are common understandings of how services, systems and system of systems are

defined and described. To address these issues, the framework also includes design patterns, documentation templates and guidelines that aim at helping systems, newly developed or legacy, to conform to Arrowhead Framework specifications.

The Arrowhead Framework aims to use existing protocols, standards and handle legacy systems and System of Systems compositions as well as future systems and future compositions to fulfill new needs. However, in order to ensure interoperability and integrability of all these different systems, we need to have a common approach, common design patterns and common parts of core functionalities.

The Arrowhead Framework consists of what is needed for anyone to design, implement and deploy an Arrowhead-compliant system aiming at enabling all of its users to work in a common and unified approach – leading towards high levels of interoperability.

2.3. Services

In the context of the Arrowhead Framework a Service is what is used to exchange information from a providing System to a consuming System. In a Service, capabilities are grouped together if they share the same context [8]. As an example, a service in a REST [9] architecture can be constituted by all interfaces which are related to the same REST resource. In this case operations over different resources can be grouped as different services. A resource is a nomenclature used on a REST-based architecture to designate an information resource. In the context of Arrowhead, a resource might be a temperature sensor or the power consumption reading from a power meter.

Similar approaches apply to different technologies being used in Arrowhead, such as COAP [10], XMPP [11] or OPC-UA [12].

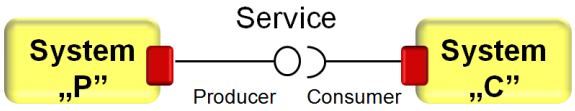

Figure 3: Services produced and consumed by Systems

The Arrowhead Framework also implies the usage of a number of service orientation principles derived from high level objectives and properties. Where a Service can be realized by an arbitrary number of service producers and service consumers, therefore insuring its reusability.

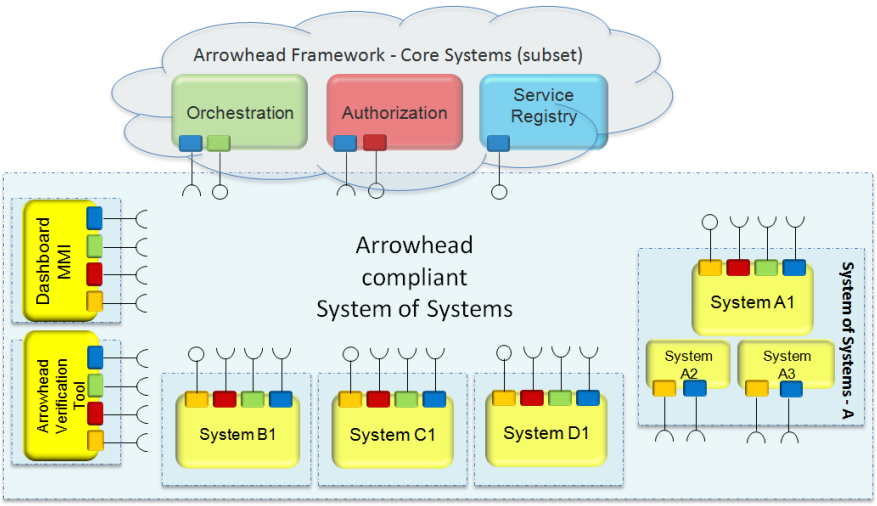

Figure 4: How the Arrowhead core components support the System of Systems

Arrowhead-compliant systems should also be capable of consuming different types of services, process and provide other services that fulfil a more complex task – the composability of services.

An Arrowhead Service might also be capable of supporting different kinds of non-functional requirements like security, real-time operation or different levels of reliability.

In relation to its development and maintenance a Service has its own group of appointed stakeholders, with an interest on this service, being one stakeholder, responsible for the governance of the Service (definition, development, deployment and maintenance).

2.4. Systems

A System is what is providing and/or consuming services (see Figure 3). 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. It normally includes software executing on hardware. It may also be referred to as Component or Device.

A system can be user interface display, used to control the air-conditioning within a house, but it can also be a small temperature sensor that complies with the Arrowhead Framework.

2.5. System of Systems

Any given system of system can be defined by five main characteristics in relation to other very large and complex but monolithic systems [13], [14], namely (i) operational independence of its systems, (ii) management independence of the systems, (iii) evolutionary development, (iv) emergent behaviour, and (v) geographic distribution.

When Arrowhead-compliant systems collaborate, they become a system of systems. Since these systems of systems can also collaborate, the Arrowhead Framework becomes a natural enabler of further, complex solutions. Figure 4 depicts such an example.

Systems A1, A2 and A3 are grouped together to create the *System of Systems A*. This SoS can be bundled together with other systems (such as B1, C1 and D1, also depicted in Figure 4), since they are Arrowhead-compliant as well. The Arrowhead Framework provides core systems, such as *Orchestration*, *Authorization* and *Service Registry*, which help on establishing the most adequate and secure connection between the players. All the application systems are consumers of the services of these core systems, which – in this example – reside in a local cloud. The *Dashboard MMI* (Man-Machine Interface) allows interaction with this system of system; whereas the *Arrowhead Verification Tool* provides means for testing Arrowhead compliance.

2.6. Documentation Structure

It is a common experience of both system developers and integrators that insufficient, and not properly structured documentation makes it hard – if not impossible – to properly understand how to build collaboration with a given system.

A system integrator needs to know how to develop, deploy, maintain and manage Arrowhead-compliant systems. Therefore, it is crucial that there are common understandings of how Services, Systems and System of Systems (SoS) are defined and described. To address these issues, the Arrowhead Documentation Framework allows documenting SOA artefacts, in a common format [13].

To this purpose the Arrowhead consortium [7] defined a three-level documentation structure: System of Systems, System and Service level. These are depicted in Figure 5, which also shows the links between documents.

The System of System level is described by two types of documents. The SoS Description, which shows an abstract view of a SoS and the SoS De-

sign Description which shows an implementation view of the SoS with its technologies and deployment views.

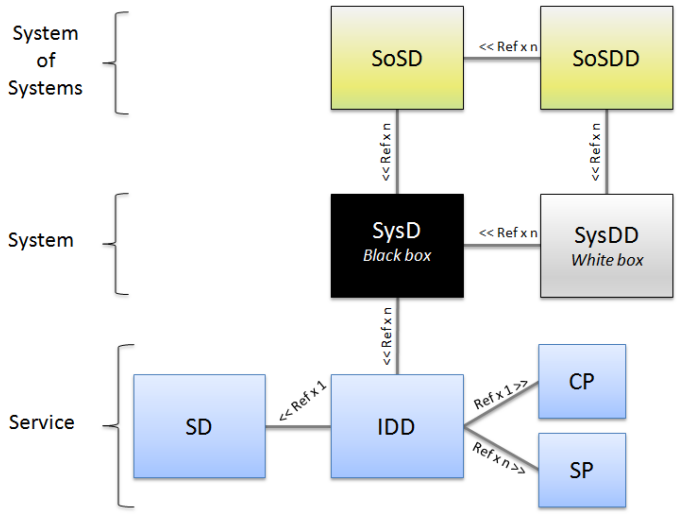

Figure 5: The Arrowhead Framework documentation relationships

The SoSD describes the main functionalities and the generic architecture of the SoS. It will mainly be used to describe one System of Systems in an abstract way, without instantiating into any specific technologies. Examples of its usage are on the description of generic SOA-based installations, like building automation systems or a factory automation system. The document should present its main building blocks as independent systems (with pointers to its specific documents). Also, diagrams representing the system behaviour, like use-case diagrams and behaviour diagrams (e.g. using UML, BPMN or SysML) must be included. This document also includes information about the non-functional requirements, like required levels of QoS and security.

The SoSDD document describes how a SoSD has been implemented on a specific scenario, showing the technologies used and its setup. Therefore, it points out all necessary Black Box SysD and White Box SysDD documents, describing the systems used in this realization.

The System Level consists of a black box design document, named SysD Black Box Design and a white box design document, named SysDD White Box Design.

The SysD describes the System as a Black Box, documenting its provided and required interfaces, which are defined in the IDD document, and

the corresponding technical solutions, without describing its internal implementation. All the produced/consumed services (with references to the IDD document) are listed. In this way a clear picture of how to interface the system is provided.

The SysDD extends the Black Box description showing its internal details. This document is optional, since it might expose the knowledge of the company which implemented the system.

The Service Level consists of four documents: the Service Description (SD), the Interface Design Description (IDD), the Communication Profile (CP), and the Semantic Profile (SP).

The Service Description is a technology independent and abstract view of a Service. The document starts by describing the main objectives and functionalities of the service and follows on defining its Abstract Interfaces, an Abstract Information Model and Sequence Diagrams, using UML or SysML.

The IDD states the actual solution, the format and protocols for achieving successful information exchange by using a specific technology. An IDD provides a detailed description of how a service is implemented by using a specific Communication Profile and technologies.

The CP contains all the information regarding the transfer protocol, the security mechanism and the data format to be used.

The SP defines all the information needed to describe the data format by pointing out what is the type of the encoding (e.g., JSON, XML, EXI compressed XML).

3. Interoperability of Systems within the Arrowhead Framework

In an Arrowhead Framework compliant system of systems, there are a set of core systems, which are marked mandatory. The core systems provide the earlier mentioned core services that cover the interoperability requirements of SOA clouds: application service registration, service discovery, authorization, and orchestration.

The core systems shall be seen as infrastructural support services – with system management and information assurance aspects.

Each application system can focus on the business value and fulfill the assigned task. The application system will not need to focus on supporting logic regarding configuration, lookup/discovery, or security. These core functions are distributed to the core systems and accessed by implement their provided services.

It is of course possible for anybody to implement their own core systems, but in order to keep the interoperability, it is crucial that they conform exactly to the same black box design as the ones approved within the framework.

The general goal of core systems is to enable and support the application systems to achieve successful information exchange.

This section introduces the core system elements, and provides an overview on system of system collaboration, showing how the core systems and the application systems work together. The purpose for this collaboration composition between core and application systems is to enable the two application systems A and B to exchange information.

Before going into these details of Arrowhead core systems (and how they support information exchange and interoperability of the application systems), the following subsections introduce the Framework's solutions for interoperability between different technologies.

3.1. Interoperability of systems communicating with different protocols and technologies

The maturity level indicates how compliant is a system with the Arrowhead Framework. This level indicates the integration status of the above mentioned core functionalities, as well. Figure 6 depicts the three maturity levels – from connecting to other, Arrowhead-compliant systems through adapters – to natively handling the connections towards core functionalities and other systems within an Arrowhead cloud.

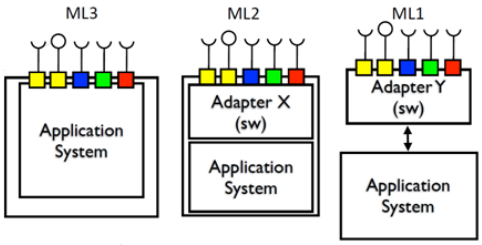

Figure 6: The three levels of maturity within the Arrowhead Framework

In order to make services transparent for systems based on different protocols and information exchange technologies, there are three interoperability approaches suggested within the Arrowhead Framework: (i) the Interoperability Layer, (ii) the Translator System, and (iii) the Translation as a Service.

3.1.1. The Interoperability Layer approach

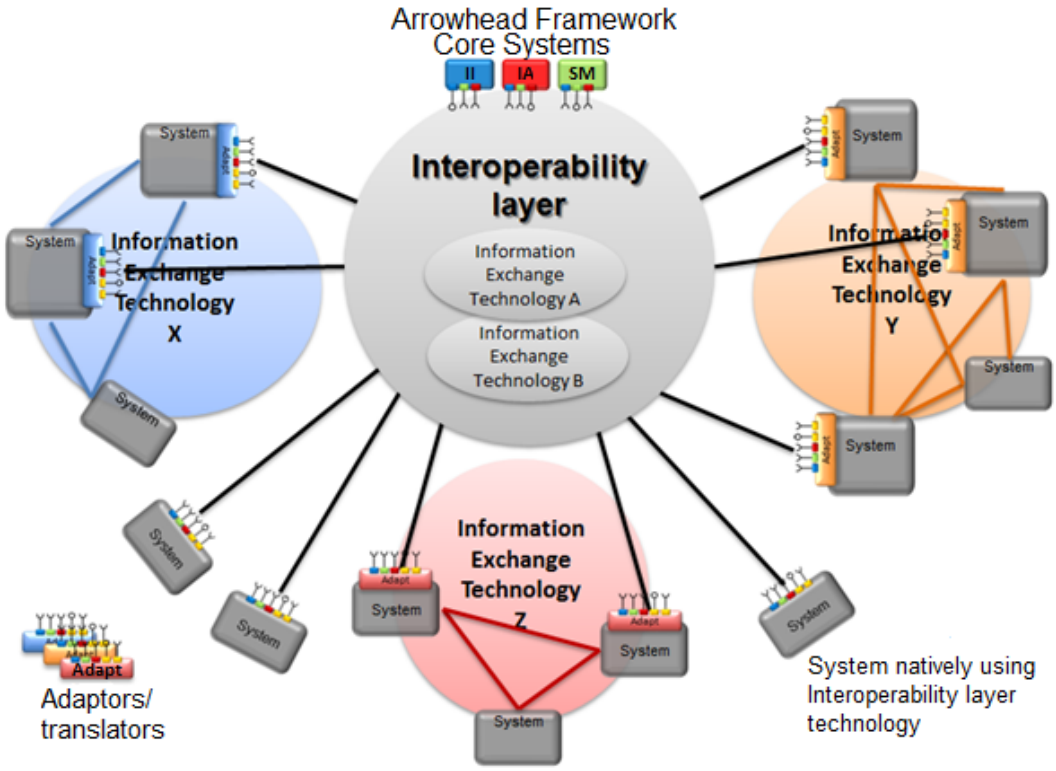

Figure 7: Systems with different protocols, technologies and semantics can communicate through the Interoperability Layer

The layer defines a suitable technology and semantics for each service and all systems translate to/from that technology and semantics. From the design documentation point of view, it is defined in one Interface Design Description (IDD) which is then used for each Service Description (SD) that wishes to utilize it. Depending on the maturity level, translators or adapters can be part of each system – while from the network perspective, all systems appear to use the technology(ies)/semantics provided by the interoperability layer. As most service characteristics within Arrowhead fit to the REST/HTTP/TLS/XML Communication Profile (CP), it is recommended for the interoperability layer of a generic System of System. A generic overview of the Interoperability Layer approach is depicted by Figure 7.

3.1.2. The Translator System approach

This approach is based on a translator system that can consume services with one technology (CP-1) and provide the same service on another technology (CP-2). In this case, the orchestration connects the translator consumer to the original provider and the end consumer system to the translator provider. Figure 8 provides and example of the Translator System approach.

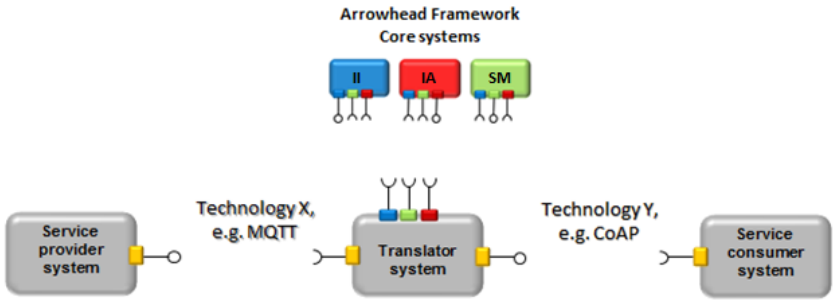

Figure 8: An example of how the Translator System fits to the framework

3.1.3. The Translation as a Service approach

This approach is similar to the translator system but adding a separate "GetTranslation service" that can be consumed by systems looking for translation. This service can be provided as a core framework element as well. Figure 9 provides and example of the approach for Translation as a Service.

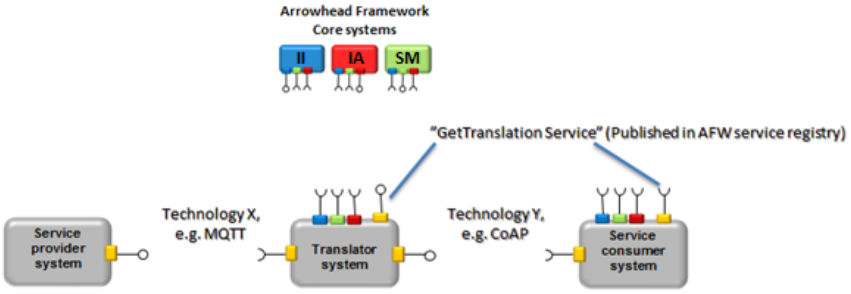

Figure 9: Translation as a Service can be provided as a core functionality of the Arrowhead Framework

Issues and solutions regarding translation in a multi-protocol environment are further detailed in [15], which also takes specific examples from Arrowhead Framework application scenarios.

3.2. The Core System Elements

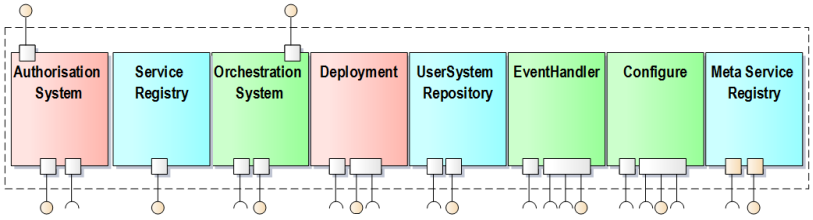

Figure 10: Core System elements of the Arrowhead Framework at Generation 2

The core systems provided within the Arrowhead Framework are depicted in Figure 10. These cover essential functionalities, which are made available for all kinds of system of systems.

Utilizing all of these core system elements are not mandatory; although these runtime systems are made available so the design, development, integration, deployment of a system of system becomes more effective with Arrowhead-compliance than without it. These core system elements already provide flexible and well-tested services that are necessary in many systems of systems – hence integrating these requires less effort from system developers than creating proprietary solutions to cover the same functionalities.

This section outlines some of the core systems that are made available within the Arrowhead Framework.

3.2.1. Service Registry System (mandatory)

The Service Registry System keeps track of all active producing services within the network. It is used to ensure that all systems can find each other – even if endpoints are dynamically changed. It supports a service registry functionality based on DNS and DNS-SD; since the Arrowhead Framework is a domain-based infrastructure. All Systems within the network that have services producing information to the network shall publish its producing service within the Service Registry by using the Service Discovery service.

Within a system of systems, the Service Registry further supports system interoperability through its capability of searching for specific service producer features, i.e. an application service producer with a specific type of output.

In short, it enables systems to publish their own application services and lookup others'.

3.2.2. Authorization System (mandatory)

The Authorization system controls that a service can only be accessed by an authorized consumer. It consists of two service producers and one service consumer and it maintains a list of access rules to system resources (i.e. services). The *Authorization Management* service provides the possibility to manage the access rules for specific resources. The *Authorization Control* service provides the possibility of managing the access for an external service to a specific resource. The system uses the Service Discovery service to publish all its producing services within the Service Registry system.

3.2.3. Orchestration System (mandatory)

The Orchestration system is a central component of the Arrowhead Framework and also in any SOA-based architecture [8]. Orchestration is used to control how systems are deployed and in what way should they be interconnected. Orchestration in the context of SOA can be viewed as the system that supports the establishment of all other systems through providing coordination, control and deployment services.

In industrial applications the use of SOA for massive distributed system of systems requires Orchestration. It is utilized to dynamically allow the re-use of existing services and systems in order to create new services and functionalities [16].

The application systems' services are initially seen as passive and being on standby. They are not connected at deployment or even during start-up of the system of systems. Their services can be managed to connect, or be connected to others – in order to fulfill a specific need.

SOA-based application systems support late binding. The application system do not know about their usage until they are orchestrated and configured to connect and interact with other system in a system-of-system collaboration network.

This core function enables long term life cycle components that can be used and reused in different future configurations for future business needs.

Figure 11 shows an example of how Orchestration works when utilizing multiple systems in order to create customized functionality. The Arrowhead Framework currently supports REST-based Orchestration of services using for example REST or CoAP. The Orchestration system is heavily depending on the Authorization, Service repository, and Configuration systems.

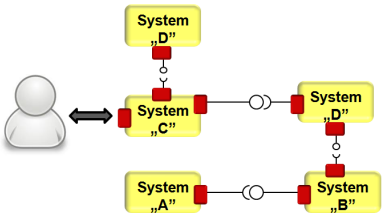

Figure 11: Service composition

3.2.4. Deployment System

The primary purpose of this system is to handle deployment of a new system and ensure authentication in a common Arrowhead network. The deployment system handles the vendor authentication, identification and assigns a specific certificate that is trusted in an Arrowhead network. This way the Deployment system can be seen as support for joining pre-assigned devices, and hence can save installation and engineering time.

3.2.5. User-System Registry

The User-System Registry system holds unique system identities for deployed systems within the Arrowhead network – it stores User- and System-specific information. A System information entry contains vendor ID, delivery time, point of contact, etc.

The Orchestration system utilizes the User-System Registry in combination with the Service Registry. This combination allows the Orchestration system to provide a strong picture of the possibilities for system of system compositions that enable system collaboration.

3.2.6. Configuration System

The Configuration system stores specific configuration values for a system. The configuration values stored at the Configuration Store are fetched through the Configuration system and applied by the Application system.

3.2.7. Event Handler System

The Event Handler System provides functionality for the handling of events that occur in an Arrowhead network, as Figure 12 depicts. The Event Handler receives events from Event Producers and dispatches them to registered Event Consumers. Where, the Event Handler is the component that logs events to persistent storage, registers producers and consumers of

events and applies filtering rules (configured by Event Consumers) to event distribution. An Event Consumer is a component that consumes events.

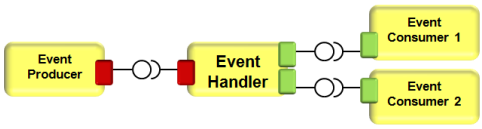

Figure 12: Connections between the Event Handler, Event Consumers and Producers

The Event Handler system has the intelligence of applying filtering rules to incoming events. These rules can be complex, e.g. based on message content or simply based on the severity level of the message, which varies from Debugging to Critical.

The Event Handler can also be connected to an internal or external database System which is responsible for the permanent storage of all events.

3.2.8. Meta-Service Registry System

The Meta-Service Registry system stores additional information about a service for offline access. This is a functionality for the service registry, since it stores additional data such as constraint information, up-time, or other valuable data during utilizing a service.

4. Use Cases

4.1. Introducing an Application System within the Arrowhead Framework

The deployment procedure is the most critical one to deploy a generic system within the Arrowhead network.

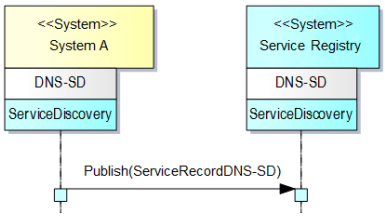

Figure 13: Application systems must publish information about their services available

At start-up the System A connects to the Deployment system and initiates a connection towards the Authenticate service to notify that System A

is available with known vendor certificate. If the Deployment allows System A to enter the network, the Deployment system establishes connection with the User-System registry by signing the User-SystemDiscovery service and requests a unique system identity for System A. Note that the Deployment system is not a mandatory core system, because automatic deployment of systems may not handle *all* aspects such as security, certificate revocation, and system identification. Hence the Deployment system's functionality should be covered by other means within those systems of systems not utilizing it.

The identification result is returned to the System A to indicate whether the deployment procedure was successful. The certification procedure result is used later for securing information exchange and for system identification.

When successfully deployed, System A publishes information about its available services to the Service Registry (see Figure 13), hence making these services available to be discovered by other systems.

4.2. An Application System is Connecting to the Arrowhead Framework for Consuming Services

The sequence depicted by Figure 14 describes the procedure of Application System B starting up and making a connection with System A (which is assumed to be already running) by using the core systems.

1. System B starts up and searches (LookUp) in the Service Registry for the Orchestration system services – to be able to fetch orchestration rules (connection rules for composition).
2. System B fetches orchestration rules (getConfiguration) to perform external connections – in this case it is a connection rule that indicates a connection to System A, Service A.
3. System B searches (LookUp) Service A endpoint (for Application System A) in the Service Registry.
4. System B connects (i.e. makes an operation call) to the Service A.
5. System A searches (LookUp) for the AuthorizationControl service in the Service Registry.
6. System A verifies the access rights of System B Service A consumption towards its Service A producing resource.
7. The producing Service A returns information over the connection.

The communication between System B and System A are established. The way of further information-sharing is described in the actual Service De-

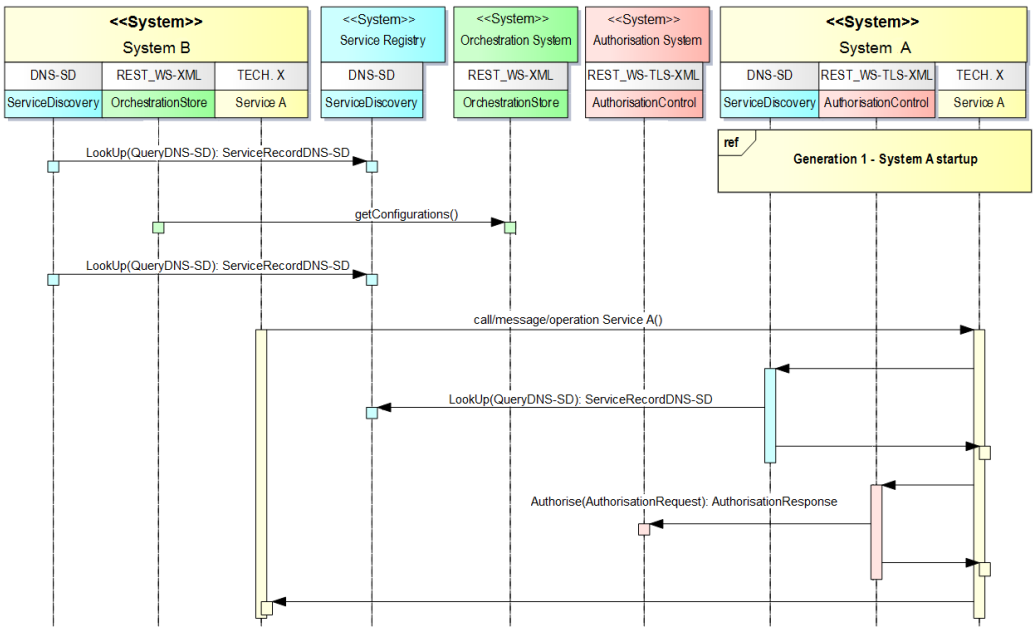

Figure 14: Startup and initial communication of System B

scription and the corresponding Interface Design Description – with reference to the Communication Profile and the Semantic Profile.

4.3. Inter-cloud information exchange

There are various scenarios when the requested services are not available in the local cloud; and there is a possibility to consume it from elsewhere. This problem area is covered by the inter-cloud servicing approach of the Arrowhead Framework, and discussed in [17].

5. Creating Arrowhead-compliant Systems

The Arrowhead System that enables collaborative automation shall comply with the Arrowhead Framework principles. The following steps shall be performed to ensure compliance to the Arrowhead Framework:

- Make design according to Arrowhead Framework templates;
- Adapt legacy system or implement new systems according to the Arrowhead Framework principles/patterns;

– Perform interoperability tests.

Beside the available design templates, the work of system architects and developers is supported through the design documentation guidelines defined in [13].

System design is often created by using already available legacy system components. In order to support interoperability for legacy systems, these should be adapted to the Arrowhead Framework. This can be done through adaptor or gateway components, which can be either integrated to the legacy systems (i.e. within the same hardware element), or can be made available as an Arrowhead core system with such gateway/adaptor capabilities.

Within the Arrowhead project the technology work packages has as objective providing compliance requirements to verify on compliance tests, and to provide test functionality for all partners. Outside the project, an *Arrowhead compliance verification board* supports the verification of individual system and service designs to be tested against a set of compliance criteria.

5.1. Verification of Arrowhead Compliance

The purpose of the verification is to make sure whether the System in question is compliant to the Arrowhead Framework. The following is checked during the verification procedure.

– Can it connect and communicate properly with the core framework components?

– Does it comply with the rules for System documentation set for Arrowhead-compliant systems?

– Does it produce and consume Services of the Arrowhead framework as it is documented within its System Description (SysD)?

In order to technically validate the compliance, the Arrowhead Verification Tool has been created. It supports the following:

– *system test and integration* procedures through manual, automatic and script-tests in order to verify and validate service realization;

– *development* though manual orchestration – to simplify producer and/or consumer interaction (with functionalities such as recording and playback of service interactions);

– *dynamic simulation* of services and the handling of function chains, as well as service relations and their information exchange.

The compliance verification is controlled by personell to evaluate operational as well as documentation aspects within different scenarios.

6. Related Work

There are various frameworks and platforms supporting the Internet of Things (IoT) movement. There is a current survey [18] available that summarizes the commercially available IoT frameworks very well. Well known approaches include the IoTivity framework by the Open Interconnect Consortium [19], the IPSO Alliance framework [20], the Light Weight Machine to Machine (LWM2M) framework by The Open Mobile Alliance [21], the AllJoyn framework by the AllSeen Alliance [22], and Smart Energy Profile 2.0 (SEP2.0), originally from the ZigBee Alliance [23] - among others.

The IoT Architectural Reference Model [24] – defined by the European IoT-A project – is of special interest, since it aims to handle all IoT-related issues at once. IoT-A provides high level models in order to specify a reference architecture. On one hand, this consists of views, such as functional, information, deployment and operation views. On the other hand, this consist of perspectives, such as "evolution and interoperability", "performance and scalability", "trust, security and privacy", and "availability and resilience". These views and perspectives are then utilized to create IoT-A compliant concrete architectures.

The ISO/IEC/IEEE "Systems and software engineering - Architecture description" standard [25] is developed from a consensus of current practices on the topic. It includes a conceptual framework to support the description of architectures, and the required content of an architectural description. Furthermore, it also defines the use of multiple views, reusable models within views, and the relationship of architecture to the system context. The majority of contemporary works utilize concepts and approaches defined there.

The model proposed in by Gross [26] (namely COBRA) was developed with the purpose of flexible composition and reuse of software artifacts is inspired by object-oriented and component-based methods. The method uses UML as primary model-based notation for all analysis and design activities. The model understands a system or a system of systems as a component that can interact with others through interfaces and can be decomposed in other systems or components.

Many cloud-based frameworks employ a data driven architecture. The focus is put on enabling collation of knowledge and performing data analytics. This architecture is well suited for distributed applications such as asset tracking, logistics and predictive maintenance [27].

The frameworks using this architecture generally leave the implementation specifics of the end-points to application developers. This means that they do not provide any support to the edge of the network for the mentioned criteria.

This architecture model is well suited for providing data as a service approach. Management of end-points is simplified to feeding data back to a central repository which can then have complex security authorisations and usage tracking. However the architectural model in general suffers from high latency and bandwidth usage. It does not recognise the evolving computing power at end points as it treats them as "outside" the framework.

A smart objects architecture makes the endpoints active participants in the framework. The end points are included as key aspects of the framework and focus is put on interconnecting the end-points. This architecture is well suited for automation tasks, such as home and building automation, and manufacturing [20].

A typical drawback of many frameworks employing this architecture is the lack of focus on collecting the data within the cloud to perform data-analytics. This implementation specifics of the data in the cloud is left to each designer with minimal support from the framework.

In several architectures many features such as end-to-end security and layered interoperation suffer due to ad-hoc development either at the end-points or in the cloud.

Newer architectures are taking into account the need to satisfy real-time automation requirements while not hindering the value of semantic big data and data analytics. Frameworks based on an architecture which enables standards based development of end points and of data warehousing, will enable secure interoperation of modularised and distributed applications.

Fortino et. al. proposes an agent-oriented and event-based framework for the development of cooperating smart objects in [28] and presets its integration with cloud computing in [29]. In the proposed framework, smart objects are modelled as agents that can cooperate as a multi-agent system to fulfill specific goals. The CA-IoT framework aims to tackle the issues of processing power and storage resources through the use of cloud computing, hence it realizes a cloud-assisted and agent-oriented IoT architecture.

The INTER-IoT project [30] is specifically addressing interoperability issues of IoT platforms and even ecosystems – and proposes the INTER-FW framework for a solution. This framework follows a multi-layered approach, integrating different IoT devices, networks, platforms, services and applications in order to allow a global continuum of data, infrastructures and services. The fundamental layers of INTER-FW are: device, networking, middleware, applications, and semantics.

The Arrowhead architecture approach in some way utilizes the COBRA model approach, but it goes beyond of it. The required architecture has to make significant emphasis on the variety of stakeholders and their needs, due to specifics of the system of systems.

In addition to related work on architecture and framework level, one can find numerous SOA protocols that supports these architectures and frameworks. Such protocols are CoAP [10], XMPP [11], MQTT [31].

7. Conclusion

The Arrowhead Framework provides fundamental functionality to support the development of interoperable, SOA-based automation systems.

This framework comprises a set of guidelines for the development of such systems, which takes into consideration the requirements of designing SOA-based systems. To this purpose the Framework already provides a set of rules for architectural setup, which splits itself in three levels: System of Systems, Systems and Services. The definition and descriptive documentation of these elements support proper understanding between various development groups. This way it helps to insure the interoperability among different systems.

Furthermore, the Arrowhead Framework defines a basic set of core systems, which are mandatory for all Arrowhead-compliant installations, specifically: Service Registry, Authorization and Orchestration. Additionally, other core systems can also be deployed, which ease on common tasks required by such installations. Finally, the compliance of a system to the Arrowhead Framework must also be proven in order to insure its full compatibility, which is supported by specific test tools.

In order to prove its suitability at various ranges of applications, the Arrowhead Framework is currently being used in over 20 different installations, covering the domains of home and industrial automation, production, virtual markets of energy and electrical vehicles infrastructures.

Acknowledgment

This work is supported by the EU ARTEMIS JU funding, within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD).

References

[1] R. Franke, J. Doppelhamer, Integration of Advanced Model Based Control with Industrial IT, Vol. 358, Lecture Notes in Control and Information Sciences, 2007.

[2] Siemens, Simatic s7-1200 programmable controller, System Manual, Nurnberg, Germany (2012).

[3] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, T. Bangemann, Towards an architecture for service-oriented process monitoring and control, in: Conference of the IEEE Industrial Electronics Society (IECON), Phoenix, AZ., 2010.

[4] A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, E. Jose L. Martinez Lastra, Industrial Cloud-Based Cyber-Physical Systems - The IMC-AESOP Approach, Springer, 2013.

[5] N. Kaur, C. McLeod, A. Jain, R. Harrisson, B. Ahmad, A. Colombo, J. Delsing, Design and Simulation of a SOA-based System of Systems for Automation in the Residential Sector, IEEE ICIT, 2013.

[6] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, J. Gustafsson, Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture, IEEE Transactions on Industrial Informatics (2013).

[7] Artemis, http://www.arrowhead.eu.

[8] T. Erl, SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl), Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[9] R. T. Fielding, Architectural styles and the design of network-based software architectures, Ph.D. thesis, University of California, Irvine (2000).

[10] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (coap) (2014).

[11] P. Saint-Andre, Extensible messaging and presence protocol (xmpp): Core (2011).

[12] OPC Foundation, Opc unified architecture specification (2010).

[13] F. Blomstedt, L. Lino Ferreira, M. Klisics, C. Chrysoulas, I. Martinez de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, P. Varga, The arrowhead approach for soa application development and documentation, in: Conference of the IEEE Industrial Electronics Society (IECON), 2014.

[14] M. W. Maier, Architecting Principles for Systems-of-Systems, Vol. 1 of 4, 1998, pp. 267–284.

[15] H. Derhamy, P. Varga, J. Eliasson, J. Delsing, P. Punal Pereira, Error handling for multi-protocol soa systems, in: IEEE International Conference on Emerging Technologies and Factory Automation (EFTA), 2015.

[16] K. Nagorny, R. Harrison, A. Colombo, G. Kreutz, A formal engineering approach for control and monitoring systems in a service-oriented environment, in: IEEE International Conference on Industrial Informatics (INDIN), 2013, pp. 480–487.

[17] P. Varga, C. Hegedus, Service interaction through gateways for intercloud collaboration within the arrowhead framework, in: 5th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace and Electronic Systems (Wireless VITAE), 2015.

[18] H. Derhamy, J. Eliasson, J. Delsing, P. Priller, A survey of commercial frameworks for the internet of things, in: IEEE International Conference on Emerging Technologies and Factory Automation (EFTA), 2015.

[19] IoTivity, Iotivity, a linux foundation collaborative project, https://www.iotivity.org/ (2015).

[20] IPSO-Alliance, Ipso alliance commercial devices, http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf (2015).

[21] OMA, Open mobile alliance - machine to machine (m2m) solution, http://openmobilealliance.org/about-oma/work-program/m2m-enablers/ (2012).

[22] AllSeen-Alliance, Allseen alliance wiki, https://wiki.allseenalliance.org/ (2015).

[23] ZigBee-Alliance, Smart energy profile 2 application protocol standard, http://splintered.net/z/Zigbee-smart-energy-profile-2.pdf (2013).

[24] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, S. Meissner, Enabling Things to Talk - Designing IoT solutions with the IoT Architectural Reference Model, Springer, 2013.

[25] ISO/IEC/IEEE 42010-2011, Systems and software engineering - architecture description (2011).

[26] H.-G. Gross, Component-Based Software Testing with UML, Springer, 2004.

[27] LogMeIn, Solutions by business opportunity, https://xively.com/solution/ (2013-2014).

[28] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, W. Russo, An agent-based middleware for cooperating smart objects, in: Practical Applications of Agents and Multi-Agent Systems (PAAMS), 2013.

[29] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Integration of agent-based and cloud computing for the smart objects-oriented iot, in: IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2014.

[30] H-2020, Inter-iot, http://www.inter-iot-project.eu (2016-2018).

[31] Z. Shelby, K. Hartke, C. Bormann, Message queuing telemetry transport (mqtt) version 3.1.1 (2014).