Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Telecommunications and Media Informatics

Making things collaborate

Interworking of Collaborative System-of-Systems in the Arrowhead Framework

Csaba Miklós Hegedűs

Supervisor: Dr. PálVarga

Budapest, Hungary

2015.

Table of Contents

Kivonat 3

Abstract 4

Acknowledgements 5

1. Introduction 6

  1.1. Motivation and objectives 6

  1.2. Outline of the work 8

2. Related work 9

  2.1. The Arrowhead Technology Framework 9

    2.1.1. Service Oriented Architectures 9

    2.1.2. Introduction to the Arrowhead Framework 11

    2.1.3. Core Systems and Services 12

    2.1.4. System-of-Systems 15

  2.2. Gateways in other technological domains 17

    2.2.1. The IP world 17

    2.2.2. The VoIP world 18

    2.2.3. A world full of IT clouds 19

    2.2.4. Tasks of a border guard 20

3. Interworking between System-of-Systems 21

  3.1. Gatekeeper as a new Core System 22

  3.2. Global Service Discovery 26

    3.2.1. Trust issues 27

    3.2.2. Hardwired Neighborhoods 27

    3.2.3. Ad hoc connections 29

    3.2.4. Utilization of a cloud hierarchy 29

  3.3. Inter-cloud Negotiations 31

4. Effects on the Arrowhead Framework 33

  4.1. Extension of the Orchestration System 33

  4.2. Security measures and other considerations 35

    4.2.1. Authentication and network security 35

    4.2.2. The Arrowhead Service Inventory 35

    4.2.3. Monitoring and handling events 37

5. Conclusions 38

Appendix 1. Gateways in the different technological domains 40

References 41

Kivonat

A Dolgok Internete (Internet of Things, IoT) jövőnk egyik kiemelkedő informatikai problémája. Manapság mindent mindennel összekapcsolunk, megcélozva a globális elérhetőséget. Egyes előrejelzések szerint 2020-ra 50 milliárd IoT eszköz lesz [1] és az emberiséget ezen szétszórt erőforrásokat adó kis beágyazott rendszerek tömkelege fogja segíteni.

Ez azonban még a (nem túl) távoli jövő, melynek megteremtéséhez számos alapvető probléma megoldásán keresztül vezet az út. Ezen problémák nagy része az együttműködés képességének hiányában rejlik. A számtalan hardver, egyedi szoftver, programozási nyelv és technológia egy sor olyan iparágat teremtettek, amelyeknél szinte lehetetlen közös alapokat, közös nyelvet találni. Emiatt kulcskérdéssé vált ezen feloldhatatlan heterogenitás kezelése.

Az Arrowhead projekt [2] egy konzorcium víziója, mely az Európai Unió támogatását is élvezi. Küldetése, hogy keretet adjon az ipari automatizálás területén nem csak a technológiai, hanem a teljes megvalósításban az együttműködés megteremtésére. Öt fő területet céloz meg: ipari termelés és feldolgozóipar, okos városok, elektromobilitás, energiatermelés és a villamos energia piac automatizálása.

Az Arrowhead Keretrendszer már számos kísérleti esettanulmányban bizonyította hatékonyságát, melyet a Szolgáltatás-orientált Architektúra (Service Oriented Architecture, SOA) alapú megközelítésének köszönheti. Ezzel válik lehetségessé az ipari automatizálás területén kialakult extrém heterogenitás kezelése, így bármilyen rendszer beilleszthetővé válik az Arrowhead felhő alapú megoldásába.

Ezen Arrowhead Felhőkben néhány központi entitás segítségével valósul az egyes rendszerek együttműködése. Azonban egyetlen Arrowhead Felhő nem tud mindent és mindenkit kiszolgálni, így szükség lesz az egyes Felhők között is az együttműködés képességének megteremtésére. E dolgozat feltárja az ilyen felhők együttműködésével szemben támasztott elvárásokat és használati eseteket, majd pedig a rendszer architektúra és folyamatok szintjén tesz javaslatokat e különböző esetek egységes megoldására.

Abstract

In this upcoming world of IoT (Internet of Things) and global interconnectedness, it is a grand challenge to create interoperability between any parties. It has been forecasted that there will be 50 billion IoT devices in 2020 [1], and mankind will be aided by globally distributed collaborating cyber-physical systems aided by cloud-based processing.

However, we are not quite there yet. Several fundamental issues are still not solved. One of the many problems stem from the lack of interoperability on all design levels. The endless varieties of hardware and software architectures, programming languages and protocols used in various industries make it hard to enable interoperability of these heterogeneous systems.

The Arrowhead Project is the vision of a consortium with the backing of the European Union [2]. It aims nothing less than to create a full-scoped framework enabling collaborative automation by networked embedded devices. It targets use cases from five business domains: production (process and manufacturing), smart buildings and infrastructures, electro mobility, energy production and virtual markets of energy.

The Arrowhead Technological Framework has proven its effectiveness in several pilot (use case) scenarios with its Service Oriented Architecture (SOA) approach. It is possible to handle extreme heterogeneity and integrate almost any kind of device (a system) into a larger System of Systems (SoS). This is achieved by the creation of a cloud-based environment with central entities that orchestrate the members of the SoS how to conduct their "business".

As one single Arrowhead Cloud (one SoS) cannot serve for all, there is a significant need for inter-cloud operations to achieve true global interconnectivity within the Framework. This work explores the different use case scenarios and their aspects and proposes design principles based on them while respecting the current concepts.

Acknowledgements

I would like to express my deepest gratitude towards my supervisor, Dr. Pál Varga, for the opportunity to work on this project and for his continuous engagement. His guidance and support made it possible to create this work.

1. Introduction

1.1. Motivation and objectives

There are various technology domains – from electro-mobility through smart buildings and infrastructures to energy production – that have a great impact on our life. This impact will be even more positive when the interoperability of these systems becomes a reality. There should be some commonly defined patterns regarding the communication and orchestration of these currently independently designed systems. Such patterns will enable standard design methods of complex services provided by functionally heterogeneous System of Systems.

In this modern age of the Internet, our daily lives are increasingly impacted by various aspects of the Information Technologies. There were over 2.6 billion smartphones worldwide in 2014 [3]: everyone seems to have at least one smartphone or some other wearable technology on them. Everyone wants to be online and accessible all the time. This is not only true for humans, but all sorts of devices as well: it is the dawn of the Internet of Things (IoT). We are trying to connect everything in one universal and global network. It is estimated that there will be 50 billion IoT devices in 2020 [1].

These type of forecasts are very common, depicting exponential growth in IoT devices and predicting an automated future, where mankind is aided by globally distributed collaborating cyber-physical systems.

Although, we are not quite there yet. Several fundamental issues are still not solved and very often overlooked because of other advances in the field. The pre-existing internet protocol suite surely provides a good technological platform for the IoT world: its major issue with global addressability seems to be resolved by the introduction of IPv6 [4]. Nevertheless, the overall penetration of IPv6 is still less than 9% according to Google [5].

A majority of the problems stem from the lack of interoperability on all design levels. The endless varieties of hardware, operating systems, middleware and programming languages created an industry where it is fundamental to deal with heterogeneity. It is really hard to create collaboration between parties in any sense.

The idea of IoT is also deeply connected to other current IT buzzwords: clouds, distributed computing, "smart devices" or even distributed sensing and actuating technologies. The standalone devices are mostly connected to a cloud-based system to perform the necessary operations that cannot be done on the single, hardware-constrained device. Since these cloud-based framework solutions are mostly closed-sourced products they are not designed to be compatible with other systems (i.e. from other vendors).

The Arrowhead Project [2] is the vision of an ECSEL [6] consortium with the backing (and funding) of the European Union. It aims nothing less than to create a full-scoped framework enabling collaborative automation by networked embedded devices. The grand challenges it faces are the creation of interoperability and integrability of almost any device. Initiatives like these could solve the ever growing problem of the deep-rooted incompatibilities of the IoT and automation world [2].

The ECSEL Arrowhead project started in 2013 with over 80 partners. These stakeholders also show heterogeneity as opening to an integrated IoT world is in the interest of many technological areas. From smart bearings to smart cities, there is the need for interoperability and convergence.

Since the primary objective is to create a common platform for all sorts of future implementations, the primary pilot (use case) scenarios of the project involve several business domains:

- industrial automation and production: both the processing and manufacturing sectors,
- smart buildings and infrastructure and smart cities,
- electro mobility: smart electric cars, smart traffic control,
- energy sector: virtual market of energy, energy production and end-user services [7].

The project also covers various activities in order to reach the above mentioned, general targets. These involve (i) creating a cloud-based overall system architecture based on Service Oriented Architecture (SOA) principles [7], (ii) establishing common design and documentation guidelines [8] and (iii) building pilot applications based on its technological framework.

This framework has proven its effectiveness in local scenarios with its service oriented approach. An energy consumption forecasting and balancing system using smart power meters has been created with the Arrowhead FlexOffer in Lulea, Sweden. The project partners both in Barcelona (Spain) and in Oulu (Finland) are creating a smart city model with intelligent urban lighting and energy-conscious buildings all communicating within the same Arrowhead network. Another one of the automation world pilots consist of an Arrowhead-aided maintenance management system for production plants. Up to now, over 20 of such pilots have been demonstrated all over the continent.

Still, as one single Arrowhead Cloud cannot serve for all (at least it probably cannot handle 50 billion devices), there is a significant need for this work about inter-cloud operations to achieve true global interconnectivity.

This paper presents high-level design approaches for current problems in the project by proposing several additions and modifications of the current overall architecture, while respecting the current system design principles. Furthermore, it also identifies the advantages and issues that arise with those new concepts and handle them on the chosen design level.

1.2. Outline of the work

In the following chapters this paper will analyze and augment the Arrowhead Technological Framework. In chapter 2, I will present the Framework in details: I will characterize the SOA based approach of the project together with the System-of-Systems and Service abstractions. After that, I will review the Core Systems and Services of the cloud-based Framework. In section 2.2 different gateway concepts and functionalities will be mapped out in the various technological domains. This part will be essential for chapter 3 as the identified characteristics will be used later in this work.

In chapter 3, inter-cloud operations will be discussed. After setting up the basic use case scenarios and preliminary specifications, I will introduce the Gatekeeper as a servicing module extending the pre-existing Arrowhead Core Systems. Using the Gatekeeper module, I will identify a methodology on how two Systems can find each other if they belong to two different Arrowhead compatible cloud environments. This process will consist of two major steps each conducted by the Gatekeeper System - and this whole procedure will be presented. Here, the different architectural scenarios and their properties will also be argued.

In chapter 4, I discuss the impacts of the proposed inter-cloud interaction methodology on the current Arrowhead Framework. Several adjustments will be necessary on the Framework to support inter-cloud functionality in the manner the IoT and the automation world need it to be. The augmentation of the current Core System functionalities, the introduction of further security measures and the creation of further development tools will also be discussed here.

In chapter 5, I will summarize the presented work and discuss how it will and should proceed.

2. Related work

2.1. The Arrowhead Technology Framework

In this chapter, I will discuss the Arrowhead Technology Framework, from here on referred to as the Arrowhead Framework. The fast evolution of the IoT and the industrial automation world is defining new challenges for engineers. The authors of [10] flagged the following IoT requirements with major importance:

- Scalability: there will be over 50 billion devices connected by 2020 [1];
- Dynamic: Smart object are dynamically deployed, therefore an architecture should support Plug-and-Play type of functionality;
- Communication constraints: IoT devices usually have very limited (mostly wireless) communication capabilities;
- Processing constraints: these low-powered devices have very limited processing resources and have to be energy consumption conscious;
- Extreme heterogeneity of devices, clients and purposes.

These expectations are not easy to meet. Especially, if we measure in the fact, that large scale industrial automation systems are mostly legacy commercial off-the-shelf (COTS) systems with monolithic design (i.e. Siemens' Simatic [11]). These should also be fitted in the IoT world to provide (real-time) flexibility for production.

2.1.1. Service Oriented Architectures

All of these requirements can be accommodated by the Service Oriented Architecture (SOA) approach. There are several definitions of the service oriented modeling paradigm. The Open Group defines SOA as the following [12]:

"Service-Oriented Architecture is an architectural style that supports service-orientation. Service-orientation is a way for thinking in terms of services and service-based development and the outcomes of the services."

This definition emphasizes the importance of service-based thinking, but does not clarify services any further. Meanwhile, Gartner [13] defines SOA in a totally different, highly technical approach:

"A software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls."

These definitions do not help to understand SOA principles. Basically, SOA is a system architectural philosophy for creating highly flexible systems with an incremental development style. It builds on the interface concept from the object-oriented programming paradigm. A SOA based system use a collection of loosely coupled components that provide so-called services to each other via specialized interfaces [14]. Figure depicts a basic view on how system provide and consume services in the Arrowhead Framework.

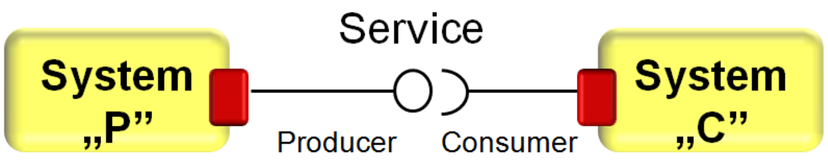


Figure 1. The producer and consumer of a Service

The building blocks of the SOA systems are these *services*. There is no standardization on what characteristics services have or how they can be implemented. The authors of [15] have collected the following general qualities of SOA services:

- Services are dynamically discoverable and boundable.
- Services are self-contained and modular. A service supports a set of interfaces and these interfaces are logically cohesive (they implement the same functionality).
- Modular decomposability: a complex service can be created from simpler atomic services. Designers have to seek sufficiently modular service descriptions and can compose complex services from the atomic ones.
- Modular understandability: the user has to be able to use the service without having knowledge of any other underlying atomic services used.
- Modular continuity and protection: the service interface has to hide its implementation details and any abnormal servicing cannot have an impact on other services or their state of internal data.
- Interoperability: systems using different (often legacy) platforms and languages are able to communicate with each other.
- Loose coupling: SOA strives to only have few well known dependencies between modules.

Although SOA is often realized with Web Service technologies, it is worth noting that SOA principles can be implemented in any system using (nearly) any technology. Moreover, SOA is not even necessarily a strictly software engineering tool: it can even be a business process engineering methodology. Not just ERP (Electronic Resource Planning) systems, but whole organizations can be (and has been) tailored to operate on SOA principles [16].

2.1.2. Introduction to the Arrowhead Framework

The Arrowhead Framework is also based on these fundamentals. The building elements of the Framework are *Systems* that provide and consume *Services*, and cooperate as a larger *System-of-Systems*, also known as Arrowhead Local Clouds.

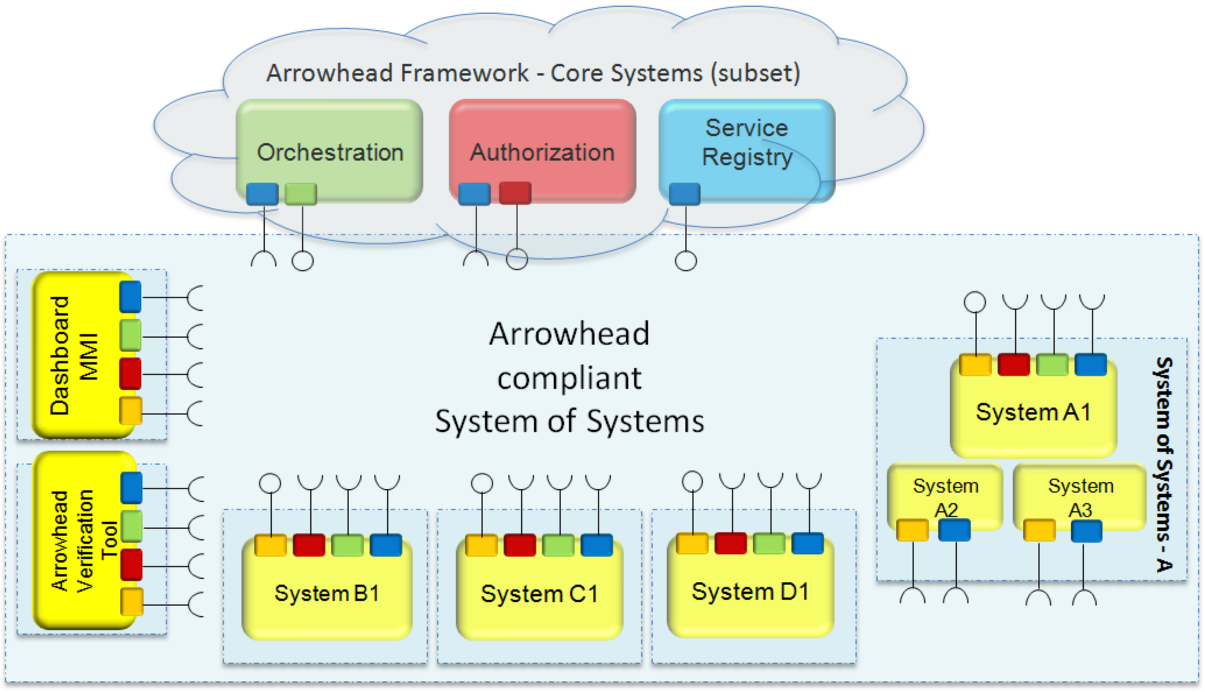


Figure 2. Arrowhead System-of-Systems with their Core System instances [8]

Therefore a Service is what is used to exchange information from a providing System to a consuming System, as shown in Figure 2. In a Service, capabilities are grouped together if they share the same context [17]. In the context of Arrowhead, a resource can be a temperature sensor or a power consumption meter. In this sense, a Service is connected to the temperature itself regardless of the access method or technology. This principle is supported in the Arrowhead Framework, as a Service by definition can be realized with different interface implementations such as RESTful Web Services [19], OPC UA [29], COAP [30] or XMPP [31] protocols and still belong to the same Service definition.

All Arrowhead compliant Systems will also be capable of consuming different types of Services in their different implementations -they might need and compatible with - during runtime. Services are also not bounded by System instances, since a Service can be realized in an arbitrary number of Service Producers and Service Consumers.

As mentioned before, Services can be grouped ("orchestrated", see section 2.1.3.) together to fulfill a more complex task, therefore ensures the composability of services. In this way, non-functional requirements can be fulfilled while servicing, like additional logging or other reporting duties. These are called Complex Services, and they build on the atomic Services.

However, there is a certain set a Services that need to be implemented to create a minimal working SOA based Arrowhead Framework. In the following section 2.1.3., these Mandatory Core Services and their Mandatory Core Systems will be described.

Furthermore, these functioning Arrowhead Framework compliant System-of-Systems will be called Arrowhead Local Clouds. One Arrowhead Local Cloud consists of various Systems and their Services and has its own instance of the Mandatory (and some other optional) Core Systems by definition. It also has its own set of stakeholders associated with it, i.e. its operator. Building on the Core Systems described in the following section, the overall architecture of such Arrowhead Local Clouds will be also dissected in section 2.1.4.

2.1.3. Core Systems and Services

The Core Systems cover essential functionality for all components and is available for all kind of Systems of that Local Cloud. All of them can be used dynamically at runtime so that the flexibility and adaptability of the SOA approach is maintained. They consist of two groups based on whether they are mandatory or optional as depicted in Figure 3. These Systems are provided by the Arrowhead Framework with a reference implementation and the documentation for self-realization.

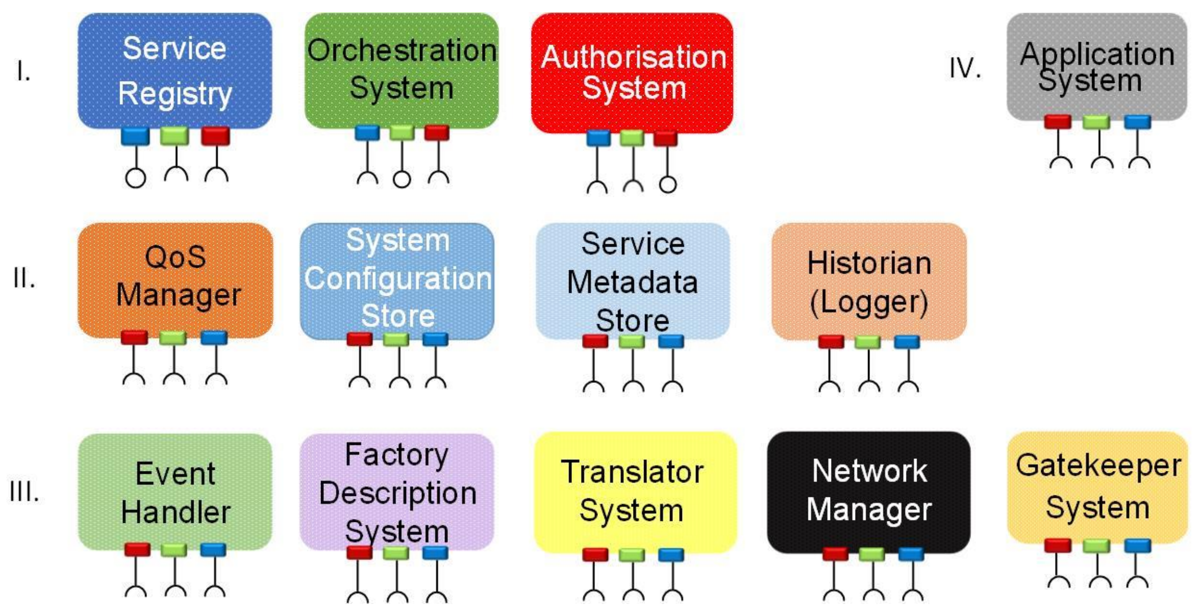


Figure 3. The mandatory (I.) and optional (II. and III.) Core Systems and other Systems (IV.) with the extensions of this work (the Gatekeeper and Network Manager Systems)

When a System wishes to consume Services or advertise its own production of some Services for its Local Cloud, it will turn to Core Services. However, it is not mandatory to use all Core System elements in all type of servicing, but the Core Services functionality cannot be replicated in any other elements than the Core Systems.

The Service Registry System keeps track of all active productive Services within the managed domain. As its name suggests, it supports a service registry functionality currently based on DNS-SD [18], see section 2.1.4. All Systems within the network have their services published within the Service Registry, and all Systems can access the Service Registry to look for resources via the Service Discovery Service that is offered by this module.

The Orchestration System controls how Systems can access Services. It is utilized to dynamically assist the usage of the different Services. The Orchestration Services are used to control how Systems are serviced: Systems that request a complex Service has to access the Orchestration first to obtain the methodology how to procure that specific (complex) Service and what additional steps are required (i.e. to provide logging). It provides the "choreography" how the transaction has to go down by providing the composition of the atomic Services provided and consumed during the complex servicing.

The last mandatory Core System is the Authorization Manager. It provides the possibility to manage the access rules for specific resources. This System has to be accessed first to obtain authorization tickets for nodes to establish trust and begin servicing. This System is also accessed by the Orchestration for obtaining preliminary authorization information to redirect the requesting node to a suitable partner.

These three Core Systems form the minimal Arrowhead Framework based System of Systems (Arrowhead Cloud). In Figure 4 the whole process of a System waking up and connecting the Arrowhead Cloud is shown. The detailed description of this procedure can be found in [8].

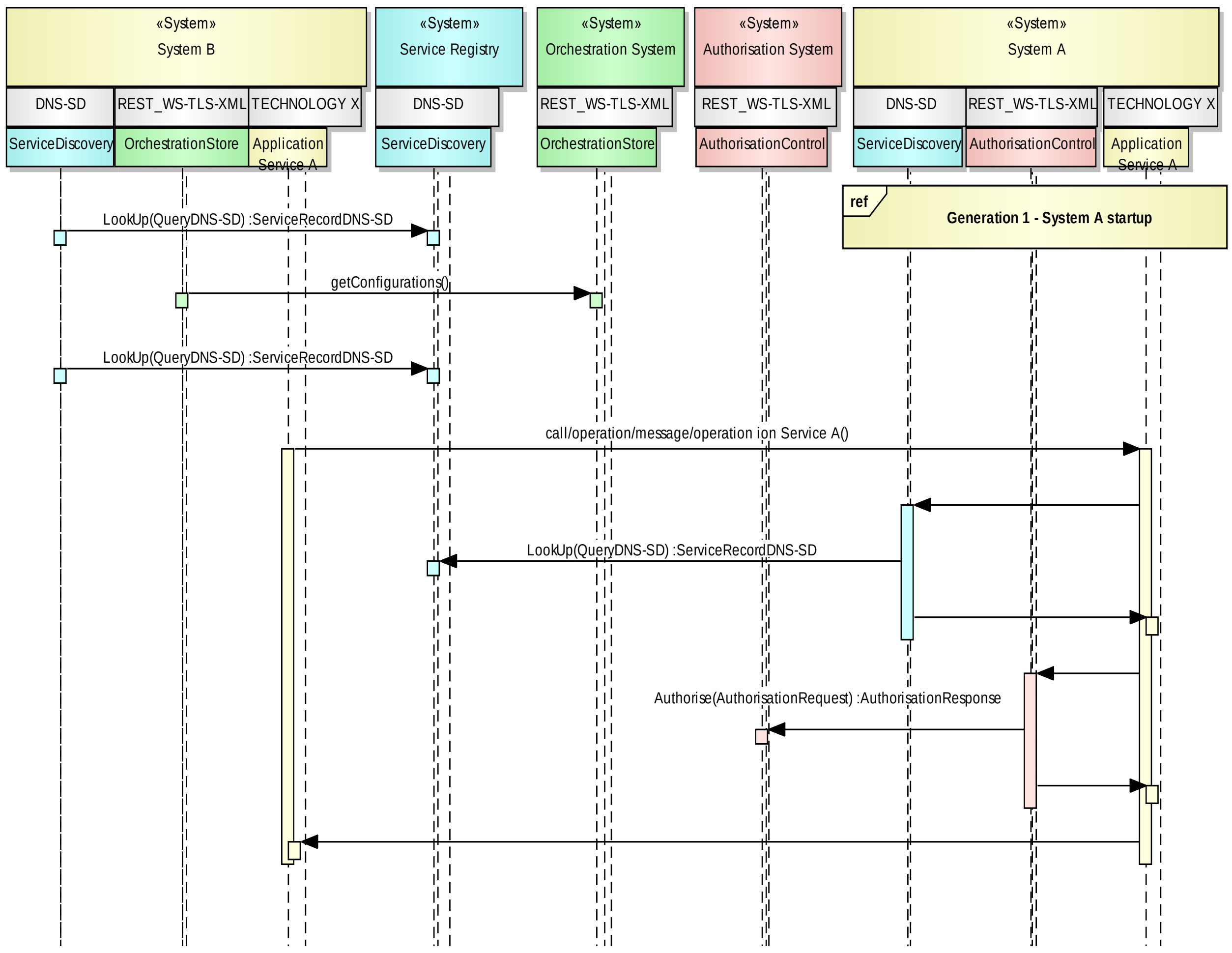


Figure 4. Startup and initial communication of a System in Arrowhead [8]

There are other (non-mandatory) Core Systems currently under development to help the Framework to be adaptable to all environmental conditions in the automation (and IoT) world.

The Service Metadata and System Configuration Stores provide storage for Systems that can be used for run-time reconfiguration and saving metadata of the Services. The QoS Manager concept aims to make the real-time constrained embedded devices integrable in a Local Cloud (i.e. actuators). This concept will be necessary in the revision of the Orchestration System.

The Historian hosts application data for Systems, and stores the data in query-able form. The Translator System can provide application-level translation so that a producer and consumer implementing a different protocol of the same Service can still interact. Event handlers are tasked with the run-time detection of various events and errors and triggering the appropriate Systems.

The further additional Core Systems and Service functionalities are not in the scope of this work as they are not necessary to understand the proposed inter-cloud servicing architecture.

2.1.4. System-of-Systems

When Systems use the same instances of the mandatory Core Systems and hence belong to the same Local Cloud, they also form a System-of-Systems. Figure 5 depicts such Local Clouds that serve local purposes.

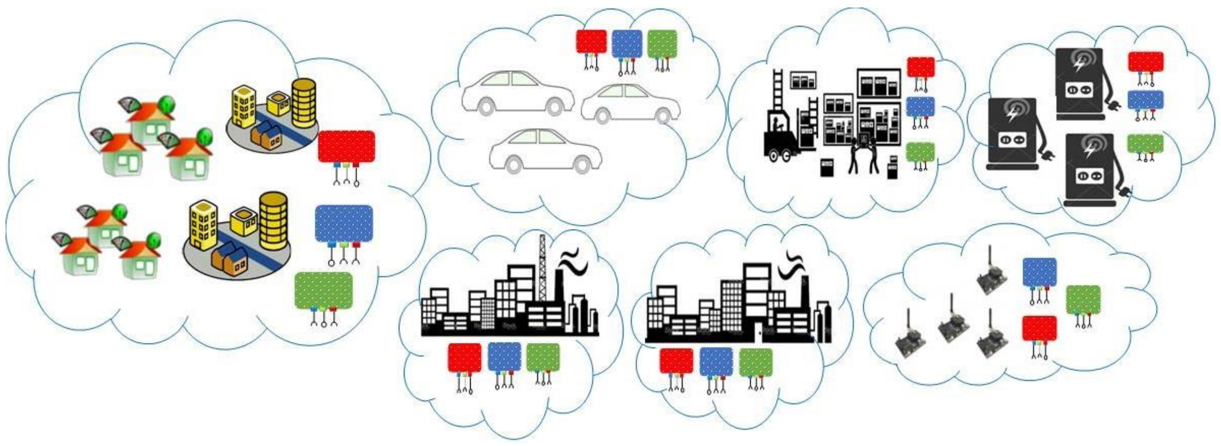


Figure 5. Arrowhead Local Clouds in application

Becoming an Arrowhead Cloud is not bounded to any restrictions yet, as Figure 5 suggests. There are five main characteristics of the Arrowhead System-of-Systems that differentiates it from other large scale, but monolithic systems [9]. These features of the Arrowhead System-of-Systems are:

- the operational independence of its Systems,
- the management independence of the incorporated Systems,

- evolutionary development of its parts,
- emergent behavior of its parts, and
- their Systems can be geographically scattered.

Regarding the technical terms of creating Arrowhead-compatible devices from legacy systems, the Arrowhead Framework defines three Maturity Levels (MLs), as Figure 6 shows. Systems that can be integrated into Arrowhead vary from a single sensor to a whole automated factory, depending on the use case. Therefore, they might require different augmentations to be compatible with the Framework.

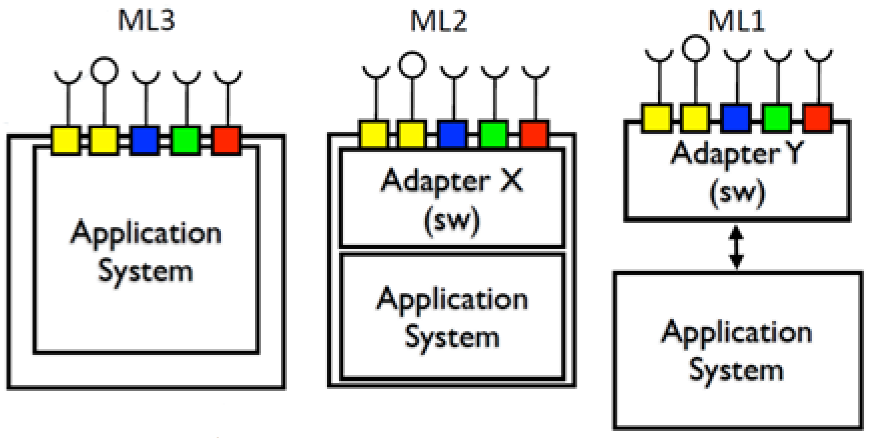


Figure 6. Ways to integrate devices into the Arrowhead Framework [8]

The Arrowhead Framework aims to support all types of technologies in its final architecture. In the current state of the Framework, the Service Registry is implemented using DNS based Service Discovery protocol dedicated to resource management in a network [18].

It uses an (string) identifier for the Service implementation and the Systems running it. For a service request, the server provides information about suitable network nodes by using SRV (service), TXT (text) and PTR (pointer) records. The hosts offering services have to publish details of their available services: instance, service type, domain name and optional configuration parameters [ibid.].

The current Service Discovery, however, does not involve any optimized matchmaking process, but simply provides one appropriate Service Provider. For the support of the inter-cloud servicing concept, this shortcoming of the Framework will have to be addressed. Addressing this issue is part of my contribution to the topic and will be detailed in section 4.1.

The Orchestration, Service Metadata and System Configuration Stores are implemented in RESTful protocols [19], can be secured by TLS (Transport Layer Security) and the data structure is mostly based on the XML (eXtensible Markup Language) format. This approach provides the technological aspect for scalability and adaptability as constrained devices and cloud-based systems are both capable of implementing them appropriately.

2.2. Gateways in other technological domains

The Arrowhead Framework currently lacks any inter-cloud servicing support. In order to close this gap, I propose to use a specific module, the Gatekeeper as an addition to the optional Core Systems. Its specification and functionality will be discussed in chapter 3. The current section sets to discover what is done on the border of the different technological domains and whether any analogies can be drawn from other technologies for this work.

Gateways, firewalls and gatekeepers serve various purposes in the different technological domains. They might stand at the border of a subnetwork, a service area or a mobile network operator – in general: they stand between various network or service domains. Their purpose is to provide means of connection and data exchange between the internal and external worlds. There could be very specific requirements set against them in the different applications.

I have examined the following technological domains:

- network management of heterogeneous systems [20],
- the world of the Internet: current IP solutions [22],
- the VoIP (Voice oer IP) system architecture (H.323 and SIP),
- cloud-based solutions: IaaS, PaaS, SaaS (Infrastructure, Platform, Software as a Service) [25] [28],
- and military applications: vehicular mobile ad hoc networks [21].

All of these areas contain valuable and relevant approaches, technologies or ideas to use while creating the Gatekeeper’s design for the Arrowhead Framework.

2.2.1. The IP world

The IP world is full of different network boundaries. Firewalls are used at the border of the local network to provide security from the outside world. They have come a long way from the simple packet filters; nowadays firewalls provide numerous features on the different levels of the so-called TCP/IP stack. Modern firewalls usually provide the following services [22]:

- Stateless access control (passive filtering based on i.e. address and port);
- Maintenance of stateful information about connection sessions to prohibit out-of-session malicious packet insertion;
- Control of the private address range with NAT (Network Address Translation) services;
- Assurance of network anonymity and data caching through proxies;
- Sole external access point of a private network: sufficient protection against i.e. Denial of Service (DoS) attacks;
- Log generation for security audits and security breach countermeasures.

These firewalls also have to be highly robust and dynamically configurable.

2.2.2. The VoIP world

The Voice over IP (VoIP) solutions are built on the IP world, but these are application layer architectures considering the TCP/IP stack. There are dedicated gateways in them, which provide very different functionalities compared to firewalls. Gateways in the H.323 system architecture stand between the VoIP subnetwork and the Public Land Mobile Network (PLMN) or Public Switched Telephone Network (PSTN), as seen in Figure 7 [24].

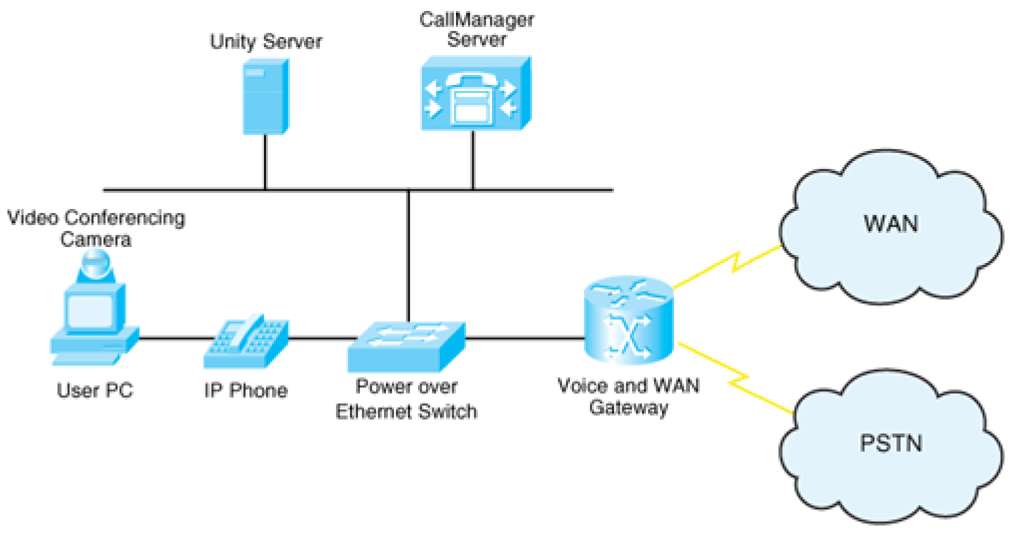


Figure 7. Generic elements of the H.323 system architecture and their connections to the external systems [23]

Address translation here refers to the conversion between ISDN Global Titles (phone numbers) and local H.323 [36] or SIP (Session Initiation Protocol) [37] addresses. This is necessary for VoIP subnetworks, as VoIP addresses are mostly URI based (Uniform Resource Identifier, domain-based), while ISDN operates with pre-assigned area codes and digit-based identifiers [ibid.].

Data conversion is also done here between the various audio codecs used in PLMN, PSTN and VoIP systems. Protocol conversion is also involved in the Gateways as well as management of the link resources (the outgoing phone network lines of the local VoIP network towards the PSTN networks) [24].

2.2.3. A world full of IT clouds

The next domain is highly relevant to the Arrowhead Framework: it is a world of the highly distributed resources, the cloud based technologies. There is a significant need to transfer data between closed, self-contained clouds. These clouds might operate on a different platforms (i.e. OpenStack [25]) but they most likely work in different domains under different operators.

There are many obstacles to tackle here, as security and reliability is a main objective. The central entity (Cloud Networking Gateway) in an inter-cloud management architecture is presented by Mechtri et al. [26]. It handles the following processes for inter-cloud transactions:

- Managing the firewalls and allocating the publicly accessible IP addresses: allocating them to the nodes involved in the current inter-cloud transactions (using the NAT);
- Contacting the partner cloud’s Gateway;
- Injecting routing information into the partner cloud’s network;
- Creating secure VPN connections for the data exchange;
- Managing and translating between the different cloud technology drivers.

The peculiarity of this architecture is the primary focus on including Service Level Agreements (SLA) in these operations to ensure all aspects of the Quality of Service (QoS). This can be achieved by the use of Software Defined Networking (SDN) [27] technologies (such as OpenFlow) controlled infrastructures.

The authors of [28] emphasize different aspects of inter-cloud operations regarding security issues. As a cloud application is intended to be operated on different infrastructures (run by different operators in IaaS, PaaS or SaaS models), virtualization (with virtual machine images) has to be combined with on-demand resource allocation (by instantiation of the virtual machine images). This requires these (virtualized) services to be easy to migrate and fast to scale. Trust becomes essential here, meaning that it is not a neutral factor where and on what the service is actually, physically running. Identity and trust management methods have been proposed to mitigate the risk of the multi-provider and multi-domain servicing.

2.2.4. Tasks of a border guard

To conclude, a wide range of functionality can be attributed to devices on a system (or a subnetwork) border. In the examined technological domains the following tasks and responsibilities have been identified:[1]

- Communication protocol conversion: providing packet format conversion;
- Data (or media coding) conversion: providing data format conversion;
- Address translation: conversion between publicly available IP (and port) and subnetwork addresses (NAT), transparent name mapping (i.e. between the OSI and IP stack [20], global addressability (IPv6) [4]);
- Routing: the Cloud Networking Gateways [26] also have to provide the necessary routing information to the infrastructure managements of the clouds for the current data exchange (for the “navigation” in the hidden subnetworks);
- Content caching, proxies: gateways often have the responsibility of representing nodes in the subnetwork to the outside world or vice versa;
- Content filters, application level security: in some cases, the data flow between the “inside and outside” worlds have to be controlled and filtered by a central entity;
- Authentication: before creating connections, the partners usually have to mutually check the identities of the partner;
- Accounting tasks (i.e. logging): can also be attributed to gateways as they are often in the direct data path;
- Link management: the gateways have to manage the available links to the outside world (i.e. the phone lines towards the PSTN). This can also manifest in the QoS management (in the IP applications) of the links.

These identified functionalities will be heavily used in the following chapters.

[1] A summary of this sub-section can be found in Appendix 1.

3. Interworking between System-of-Systems

There is a valid need for inter-cloud servicing in the Arrowhead Framework. This is because one single Arrowhead Cloud cannot serve for all demands and limiting devices to their own System-of-Systems is not promoting interoperability. This and the following chapters of the paper are the main and novel contributions to the Arrowhead Framework: suggesting ways to handle inter-cloud services. Figure 8 depicts the impact of this work on the current Arrowhead Framework (confer with Figure 5).

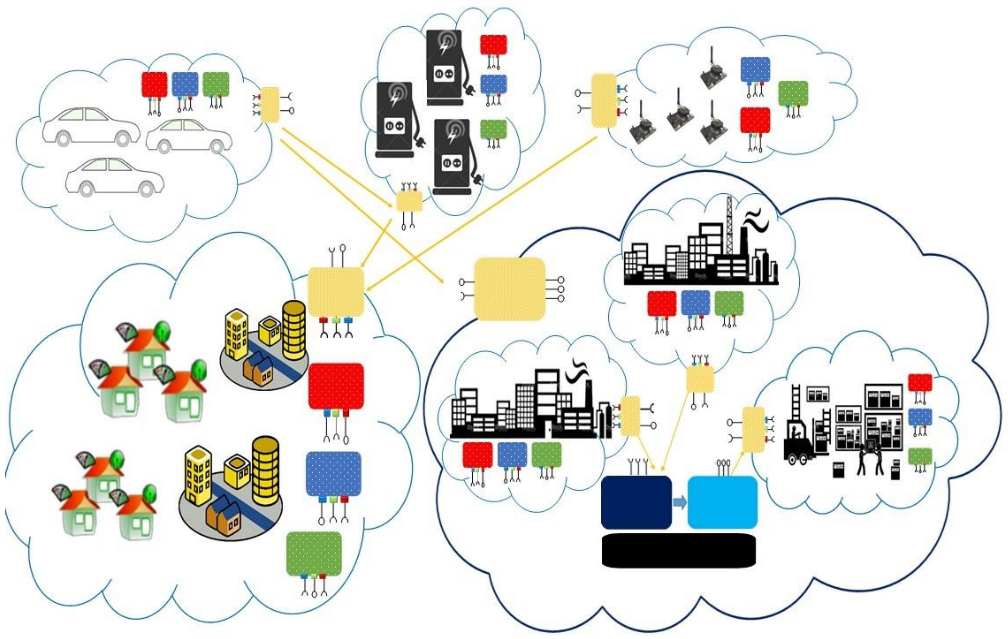


Figure 8. Arrowhead Local Clouds with the capability of inter-cloud servicing

The following primary use cases are considered to be the triggering point for inter-cloud interactions:

- Servicing is not possible: no adequate service provider can be found locally;
- Local service provider is not available: is in out-of-order condition or registered in the network but currently cannot be located (Plug and Play support);
- Local servicing is currently not possible: no free resources (all of the Service Providers are currently overburdened) or the QoS expectations can't be satisfied locally (real-time constraints);
- Servicing within the home cloud is not optimal: geographically a different is closer or somewhat better.

One of the major philosophical questions that determine how services can be handled inter-cloud is about data ownership. Who owns the data of a System? It can be the System itself or the System of Systems ("the local Arrowhead Cloud").

In case the ownership belongs to the standalone System, it means that various intelligent decision-making processes (for example resource management or access control) will have to be handled internally, within that very System. That would lead to the legacy system adapters (see Figure 6) to be inefficiently overgrown.

On the other hand, if it is owned by the managing Cloud, the control of inter-cloud service interactions is handled by the local Core Systems after the System indicates its need for a service. This would mean that the sensitive (e.g measurement) data ownership rights will be "given away" from its producer and the Core Systems will instruct the producer what to do with it.

As this question has not been resolved yet, the proposed inter-cloud communication methodology has to support both sides. In my primary scenario I will only assume that one identity of an Arrowhead SoS Cloud is requesting a Service that cannot be fulfilled in the local cloud, and therefore it might be viable to look for clients in other Arrowhead Clouds.

Security is the other fundament that has to be considered the highest priority in this task. Interactions in a closed Arrowhead Cloud are secured by the central entities (Core Services) and the closed nature of the SoS (local addressing schemes and security firewalls to the outside world). One Arrowhead Cloud has to be absolutely certain that it can completely trust its partner (the Partnering Cloud) when redirecting its node there (similarly as seen in the IT inter-cloud architectures). For this reason, identity verification and authentication will play a key role in inter-cloud servicing, however, those are currently not part of the Arrowhead Framework (see section 3.2.1).

3.1. Gatekeeper as a new Core System

Following the SOA principles, we can only add modules to the Framework that can provide and consume self-contained atomic Services. These elementary Services then can be orchestrated (by a "step-by-step instruction set") into complex ones to realize sophisticated functionality.

Figure 9 shows the proposed Gatekeeper module. This module consumes the regular Core Services from its own Cloud. In this representation, the Gatekeeper Services are provided intra-cloud (to cooperate with the Core Systems), while the Global Service Discovery (GSD) and Inter-Cloud Negotiations (ICN) Services are provided inter-cloud (among the Gatekeeper instances).

The GSD services extend the service discovery functionality between Local Clouds and the ICN process will be dedicated to creating the inter-cloud Service production.

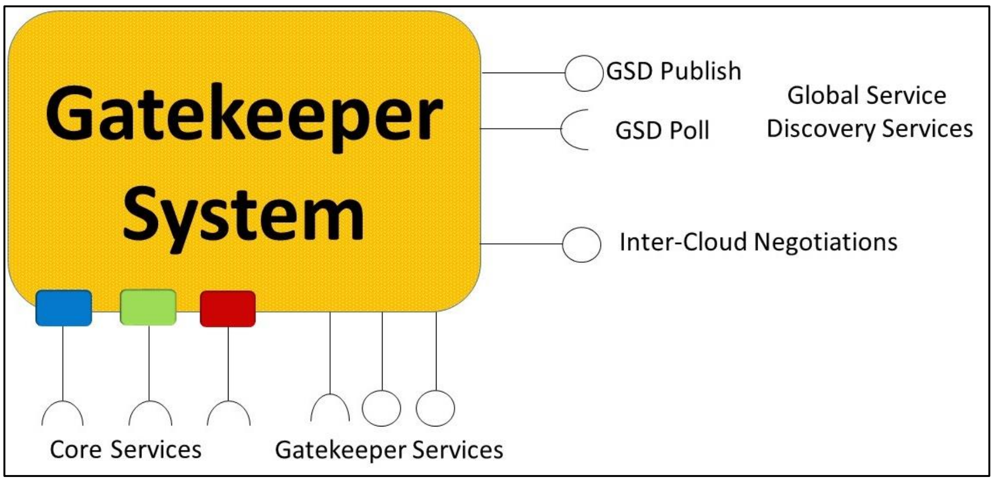


Figure 9. The proposed Gatekeeper servicing module

All inter-cloud interactions are to be handled by these Gatekeepers until the requesting node enters the other cloud (referred to as the “Partnering Cloud”) in order to create the desired producer-consumer connection.

Using this Gateway module (and other Systems in the local Cloud) the following functionality has to be provided:

- **Finding** other Arrowhead Clouds with suitable provider (or consumers) for the requested purposes (Service Discovery functionality);
- **Choosing** where to connect (choosing the “Partnering Cloud”);
- **Connecting** to the Partnering Cloud;
- **Managing** the connection to the Partnering Cloud (and closing it after transaction).

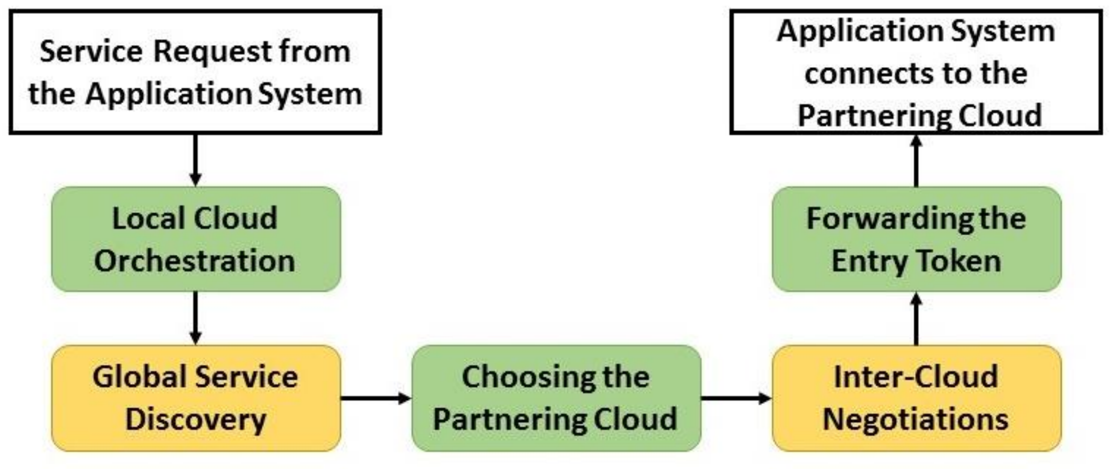
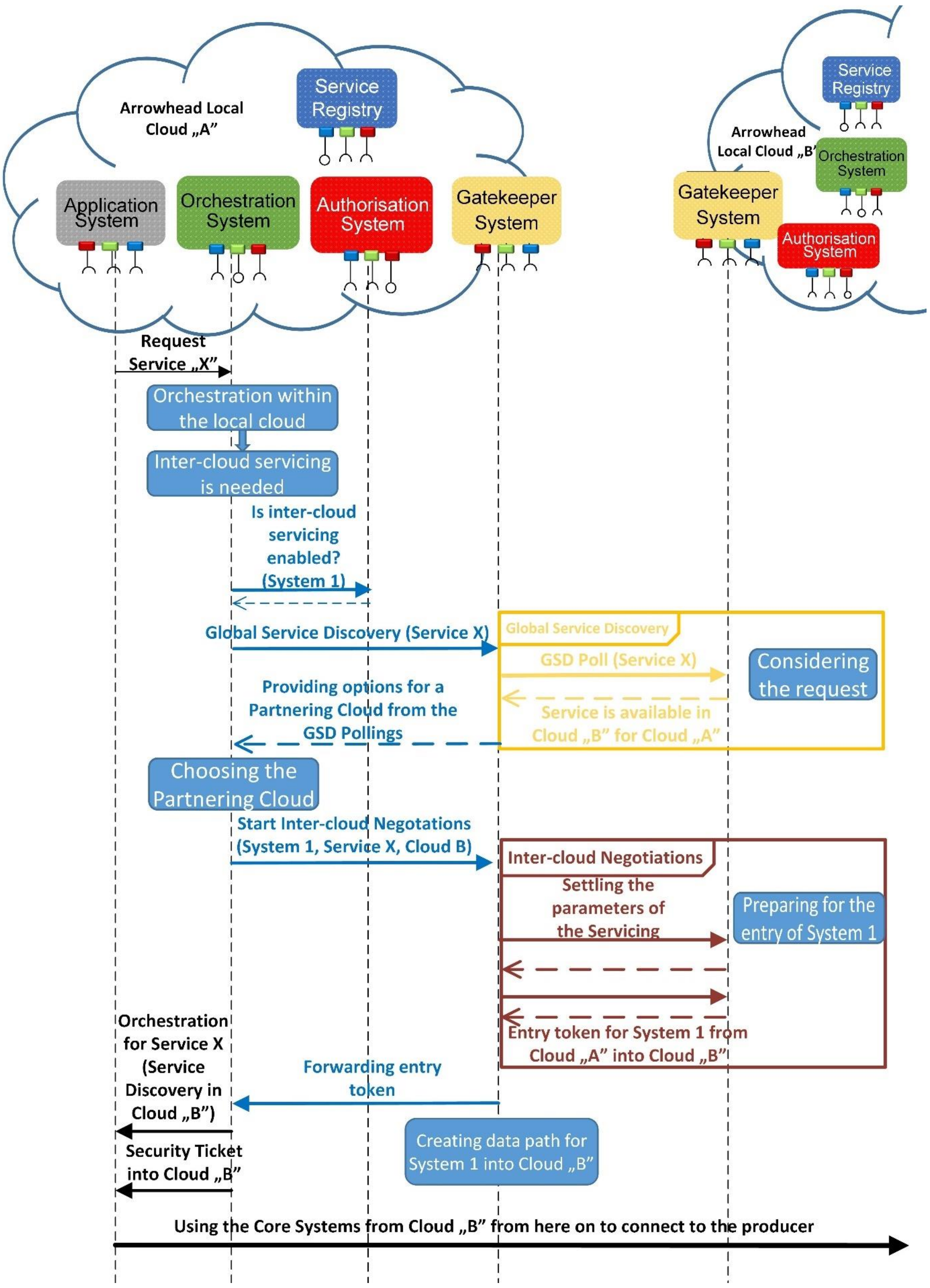

Service Request from the Application System
Local Cloud Orchestration
Global Service Discovery
Choosing the Partnering Cloud
Inter-Cloud Negotiations
Forwarding the Entry Token
Application System connects to the Partnering Cloud

Figure 10. The steps for creating inter-cloud servicing connections

The proposed process for this is sketched in Figure 10, and further detailed in Figure 11. The Gatekeeper is first contacted after the local Core Systems fail to satisfy the Service request. As the current implementation of the Core Systems do not involve such intelligent decision-making, the Orchestration process will have to be extended, see section 4.1.

1. Normal Local Cloud orchestration process begins to provide the requesting System with the most suitable Service Provider.
2. If that process fails to accommodate the need, the (revised) Orchestrator will turn to the Gatekeeper module to initiate Global Service Discovery after making the necessary preliminary assessments (i.e. checking the requester System's AAA rights for inter-cloud Service procurement).
3. The Orchestrator chooses the Partnering Cloud (where it will redirect the System) from the options provided by the GSD Services.
4. The Gatekeeper is then tasked with the negotiations of the different aspects of the transaction based on the configurations provided by the local Orchestrator Core System.

In the following sections, the elements of this inter-cloud servicing process with the Gatekeeper Core System concept will be discussed and different approaches will be proposed for the different environments in each step.



Arrowhead Local Cloud „A"
Service Registry
Application System
Orchestration System
Authorisation System
Gatekeeper System
Arrowhead Local Cloud „B"
Service Registry
Orchestration System
Gatekeeper System
Authorisation System
Request Service „X"
Orchestration within the local cloud
Inter-cloud servicing is needed
Is inter-cloud servicing enabled? (System 1)
Global Service Discovery (Service X)
Global Service Discovery
GSD Poll (Service X)
Considering the request
Service is available in Cloud „B" for Cloud „A"
Providing options for a Partnering Cloud from the GSD Pollings
Choosing the Partnering Cloud
Start Inter-cloud Negotations (System 1, Service X, Cloud B)
Inter-cloud Negotiations
Settling the parameters of the Servicing
Preparing for the entry of System 1
Entry token for System 1 from Cloud „A" into Cloud „B"
Orchestration for Service X (Service Discovery in Cloud „B")
Forwarding entry token
Security Ticket into Cloud „B"
Creating data path for System 1 into Cloud „B"
Using the Core Systems from Cloud „B" from here on to connect to the producer

Figure 11. The full process for creating inter-cloud consumer-producer relationship.

3.2. Global Service Discovery

It all starts with a node requesting a Service that cannot be served locally (as in the explored primary use cases). The Core Systems now have to check for the node’s and the Service’s authorization information whether it is possible to go inter-cloud on this matter (as seen on Figure 11).

After this, the Cloud has to find a place (another Cloud) where the Service is available for it right now. This process is basically the extension of Service Discovery, therefore called Global Service Discovery (GSD). This Service is not provided intra-cloud, but inter-cloud among the Gatekeepers.

In this section, different approaches will be provided for handling GSD. In this process, only the Gatekeeper components of the different Arrowhead Clouds have the ability to look for resources outside the boundaries of their own Cloud.

They are capable of searching for specific types of Services that are available somewhere else and the here is to provide the local Cloud with options for redirecting to a servicing partner. Only the Gatekeepers from each Cloud participate in this process. There is no authentication or identity checking in this phase as this process only sets to collect “advertisements”.

In the following Subsections, three basic approaches will be proposed, suiting different environments and requirements. Future operators of Arrowhead Clouds should have the ability to choose a methodology that fits their technological and business purposes best.

All of these scenarios use the Global Service Discovery Services (GSD) of the Gatekeeper modules, as depicted in Figure 11. This common interface would ensure that the all the different implementations can be chosen from the Arrowhead Framework models in engineering time according to the actual needs.

These GSD Services will have to operate with the proper messaging protocols and sequences, yet to be elaborated. In the GSD Poll Service, a Service Request Form will have to be implemented to provide sufficient data for the polled Local Cloud to decide whether it can positively respond to the query or have to reject it.

The GSD Publish Service will be mainly used in the Cloud-of-Cloud concept, see section 3.2.4. It will aim to provide the central entities with the current status of the publisher Local Cloud’s currently available Services.

In the following subsections, different Global Service Discovery architectures will be proposed for the different Arrowhead use case scenarios using the same definition of GSD Services.

3.2.1. Trust issues

Trust is essential in industrial automation communications (besides other factors like real-time constrains). Cyber-physical systems can be involved here, therefore Service procurement from outside the trusted Local Clouds requires prudence in every design level and implementation phases.

Therefore, it might be viable to track the trustworthiness of the "advertisers" in some cases, i.e. in environments where the Service supply and demand is highly volatile and the participants often change. This could happen in ad hoc, peer to peer based use cases, for example smart electric cars looking for charging stations for a one-time charge along the road.

Since a central Certificate Agency or Authentication Service might not be available in all implementations, this Framework has to be flexible enough to handle ad hoc, peer-to-peer based uses cases as well.

In [32], trust-based recommendations control the personal information between handheld devices. This enables unobtrusive information exchange while limiting the access to confidential information. Recommendations based on the different degrees of trust belief and disbelief of the recommender is treated differently. Basically, authentication and identity checking can be done based on previous experience with the advertiser, or it being a trustworthy contact of "my" trusted contacts.

3.2.2. Hardwired Neighborhoods

The most basic approach is hardwiring information about a certain set of other Gatekeepers into each Gatekeeper where they can to turn to for GSD. This can be based on the Service type, time of day, geographical requirements (for choosing a physically close partner), etc.

In this scenario, when a Gatekeeper is triggered to find a specific Service among the hardwired partners, it will poll all of them through the GSD Poll Service, as Figure 11 suggests. This per-request polling still supports real-time resource management, as only those partners will respond to the GSD poll that can satisfy the request instantaneously at the moment.

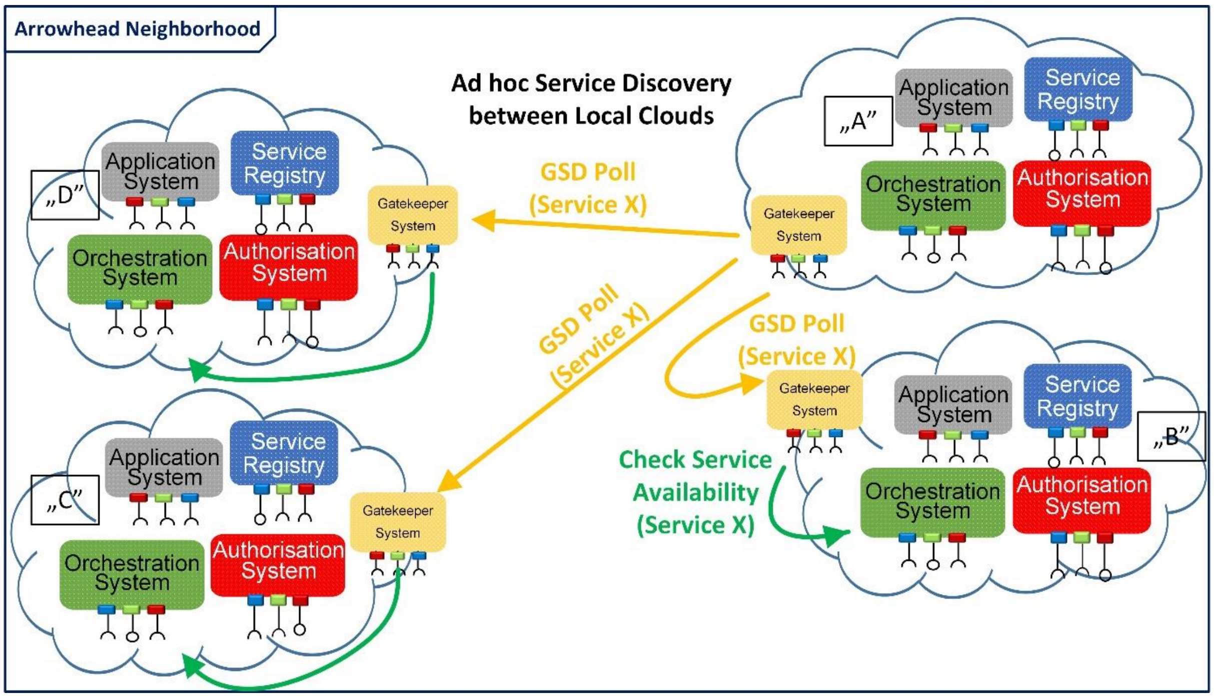


Figure 12. A hardwired neighborhood using GSD Services

What are the characteristics of this approach?

- Cloud operators have a strict oversight of their Arrowhead Clouds,
- As they can easily log the interactivity, billing and access control can be set up between operators.
- No operational risks are involved here: no need for high-level (global) authentication, identity control of the partnering clouds or trust management in any phase.
- It is highly secure: easy to implement security measures (i.e. with static configuration of firewalls).
- It fits various business purposes: contractual terms can be implemented relatively easily, SLA-s can be involved to provide proper QoS.
- However, there is absolutely no option for adaptation automatically as,
- Manual edition of the configuration is required each time anything changes.

This use case also does not scale favorably as the “per transaction polling” of all partners has a resource usage of $O\ (n^2)$ where n is the population of the Neighborhood. However, by implementing pre-determined queries directed only towards the Clouds equipped with the requested Service instance can lower the overhead.

3.2.3. Ad hoc connections

The second approach is still based on per-transaction polling, but the neighborhood is not fixed in engineering time. By letting the Gatekeepers automatically detect their neighborhood, the use cases of volatile (fast changing) environments can be served this way.

However, the administrative overhead is quite high, as “getting to know” and tracking the neighbors and preliminary assessing them for trust management purposes has to be included. These problems are to be solved, starting with the actual definition of neighborhood in this context. Authentication and identity checking is also an important but yet to be elaborated segment of this approach.

The GSD Services might also have to be augmented with proper protocols and Services in order to accommodate the ad hoc peer-to-peer discovery nature of this scenario.

3.2.4. Utilization of a cloud hierarchy

The third approach is pointing towards the creation of a dedicated inter-cloud system, where the “demand and supply” of inter-cloud service requests can meet. The main purpose of this is to take off the administrative, matchmaking and resource allocation tasks from the Arrowhead Cloud Gatekeepers and Core Services.

The Gatekeepers would only connect to a central entity while performing the GSD Poll Service and get back only one, globally optimal candidate for the Partnering Cloud, as seen in Figure 13. Using the Arrowhead nomenclature this setup can be called Cloud-of-Clouds (CoC). This centralized component would take care of the authentication and trust management functionalities as only manual registration to a CoC should be allowed (by the operators using contractual terms).

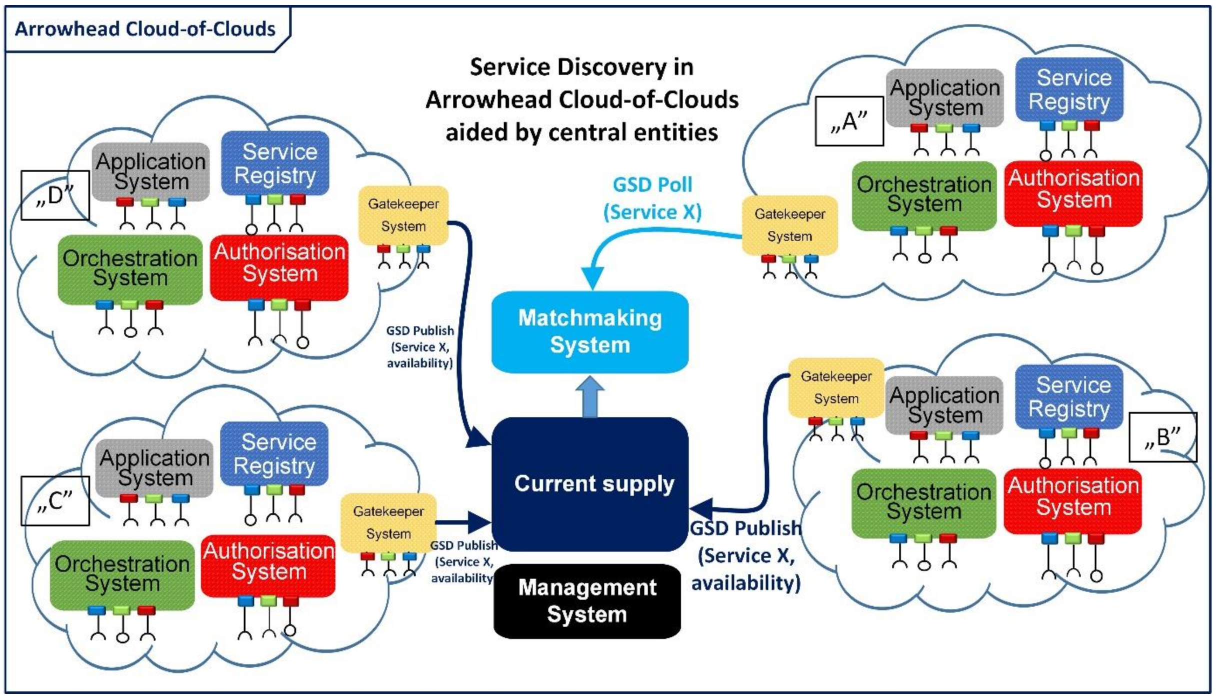


Figure 13. The proposed Cloud-of-Clouds concept

The grand challenge here is create a matchmaking system. The matchmaking system will have to provide the following functionalities:

- Continuously collect Service Request Forms from consumers and Service Supply Forms from providers;
- Processing each request by searching for adequate supplies (with matching services identification);
- Optimizing the matchmaking by creating an allocation table of matched consumer-producer couples;
- Notifying the requesting consumers where to proceed with Inter-Cloud Negotiations.

A CoC can be operated by the Arrowhead Consortium to provide the global interoperability level for the Arrowhead compatible instances of Local Clouds, or by operators creating regional (or corporal) interoperability. In the former case, this cloud architecture will have to be highly scalable, while in the latter case further levels of CoC systems will have to be created to provide true global interconnectivity (multi-tiered Cloud-of-Clouds) – as depicted in Figure 8.

3.3. Inter-cloud Negotiations

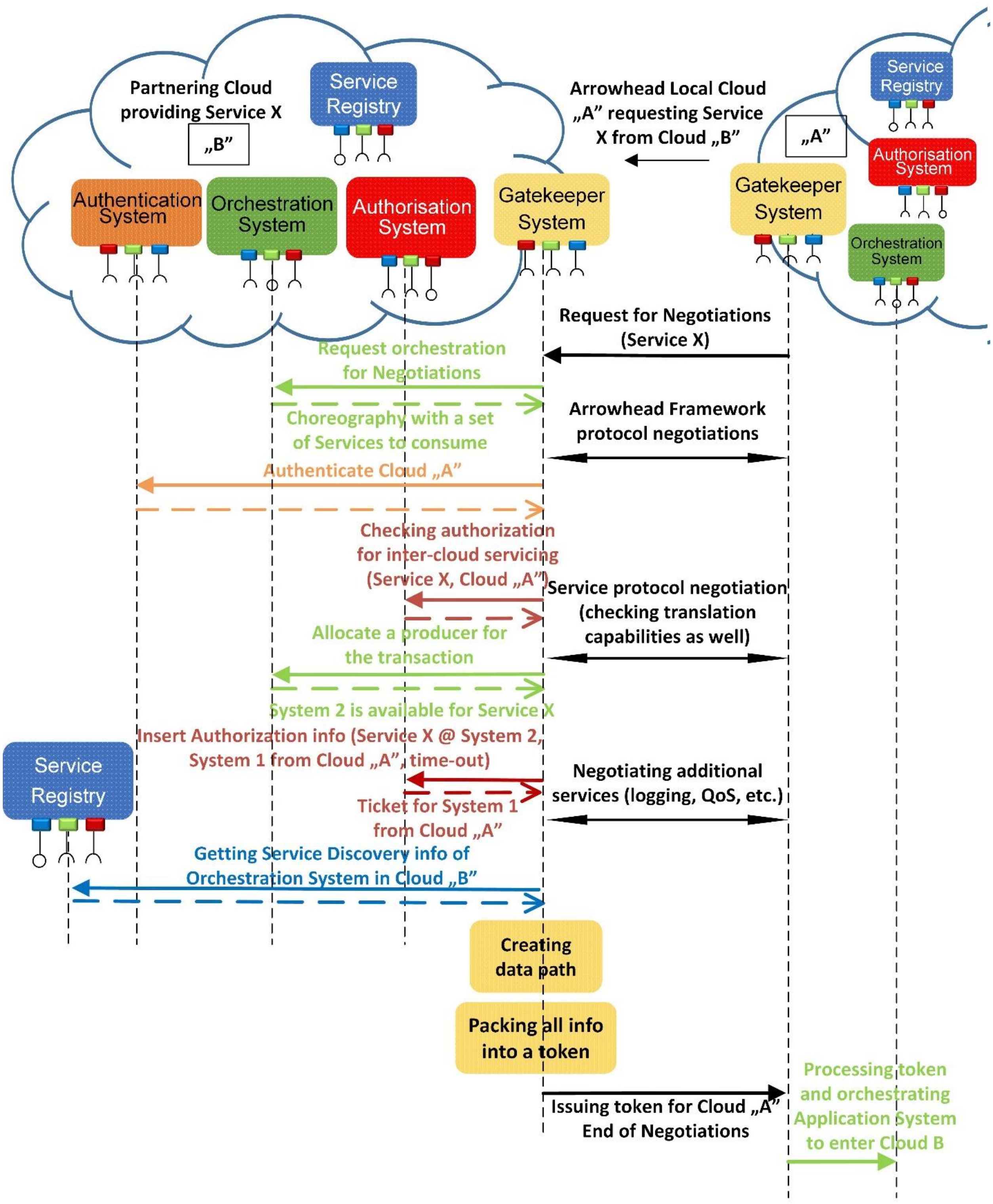
Assuming that the mutual agreement has been made between two Arrowhead Clouds to let their Systems provide a specific Service to each other, there are still a lot of “negotiations” to conduct before that can actually happen.

This process is hence to be called “Inter-cloud Negotiations” (ICN). This process has to cover a lot of aspects of the servicing. These are the technical terms that need to be settled. This process will rely on the actual implementation of the Arrowhead Framework.

At the end of this phase, the requesting System gets a token that will authenticate and navigate (using the local Service Discovery data and local Core Systems) it in the Partnering Cloud.

This process should be a sequence of the following, as seen on Figure 14:

1. Protocol Negotiations: the two Clouds have to have a matching version of the Arrowhead Framework communication protocols to “speak the same Arrowhead language”.
2. As GSD did not cover actual authentication or identity checking of the parties, these have to happen in this process before either parties would let any of its sensitive data out (i.e. to an attacker with forged identity).
3. In GSD, the Clouds only provide an “advertisement” about their resources. With the intent of actually allocating resources, local authorization and resource management processes have to be completed.
4. Having both parties prepared for the inter-cloud interaction, a secure connection can be built between the two Gatekeepers for the exchange of their inner control data.
5. The Partnering Cloud (the one that is hosting the node from the other one) will have to inject temporary authorization information into its Core Services to provide access control for the “foreigner” node.
6. The clouds (the Gatekeepers) create a data path for the nodes to interact. This is done based on the negotiations on the security expectations. The data path can be realized in several ways, in chapter 4, the different scenarios will be explored.
7. Having completed this, the Partnering Cloud’s Gatekeeper issues a token for the requester Gatekeeper with temporary entry data and shares it.
8. The Requesting Gatekeeper forwards this token to the requesting node.
9. With this token, the requesting node can access the Partnering Cloud and interact with the remote Core Systems and locate its servicing partner.



Partnering Cloud providing Service X
„B"
Service Registry
Arrowhead Local Cloud „A" requesting Service X from Cloud „B"
„A"
Service Registry
Authentication System
Orchestration System
Authorisation System
Gatekeeper System
Gatekeeper System
Authorisation System
Orchestration System
Request for Negotiations (Service X)
Request orchestration for Negotiations
Choreography with a set of Services to consume
Arrowhead Framework protocol negotiations
Authenticate Cloud „A"
Checking authorization for inter-cloud servicing (Service X, Cloud „A")
Service protocol negotiation (checking translation capabilities as well)
Allocate a producer for the transaction
System 2 is available for Service X
Insert Authorization info (Service X @ System 2, System 1 from Cloud „A", time-out)
Negotiating additional services (logging, QoS, etc.)
Ticket for System 1 from Cloud „A"
Service Registry
Getting Service Discovery info of Orchestration System in Cloud „B"
Creating data path
Packing all info into a token
Processing token and orchestrating Application System to enter Cloud B
Issuing token for Cloud „A" End of Negotiations

Figure 14. Inter-Cloud Negotiations Sequence Diagram

4. Effects on the Arrowhead Framework

4.1. Extension of the Orchestration System

The GSD and ICN processes aim to create the possibility for one System from one Local Cloud to consume a Service from a producer of a second, different Local Cloud. This process will utilize the Core Systems instances of each SoS. However, the Gatekeeper module and this proposed inter-cloud servicing methodology will mean several addition and modification to the Arrowhead Framework Core System definitions. In this Chapter, the major points will be identified.

The Mandatory Core Services have to be functionally augmented. The Authorization System will have to support inter-cloud servicing rights of the Systems with a new set of parameters, such as:

- Systems that have the right to be serviced inter-cloud;
- Services that can be procured inter-cloud;
- Services that can be provided inter-cloud;
- List of trusted Local Clouds or domains and their Services;
- Inter-cloud servicing boundaries: geographical restrains, time of day (i.e. disabled at night), etc.

The Orchestration Service will also have to gain new functionalities. These do not only stem from the inter-cloud requirements, but it is a general shortcoming of the current Framework generation. Resource allocation is not handled currently: there is a need for a central decision making entity that would allocate which Service Provider is to serve which Service Consumer. During this process, the Orchestrator (using the other Core Systems) will have to assess, which Service Provider is the best fit for a Service request (in the Local Cloud). If there is no suitable Service Provider, the GSD Services is to be called from the Gatekeepers module.

The proposed Orchestration process is depicted by Figure 15. This sequence should consist of the following steps (in accordance with the numbering on Figure 15). At the end of this process, the Orchestrator either provides a suitable local Service Provider or initiates the above described inter-cloud servicing procedures.

1. The Application System submits a Service Request with its parameters.
2. The Orchestrator begins the process by loading the orchestration configuration from the Store.
3. The Orchestrator collects the data about the requester System and the requested Service from the Service Metadata and System Configuration Stores. This will involve protocol (and translation) capabilities, available configurations for devices, preferred producers and other parameters.
4. The Orchestrator will now request a list of the Service Providers with the appropriate Service implementations (if protocol-translation is available) from the Service Registry.
5. Before further proceedings, preliminary authorization rights of the requester System and the proposed Service Providers will be collected from the Authorization Store.
6. The Orchestrator will now check which Service Provider can provide the requested servicing level (with proper QoS). The requester System will be redirected to the most suitable partner.
7. If the Orchestration process fails to satisfy the request locally, the inter-cloud servicing procedures will be initialized by turning to the Gatekeeper Services.

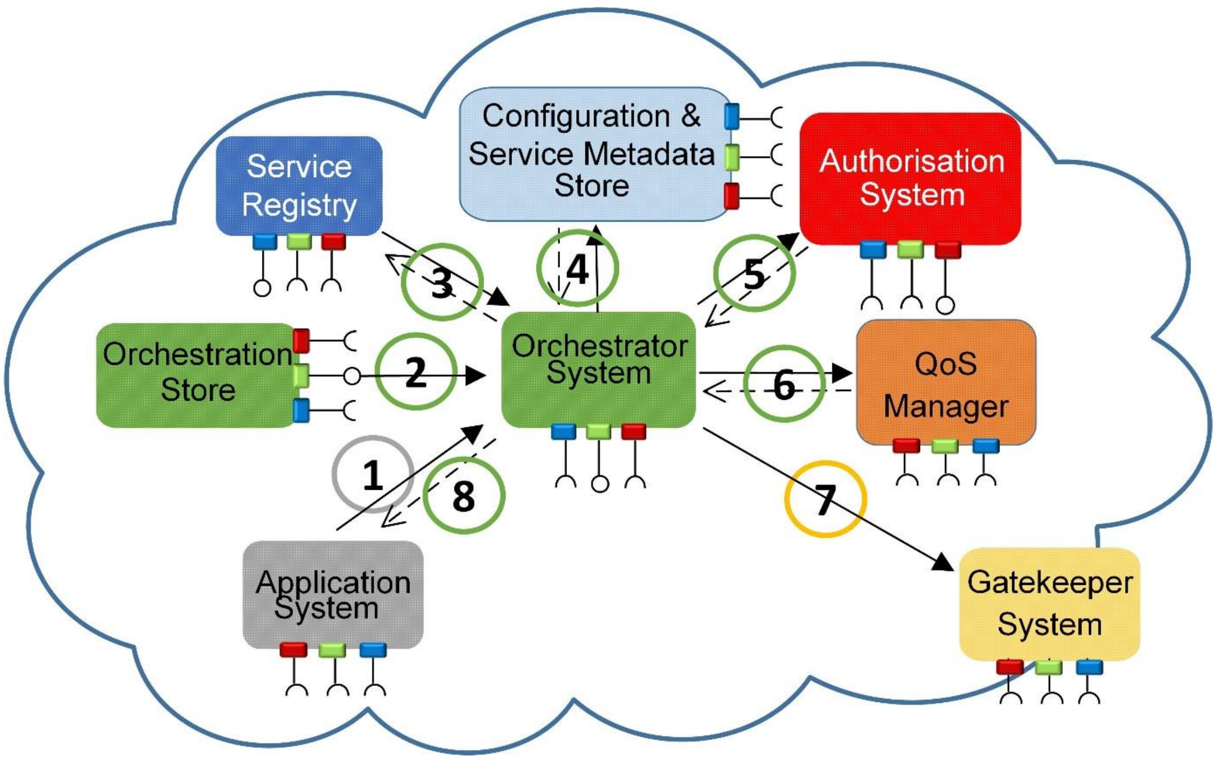


Figure 15. The proposition for the preliminary Orchestration sequence before inter-cloud servicing

This revised Orchestrator can become a very complex decision-making entity, when all elements are implemented and the most flexible behavior is expected. However, this extended procedure has to be modular and highly scalable as not all Arrowhead Local Clouds will require this freedom and run-time flexibility.

4.2. Security measures and other considerations

4.2.1. Authentication and network security

As concluded in Appendix 1, authentication (and identity verification) is essential for the gateways in the different technological domains to prevent malicious attacks. Currently, this Core Systems in not Mandatory, and only provides a general authentication via certificates. This is because the current Arrowhead Framework's primary use case is an automation system in a closed network. However, energy-constrained Systems cannot be forced to process complex certificates at each interaction, therefore it might be viable for the Arrowhead Framework to implement a ticketing system as secondary authentication (throughout a transaction).

Moreover, the creation of the data path was the final step in the ICN process (see section 3.3). However, the data path can be implemented several ways (best suitable for the use case), such as:

- creating a temporary hole in the firewall with NAT support (temporarily assigning public IP & port to the Application System),
- creation of secure VPN between the System and the Partnering Cloud,
- using a dedicated System on the boarder of the Local Clouds that are visible from the outside and can "tunnel" the communication between the producer and consumer.

Hence, this process requires interfaces towards network-security management Systems. On the other hand, these network isolation issues are currently not addressed in the Arrowhead Framework as only Local Cloud scenarios have been explored so far.

4.2.2. The Arrowhead Service Inventory

The Framework currently uses a Service Registry concept based on DNS-SD (as described in Sub-section 2.1.3.). This methodology is based on identifiers (names) of the Services and the System instances. However, considering the current use case scenarios, interoperability will be impossible to achieve with the current Framework mindset.

Imagining a large amount of Service Descriptions created by developers for their own Local Clouds, different Services with overlapping functionalities are inevitable. Hence, the current Service Discovery (and the GSD methodology based on it) will only provide Systems with the exact matching Service Description and not functionally suitable matches. This issue could be addressed by the introduction of a Service Inventory to help the developers use the pre-existing Service Description models and implementations instead of creating the same functionality once again from scratches.

However, the Global Service Discovery functionality could also be enhanced with machine-readable Service Descriptions and intelligent semantical service matching. These mostly operate on a formalized service description and provide results for an also formalized service request form using intelligent analysis on their content. Basically, the question here is: "Does this service advertisement match this service request?"

"The aim of Semantic Web Service technology is to reduce the manual discovery and usage of Web Services, by allowing software agents and applications to automatically identify, integrate and execute these Web Services to achieve the user purposes" [33].

These algorithms can operate on several levels. Traditional service matchmaking approaches are based on syntactical analysis of the service name (as currently in the Arrowhead Framework using DNS SD exact matches). With the development of semantic service specification languages, such as OWL-S [34], there are matchmaking approaches that also use the input and output parameters of the service descriptions as well. This can be further enhanced with semantical (ontology-based) analysis tools that can identify matching descriptions even if the content is syntactically different but mean the same (i.e. 'car' is equal to 'vehicle' semantically only syntactically not) [35].

The inclusion of such service description and service matchmaking solutions would require the Framework to redesign the Service Registry and the documentation procedures as well. Nonetheless, the Arrowhead Framework is facing service matchmaking issues as Local Clouds of different stakeholders will seek interactions with each other.

4.2.3. Monitoring and handling events

Finally, the oversight of the servicing does not end with the Service producer and consumer establishing a connection. Further tasks involve logging all sorts of data, processing the upcoming events, handling errors and accommodating to changes during service runtime.

These are currently handled by the local Core Systems (see section 2.1.3.). Nevertheless, this functionality is also required when a System procures a Service from another Local Cloud. All stakeholders (the Core Systems from both Clouds) have to be properly informed about the happenings and have to be able to intervene if necessary.

All of these tasks can be supported by a proper orchestration sequence pushed on the requesting System after the ICN process. The Orchestrator can provide instructions about the additional Services to be consumed with the entry token received from the Partnering Cloud. This can involve instructing the involved Systems to provide logs and reports to both Clouds. However, the supervision (the event monitoring and handling) of the servicing should belong to only one Cloud and the other Cloud should only be notified. These aspects are settled during the ICN process.

5. Conclusions

This work examined the possibilities for inter-cloud servicing in the Arrowhead Framework.

Since this contribution is becoming an essential part of the Arrowhead project, the impact of this work within that domain – at least 80 partners from SKF through Schneider and FIAT to Airbus – is potentially high.

The Arrowhead project aims to create interoperability of systems and devices in the automation and in the IoT world. This is a grand challenge as these worlds are full of heterogeneity in every sense. The project wishes to tackle this problem with its Service Oriented Architecture based approach and by using the experiences it can draw from the multi-domain pilot programs (implementations) conducted all across Europe.

So far the project focused on the local level: creating a System of Systems in Local Cloud scenarios. The current architecture provides very flexible ways of integrating devices in these closed environment as described in chapter 2. However, as one Arrowhead Local Cloud is not a viable option, there is a need for providing Services across Local Clouds.

Through chapter 3 and chapter 4 I proposed a high-level design for these interactions and examined the different use case scenarios involved. The involved processes are characterized in details and their implications are also discussed.

The key element of this concept is the Gatekeeper Core System and its Services provided intra- and inter-cloud. This module will provide Global Service Discovery functionality configured by the operator of the Local Cloud and provide the Orchestration process with suitable Service Provider Clouds. It also creates the connection to the Partnering Cloud (that will be the producer eventually) through the proposed Inter-Cloud Negotiations process.

The Framework will also have to be tailored to the new needs: the servicing leaving the borders of their trusted Local Clouds. Intelligent decision-maker entities will have to be introduced in a robust, secure and highly scalable manner. Therefore the Core Services have to be redesigned and their responsibilities and functionality boundaries reset.

Heterogeneity is also present between developers: the Framework has not yet been fully prepared to handle developers from different walks of life. This will make inter-cloud servicing nearly impossible as Service Descriptions and other documentations will not be synchronized between Local Clouds. To mend these issues, the Arrowhead Service Inventory has been proposed.

To conclude, I have proposed several possible ways and listed their major aspects and issues for providing inter-cloud servicing in the Arrowhead Framework. This work is only the tip of the iceberg mapping out the possible paths for the Arrowhead project to proceed on.

Parts of this work was accepted for publication in the V. International Conference on Wireless Communications, Vehicular Technology, Information Theory, Aerospace & Electronics Systems (Wireless VITAE), Hyderabad, India, 2015 [38].

With regard to the current endeavors, BME-TMIT is involved in the realization of this concept with its industrial partner AITIA International Inc. This cooperation will hopefully manifest in communicating Local Clouds hosted by BME and further developed by its researchers and students as thesis projects.

Appendix 1. Gateways in the different technological domains

Approach	Protocol & data conversion (coding conversion)	addressing issues, routing	caching	content filtering (access control)	Security & AAA	Link & resource management
Managed Internet Services	Protocol Tunneling: IPv6 packets in IPv4	NAT: local and global addresses; concept of VPN	web caches, proxies	proxy	Dedicated AAA servers (i.e. RADIUS); stateful firewalls: package inspection, session tracking	QoS: DiffServ, IntServ
VoIP services [24]	conversion between DSS1 and H.323; media codec conversion	Private Branch Exchange (PBX); Conversion between GT and H.323. addresses	x	PBX	AAA provided by the VoIP Gateways	separate management for the outgoing lines to PSTN, PLMN
IT Cloud Solutions [26]	Using virtual machine images for services; cloud technology drivers (i.e. Azure - Amazon conversion)	exposure: temporarily assigning public IPs to nodes; injecting routing info into partner cloud network	x	Cloud Networking Gateway: establishing data transfer between clouds	Secure VPNs is built during transfer	Service Virtualization; Grid resource management; connection isolation: using secure VPNs
Heterogenous systems [20]	Service Emulation: map a service to a (set of) services in other system; direct (one-to-one mapping) or abstract mapping (with overlapping functionality)	namespace mapping (example: OSI vs IP stack)	Application gateway: protocol translation, service provider emulation	transparency of the gateway	x	Performance Transparency (efficiency, turnaround time) is affected by entities in the data path
DARPA mobile ad hoc vehicular networks [21]	Generic Packet Encapsulation (protocol tunneling)	maintaining topology information with Simple Network Mgmt. Prot.	data is cached until successful transmission	x	Key management (generation, revocation, distribution): Internet Key Exchange Protocol IPSec, TLS	QoS with DiffServ middleware

References

[1] Evans, Dave (2011): *The Internet of Things – How the Next Evolution of the Internet is Changing Everything*, Cisco White Paper. Available: http://www.iotsworldcongress.com/documents/4643185/0/IoT_IBSG_0411FINAL+Cisco.pdf Accessed: 2015. 10. 19.

[2] *The Arrowhead Project website*. Available: www.arrowhead.eu Accessed: 2015. 10. 10.

[3] The GSM Alliance (2015): *The Mobile Economy 2015*. Available: http://www.gsmamobileeconomy.com/GSMA_Global_Mobile_Economy_Report_2015.pdf Accessed: 2014. 10. 01.

[4] S. Ziegler and C. Crettaz (2014): *IPv6 as a global addressing scheme and integrator for the Internet of Things and the Cloud*. Advanced Information Networking and Applications Workshop (WAINA), pp. 797–802, May 2014.

[5] Google (s.a.): *Global IPv6 statistics*. Available: https://www.google.com/intl/en/ipv6/statistics.html Accessed: 2015. 10. 17.

[6] *Electronic Components and Systems for European Leadership (ECSEL) Undertaking website*. Available: http://www.ecsel-ju.eu/web/index.php Accessed: 2015. 09. 10.

[7] *The Arrowhead framework.* The Arrowhead Project Website. Available: http://www.arrowhead.eu/about/arrowhead-common-technology/arrowhead-framework/ Accessed: 2015. 09.02.

[8] Varga, P.; Blomstedt, F.; Ferreira, L. L; Eliasson, J.; Johansson, M. and Delsing, J. (2015): *Making System of Systems Interoperable – The Core Components of the Arrowhead Technology Framework*. Submitted to the Journal of Network and Computer Applications, Special Issue on Engineering Future Interoperable and Open IoT Systems.

[9] Blomsted, F.; Ferreira, L. L.; Klisics, M.; Chrysoulas, C.; Martinez de Soria, I.; Morin, B.; Zabasta, A.; Eliasson, J.; Johansson, M. and Varga, P. (2014): *The Arrowhead Approach for SOA Application Development and Documentation*. Proceedings of the 40th Annual Conference on Industrial Electronics, Dallas, pp. 2631-2637.

[10] Lopez, Pablo, et al. (2013): *Scalable Oriented-Service Architecture for Heterogeneous and Ubiquitous IoT Domains*. Preprint. ISBN 1311.4293.

[11] Siemens: *Simatic – The Product family for Automation*, website. Available: http://w3.siemens.com/mcms/topics/en/simatic/pages/default.aspx Accessed: 2015. 09.21.

[12] The OpenGroup (2013): *Service Oriented Architecture- What is SOA?* Available: http://www.opengroup.org/soa/source-book/soa/soa.htm Accessed: 2015. 09. 20.

[13] Gartner (2003): *Service-Oriented Architecture Scenari*o. Available: https://www.gartner.com/doc/391595/serviceoriented-architecture-scenario Accessed: 2015. 10. 20.

[14] Michael Bell (2008*): Introduction to Service-Oriented Modeling*. In: Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. ISBN 978-0-470-4111-3.

[15] Valipour, M. H; Amirzafari, B.; Maleki, K. N. and Daneshpour N. (2009): *A brief survey of software architecture concepts and service oriented architecture*. 2nd IEEE International Conference on Computer Science and Information Technology. pp. 34–38.

[16] Crawford, C. H.; Bate, G. P.; Cherbakov, L.; Holley, K. and Toscanos, C. (2005): *Towards an on demand service-oriented architecture*. IBM Systems Journal, Vol. 44, No. 1. pp. 81-107.

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

[18] Cheshire, S.; Krochmal M. (2013): *DNS-Based Service Discovery*. RFC 6763. Available: https://tools.ietf.org/html/rfc6763 Accessed: 2015. 10. 20.

[19] R. T. Fielding (2000): *Architectural styles and the design of network-based software architectures*. Ph. D dissertation, University of California, Irvine, USA.

[20] Kalyanasundaram, P.; Sethi, A. S. (1994): *Interoperability Issues in Heterogeneous Network Management.* Journal of Network and Systems Management, Vol. 2, No. 2, pp. 169–193.

[21] DaSilva. L. A.; Midkiff, S. F; Park, J. S.; Hadjichristofi, G.C; Davis, N. J.; Kaustubh, S. P. and Lin, T. (2004): *Network Mobility and Protocol Interoperability in Ad Hoc Networks.* IEEE Communications Magazine, Vol. 42, No. 11, pp. 88–96.

[22] Ingham, K.; Forrest, S. (2002). *A history and survey of network firewalls.* University of New Mexico, USA. Available: http://www.cs.unm.edu/~moore/tr/02-12/firewall.pdf Accessed: 2015. 09. 16.

[23] Safari Books Online: *VoIP in a Campus Network*. Available: https://www.safaribooksonline.com/library/view/ccnp-bcmsn-quick/1587053136/ch07.html Accessed: 2014. 10. 01.

[24] *BME TMIT Infocommunications Laboratory Course Material* (2014). Available: http://qosip.tmit.bme.hu/foswiki/bin/view/Meres/VoIP Accessed: 2015. 10. 01.

[25] *OpenStack*. [Online]. Available: http://www.openstack.org Accessed: 2015. 09. 13.

[26] Mechtri. M.; Djamal, Z.; Zekri, E. and Marshall, I.J. (2013): *Inter-Cloud Networking Gateway Architecture.* Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference Paper, Volume: 2.

[27] ONF: *Software Defined Networking. Definitions*. Website. Available: https://www.opennetworking.org/sdn-resources/sdn-definition Accessed: 2015. 10. 05.

[28] Y. Demchenko, C. Laat, C. Lee (2014): *Federated Access Control in Heterogeneous Intercloud Environment- Basic models and Architecture Patterns.* 2014 IEEE International Conference on Cloud Engineering.

[29] OPC Foundation (2010): *OPC Unified Architecture Specification*, Standard IEC 624 541.

[30] Shelby, Z. et al (2014): *The Constrained Application Protocol (CoAP).* RFC 7252. Available: https://tools.ietf.org/html/rfc7252 Accessed: 2015. 10. 21.

[31] Saint-Andre, P. (2011): *Extensible Messaging and Presence Protocol (XMPP).* RFC 6120. Available: https://tools.ietf.org/html/rfc6120 Accessed: 2015. 10. 21.

[32] B. Shand, N. Dimmock, and J. Bacon (2003): *Trust for Ubiquitous, Transparent Collaboration*. IEEE International Conference on Pervasive Computing and Communications (PERCOM), pp. 153–160.

[33] Pakari, S.; Kheirkhah, E. and Jalali, M. (2014): *A Novel Approach: A Hybrid Semantic Matchmaker for Service Discovery in Service Oriented Architecture*. International Journal of Network Security & Its Applications, Vol. 6, No. 1, pp. 37-48.

[34] Hobbs, J; Narayanan S. (s.a.): *OWL-S: Semantic Markup for Web Services*. Available. http://www.ai.sri.com/~daml/services/owl-s/1.2/overview/ Accessed: 2015. 10. 05.

[35] Yau S. S.; Liu, J.: (s.a.): *A Service Matchmaking Approach for Service-Oriented Architecture Based on Service Functionalities.* Manuscript, Arizona State University, USA. Available: http://dpse.eas.asu.edu/papers/F-Match.pdf Accessed: 2015. 10. 05

[36] ITU-T (2014): *H.323 Packet-based multimedia communications systems*. Available: http://www.itu.int/rec/T-REC-H.323/en/ Accessed: 2015. 10. 26.

[37] J. Rosenberg et al (2002): *SIP: Session Initiation Protocol*. RFC 3261. Available: http://www.itu.int/rec/T-REC-H.323/en/ Accessed: 2015. 10. 26.

[38] Varga, Pál; Hegedűs, Csaba (2015): *Service Interaction through Gateways for Inter-Cloud Collaboration within the Arrowhead Framework*. Submitted to the V. International Conference on Wireless Communications, Vehicular Technology, Information Theory, Aerospace & Electronics Systems (Wireless VITAE), Hyderabad, India, 2015.